

Entwicklerhandbuch

AWS Cloud Development Kit (AWS CDK) v2



Version 2

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

AWS Cloud Development Kit (AWS CDK) v2: Entwicklerhandbuch

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

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist der AWS CDK?	1
Vorteile des AWS CDK	2
Beispiel für AWS CDK	5
AWS CDK features	10
Das AWS CDK GitHub Repository	10
Die AWS CDK API-Referenz	10
Das Construct-Programmiermodell	10
Der Construct Hub	11
Nächste Schritte	11
Weitere Informationen	11
CDK-Kernkonzepte	13
AWS CDK und IaC	13
AWS CDK und AWS CloudFormation	13
AWS CDK und Abstraktionen	14
Erfahren Sie mehr über die AWS CDK Kernkonzepte	14
Interaktion mit dem AWS CDK	14
Entwickeln mit dem AWS CDK	14
Bereitstellung mit dem AWS CDK	15
Weitere Informationen	15
Sprachen	15
Projekte	18
Universale Dateien und Ordner	18
Sprachspezifische Dateien und Ordner	19
Apps	31
Definieren von Apps	32
Der Konstruktbaum	33
Der Anwendungslebenszyklus	35
Stacks	39
Stapel definieren	40
Arbeiten mit -Stacks	47
Konstrukte	54
Die Construct-Bibliothek	55
Konstrukte definieren	59
Mit Konstrukten arbeiten	68

Arbeiten mit Konstrukten von Drittanbietern	74
Weitere Informationen	83
Umgebungen	83
Weitere Informationen	85
Bootstrapping	85
Was ist Bootstrapping?	86
Wie funktioniert Bootstrapping?	86
Weitere Informationen	86
Ressourcen	87
Konfiguration von Ressourcen mithilfe von Konstrukten	87
Ressourcen referenzieren	90
Physische Namen der Ressourcen	100
Übergeben von eindeutigen Ressourcen-Identifikatoren	102
Erteilen von Berechtigungen zwischen Ressourcen	105
Metriken und Alarmer für Ressourcen	108
Netzwerkdatenverkehr	111
Behandlung von Ereignissen	115
Richtlinien zur Entfernung	116
IDs	121
Konstrukt-IDs	122
Pfade	125
Eindeutige IDs	126
Logische IDs	127
Token	128
Token und Token-Kodierungen	130
Zeichenfolgenkodierte Token	132
Listencodierte Token	134
Nummernkodierte Token	134
Lazy Values	134
Konvertieren in JSON	137
Parameter	138
Über Parameter	138
Parameter definieren	139
Verwenden von Parametern	141
Bereitstellung mit Parametern	143
Tagging	147

Verwenden von Markierungen	147
Tag-Prioritäten	149
Optionale Eigenschaften	150
Beispiel	154
Markieren einzelner Konstrukte	157
Objekte	160
Komponenten im Detail	160
Komponententypen	161
Amazon S3-Komponenten	161
Docker-Image-Komponenten	174
AWS CloudFormation Ressourcenmetadaten	185
Berechtigungen	186
Auftraggeber	186
Gewährungen	187
Rollen	189
Ressourcenrichtlinien	196
Verwenden externer IAM-Objekte	197
Kontext	198
Quellen von Kontextwerten	200
Context-Methoden	201
Kontext anzeigen und verwalten	201
AWS CDK Toolkit-Flagge <code>--context</code>	203
Beispiel	203
Feature-Flags	208
Zurücksetzen auf das Verhalten v1	208
Aspekte	209
Detailaspekte	210
Beispiel	212
Erste Schritte	215
Voraussetzungen	215
Schritt 1: Erstellen eines AWS-Konto	217
Schritt 2: Programmgesteuerten Zugriff konfigurieren	217
Starten einer - AWS Zugriffsportalsitzung	219
Schritt 3: Installieren der AWS CDKCLI	220
Schritt 4: Bootstrappen Ihrer Umgebung	221
Optionale AWS CDK Tools	221

Nächste Schritte	222
Weitere Informationen	222
Deine erste AWS CDK App	222
Über dieses Tutorial	223
Schritt 1: Erstellen Sie die App	224
Schritt 2: Erstellen Sie die App	226
Schritt 3: Listet die Stapel in der App auf	227
Schritt 4: Einen Amazon S3 S3-Bucket hinzufügen	227
Schritt 5: Synthetisieren Sie eine Vorlage AWS CloudFormation	231
Schritt 6: Stellen Sie Ihren Stack bereit	233
Schritt 7: Ändern Sie Ihre App	233
Schritt 8: Zerstören der Ressourcen der App	240
Nächste Schritte	240
Arbeiten mit der AWS CDK	242
Importieren der AWS Konstruktbibliothek	242
Die AWS CDK API-Referenz zu	244
Schnittstellen im Vergleich zu Konstruktclassen	244
Verwalten von Abhängigkeiten	245
Vergleich von AWS CDK in TypeScript mit anderen Sprachen	246
Importieren eines Moduls	247
Instanzieren eines Konstrukts	250
Zugreifen auf Mitglieder	253
Aufzählungskonstanten	254
Objektschnittstellen	255
In TypeScript	256
Erste Schritte mit TypeScript	257
Erstellen eines Projekts	258
Verwenden von lokal tsc und cdk	258
Verwalten von Modulen AWS der Construct Library	260
Verwalten von Abhängigkeiten in TypeScript	261
AWS CDK idiomata in TypeScript	265
Erstellen, Synthetisieren und Bereitstellen	266
In JavaScript	267
Erste Schritte mit JavaScript	268
Erstellen eines Projekts	268
Verwenden von Local cdk	258

Verwalten von Modulen AWS der Construct Library	270
Verwalten von Abhängigkeiten in JavaScript	272
AWS CDK idioms in JavaScript	275
Synthetisieren und Bereitstellen	277
Verwenden von TypeScript Beispielen mit JavaScript	278
Migrieren zu TypeScript	281
In Python	282
Erste Schritte mit Python	282
Erstellen eines Projekts	284
Verwaltung von AWS Construct Library-Modulen	285
Verwaltung von Abhängigkeiten in Python	287
AWS CDK Redewendungen in Python	289
Synthetisieren und bereitstellen	292
In Java	293
Erste Schritte mit Java	294
Erstellen eines Projekts	294
Verwalten von Modulen AWS der Construct Library	295
Verwalten von Abhängigkeiten in Java	296
AWS CDK idioms in Java	297
Erstellen, Synthetisieren und Bereitstellen	299
In C#	300
Erste Schritte mit C#	301
Erstellen eines Projekts	301
Verwalten von Modulen AWS der Construct Library	302
Verwalten von Abhängigkeiten in C#	303
AWS CDK idioms in C#	306
Erstellen, Synthetisieren und Bereitstellen	308
In Go	309
Erste Schritte mit Go	310
Erstellen eines Projekts	310
Verwalten von Modulen AWS der Construct Library	311
Verwalten von Abhängigkeiten in Go	311
AWS CDK idiomias in Go	312
Erstellen, Synthetisieren und Bereitstellen	314
Migrieren von AWS CDK v1 zu AWS CDK v2	316
Neue Voraussetzungen	318

Upgrade von der AWS CDK v2-Entwicklervorschau	319
Migrieren von AWS CDK v1 zu CDK v2	319
Aktualisieren auf ein aktuelles v1	320
Aktualisieren von Feature-Flags	320
Kompatibilität mit CDK Toolkit	321
Aktualisieren von Abhängigkeiten und Importen	321
Testen Ihrer migrierten App vor der Bereitstellung	327
Fehlerbehebung	328
Suchen von v1-Stacks	329
Migrieren Sie zum AWS CDK	330
Wie funktioniert die Migration	330
Vorteile von CDK Migrate	331
Überlegungen	332
Allgemeine Überlegungen	332
Überlegungen bei der Migration von einer Vorlage AWS CloudFormation	333
Überlegungen bei der Migration von bereitgestellten Ressourcen	333
Voraussetzungen	334
Beginnen Sie mit CDK Migrate	334
Migrieren Sie von einem AWS CloudFormation Stack	335
Aus einer AWS CloudFormation Vorlage migrieren	336
Migrieren Sie von einer AWS SAM Vorlage	337
Migrieren Sie von bereitgestellten Ressourcen	337
Verwenden Sie Filter	337
Mit dem IaC-Generator nach Ressourcen suchen	338
Löst Eigenschaften auf, die nur über Schreibzugriff verfügen	338
Die Datei migrate.json	341
Verwalten und implementieren Sie Ihre CDK-App	341
Bereiten Sie sich auf die Bereitstellung vor	341
Stellen Sie Ihre CDK-App bereit	342
Konfigurieren Sie Sicherheitsanmeldedaten	344
Voraussetzungen	344
Erstellen Sie einen AWS-Konto Benutzer mit Administratorrechten	344
Bestimmen Sie Ihre Methode zur Erstellung und Verwaltung von Benutzern	345
Installieren Sie das AWS CLI	345
Installieren Sie den AWS CDKCLI	345
Wie konfiguriert man Sicherheitsanmeldedaten	346

Zusätzliche Informationen	346
IAM Identity Center-Benutzer	347
Konfigurieren Sie das AWS CLI für die Verwendung von IAM Identity Center	347
Verwenden Sie die AWS CLI , um kurzfristige Anmeldeinformationen zu erhalten	348
Konfigurieren Sie kurzfristige Anmeldeinformationen manuell	348
Verwenden Sie kurzfristige Anmeldeinformationen mit dem CDK CLI	348
Beispiel: Verwenden Sie den, AWS CLI um Sicherheitsanmeldedaten zu konfigurieren	348
Umgebungen konfigurieren	353
Wo Sie Umgebungen angeben können	353
Anmeldeinformationen und Konfigurationsdateien	353
env-Eigenschaft des Stack-Konstrukts	354
Priorität der Umgebung mit dem AWS CDK	354
Wann müssen Umgebungen angegeben werden	354
Geben Sie Umgebungen bei der Vorlagensynthese an	355
Geben Sie die Umgebungen bei der Stack-Bereitstellung an	355
Wie spezifiziert man Umgebungen mit dem AWS CDK	356
Spezifizieren Sie hartcodierte Umgebungen für jeden Stack	356
Geben Sie Umgebungen mithilfe von Umgebungsvariablen an	358
Spezifizieren Sie Umgebungen anhand Ihrer Anmeldeinformationen und Konfigurationsdateien mit dem CDK CLI	361
Überlegungen bei der Konfiguration von Umgebungen mit dem AWS CDK	362
Beispiele	362
Synthetisieren Sie eine umgebungsunabhängige Vorlage aus einem CDK-Stack CloudFormation	362
Verwenden Sie Logik, um Umgebungsinformationen bei der Vorlagensynthese zu ermitteln	365
Bootstrap deine Umgebung	371
Wie booten Sie Ihre Umgebung	371
Verwenden Sie das CDK CLI	371
Verwenden Sie ein beliebiges AWS CloudFormation Tool	372
Wann sollten Sie Ihre Umgebung booten	374
Aktualisiere deinen Bootstrap-Stack	374
Passen Sie das Bootstrapping an	374
Bootstrapping mit CDK Pipelines	375
Schützen Sie Ihren Bootstrap-Stack vor dem Löschen	375
Versionsverlauf der Bootstrap-Vorlage	376

Führen Sie ein Upgrade von einer älteren auf eine moderne Bootstrap-Vorlage durch	380
Gehen Sie auf die Ergebnisse von Security Hub ein	381
[KMS.2] IAM-Prinzipale sollten keine IAM-Inline-Richtlinien haben, die Entschlüsselungsaktionen für alle KMS-Schlüssel zulassen	382
Überlegungen	384
Passen Sie das Bootstrapping an	385
Verwenden Sie das CDKCLI, um Bootstrapping anzupassen	385
Ändern Sie die Standard-Bootstrap-Vorlage	388
Der Bootstrap-Vorlagenvertrag	389
Konfiguration und Anpassung der Synthese	392
Konfigurieren Sie die CDK-Stack-Synthese	392
Synthetisieren Sie einen CDK-Stack	394
Passen Sie die CDK-Stack-Synthese an	394
Ändern Sie den Qualifier	395
Ressourcennamen ändern	396
Entwickeln von AWS CDK Anwendungen	403
Anpassen von Konstrukten	403
Verwenden von Escape-Schraffuren	403
Escape-Schraffuren entfernen	410
Rohüberschreibungen	412
Benutzerdefinierte Ressourcen	414
Abrufen des Umgebungswerts	415
CloudFormation Wert abrufen	416
Eine AWS CloudFormation Vorlage importieren	417
Eine Vorlage importieren	418
Zugreifen auf importierte Ressourcen	423
Parameter ersetzen	426
Andere Vorlagenelemente	427
Verschachtelte Stacks	428
SSM-Wert abrufen	431
Lesen von Systems Manager-Werten zur Bereitstellungszeit	432
Systems Manager-Werte zur Synthetisierungszeit lesen	434
Schreiben von Werten in Systems Manager	436
Secrets-Manager-Wert abrufen	436
CloudWatch Alarm einstellen	439
Verwenden einer vorhandenen Metrik	439

Erstellen Ihrer eigenen Metrik	440
Erstellen des Alarms	441
Kontextwert abrufen	444
Angaben von Kontextvariablen	444
Abrufen von Kontextvariablenwerten	445
Verwenden von Ressourcen aus der CloudFormation öffentlichen Registrierung	446
Aktivieren einer Drittanbieter-Ressource in Ihrem Konto und Ihrer Region	447
Hinzufügen einer Ressource aus der AWS CloudFormation öffentlichen Registrierung zu Ihrer CDK-App	450
Bereitstellen von AWS CDK Anwendungen	452
Richtliniengültigkeit	452
Richtliniengültigkeit	452
Für Anwendungsentwickler	453
Für Plugin-Autoren	456
CDK Pipelines erstellen	458
Bootstrap für Ihre Umgebungen AWS	459
Initialisieren Sie ein Projekt	462
Definieren Sie eine Pipeline	464
Phasen der Bewerbung	473
Bereitstellungen testen	485
Sicherheitshinweise	494
Fehlerbehebung	495
Bewährte Methoden	497
Bewährte Methoden für Organisationen	499
Bewährte Methoden für die Codierung	500
Starten Sie einfach und fügen Sie Komplexität nur hinzu, wenn Sie sie benötigen	501
Abstimmung mit dem AWS Well-Architected Framework	501
Jede Anwendung beginnt mit einem einzigen Paket in einem einzigen Repository	502
Verschieben von Code in Repositorys basierend auf dem Codelebenszyklus oder der Teameigentümerschaft	502
Infrastruktur- und Laufzeitcode live im selben Paket	503
Bewährte Methoden für die Erstellung von	503
Modell mit Konstrukten, Bereitstellung mit Stacks	504
Konfigurieren mit Eigenschaften und Methoden, nicht mit Umgebungsvariablen	504
Komponententest Ihrer Infrastruktur	504
Ändern Sie die logische ID von zustandsbehafteten Ressourcen nicht	505

Konstrukte reichen für Compliance nicht aus	505
Bewährte Methoden für Anwendungen	506
Treffen Sie Entscheidungen zur Generierungszeit	506
Verwenden Sie generierte Ressourcennamen, keine physischen Namen	506
Definieren von Entfernungsrichtlinien und Protokollaufbewahrung	507
Trennen Sie Ihre Anwendung gemäß den Bereitstellungsanforderungen in mehrere Stacks	508
Bekennen Sie sich <code>cdk.context.json</code> , um nicht deterministisches Verhalten zu vermeiden	508
Lassen Sie die Rollen und Sicherheitsgruppen AWS CDK verwalten	510
Modellieren aller Produktionsphasen im Code	510
Alles messen	511
AWS CDKCLIBefehlsreferenz	512
Verwendung	512
Befehle	512
Globale Optionen	513
Bereitstellung und Konfiguration von Optionen	519
Optionen an der Befehlszeile übergeben	519
Boolesche Werte übergeben	519
cdk ack	520
Verwendung	520
Argumente	520
Optionen	520
Beispiele	521
cdk bootstrap	521
Verwendung	522
Argumente	522
Optionen	522
Beispiele	528
cdk context	529
Verwendung	529
Optionen	529
cdk deploy	530
Verwendung	530
Argumente	530
Optionen	531

Beispiele	538
cdk destroy	541
Verwendung	541
Argumente	541
Optionen	541
Beispiele	542
cdk diff	542
Verwendung	542
Argumente	542
Optionen	543
Beispiele	544
cdk docs	545
Verwendung	545
Optionen	545
Beispiele	545
cdk doctor	545
Verwendung	546
Optionen	546
Beispiele	546
cdk import	546
Verwendung	548
Argumente	548
Optionen	548
cdk init	549
Verwendung	549
Argumente	550
Optionen	550
Beispiele	551
cdk list	551
Verwendung	551
Argumente	551
Optionen	552
Beispiele	552
cdk metadata	553
Verwendung	553
Argumente	553

Optionen	553
cdk migrate	553
Verwendung	554
Optionen	554
Beispiele	557
cdk notices	558
Verwendung	558
Optionen	558
Beispiele	559
cdk synth	560
Verwendung	560
Argumente	561
Optionen	561
Beispiele	562
cdk watch	562
Verwendung	563
Argumente	563
Optionen	564
Beispiele	566
AWS CDK Referenz	568
API-Referenz	568
Versionsverwaltung	568
AWS CDKCLI Kompatibilität	569
AWS Versioning der Konstruktbibliothek	570
Stabilität der Sprachbindung	570
Tutorials	572
Serverloses Hello World	572
Voraussetzungen	574
Schritt 1: Erstellen Sie ein CDK-Projekt	574
Schritt 2: Erstellen Sie Ihre Lambda-Funktion	581
Schritt 3: Definieren Sie Ihre Konstrukte	584
Schritt 4: Bereiten Sie Ihre Anwendung für die Bereitstellung vor	596
Schritt 5: Ihre Anwendung bereitstellen	596
Schritt 6: Interagieren Sie mit Ihrer Anwendung	605
Schritt 7: Löschen Sie Ihre Anwendung	605
Fehlerbehebung	606

Eine App mit mehreren Stacks erstellen	607
Bevor Sie beginnen	608
Optionalen Parameter hinzufügen	609
Definieren Sie die Stack-Klasse	613
Erstellen Sie zwei Stack-Instanzen	617
Synthetisieren Sie den Stack und stellen Sie ihn bereit	620
Bereinigen	621
Beispiele	622
ECS	622
Das Verzeichnis erstellen und das initialisieren AWS CDK	624
Einen Fargate-Dienst erstellen	625
Bereinigen	629
AWS CDK Beispiele	629
Tools	630
AWS CDK Toolkit	630
Toolkit-Befehle	631
Optionen und ihre Werte angeben	631
Integrierte Hilfe	632
Versionsberichterstattung	632
Authentifizierung mit AWS	633
Angabe der Region und anderer Konfigurationen	636
Den App-Befehl angeben	637
Stacks angeben	638
Bootstrapping für Ihre Umgebung AWS	639
Eine neue App erstellen	641
Stapel auflisten	642
Synthetisieren von Stacks	642
Stacks bereitstellen	643
Stapel vergleichen	648
Importieren vorhandener Ressourcen in einen Stack	650
Konfiguration () cdk.json	651
AWS Toolkit für VS-Code	655
AWS SAM -Integration	655
Konstrukte testen	656
Erste Schritte	656
Der Beispielstapel	659

Die Lambda-Funktion	667
Ausführen von Tests	667
Fein abgestufte Behauptungen	668
Matcher	675
Erfassen	682
Snapshot-Tests	685
Tipps für Tests	690
Sicherheit	691
Identity and Access Management	691
Zielgruppe	692
Authentifizierung mit Identitäten	692
Compliance-Validierung	696
Ausfallsicherheit	697
Sicherheit der Infrastruktur	698
Fehlerbehebung	699
OpenPGP-Schlüssel	708
Aktuelle Schlüssel	708
AWS CDK OpenPGP-Schlüssel	708
jsii OpenPGP-Schlüssel	709
Historische Schlüssel	710
AWS CDK OpenPGP-Schlüssel (2022-04-07)	711
jsii OpenPGP-Schlüssel (2022-04-07)	712
AWS CDK OpenPGP-Schlüssel (2018-06-19)	713
jsii OpenPGP-Schlüssel (2018-08-06)	714
Dokumentverlauf	716
.....	dccxviii

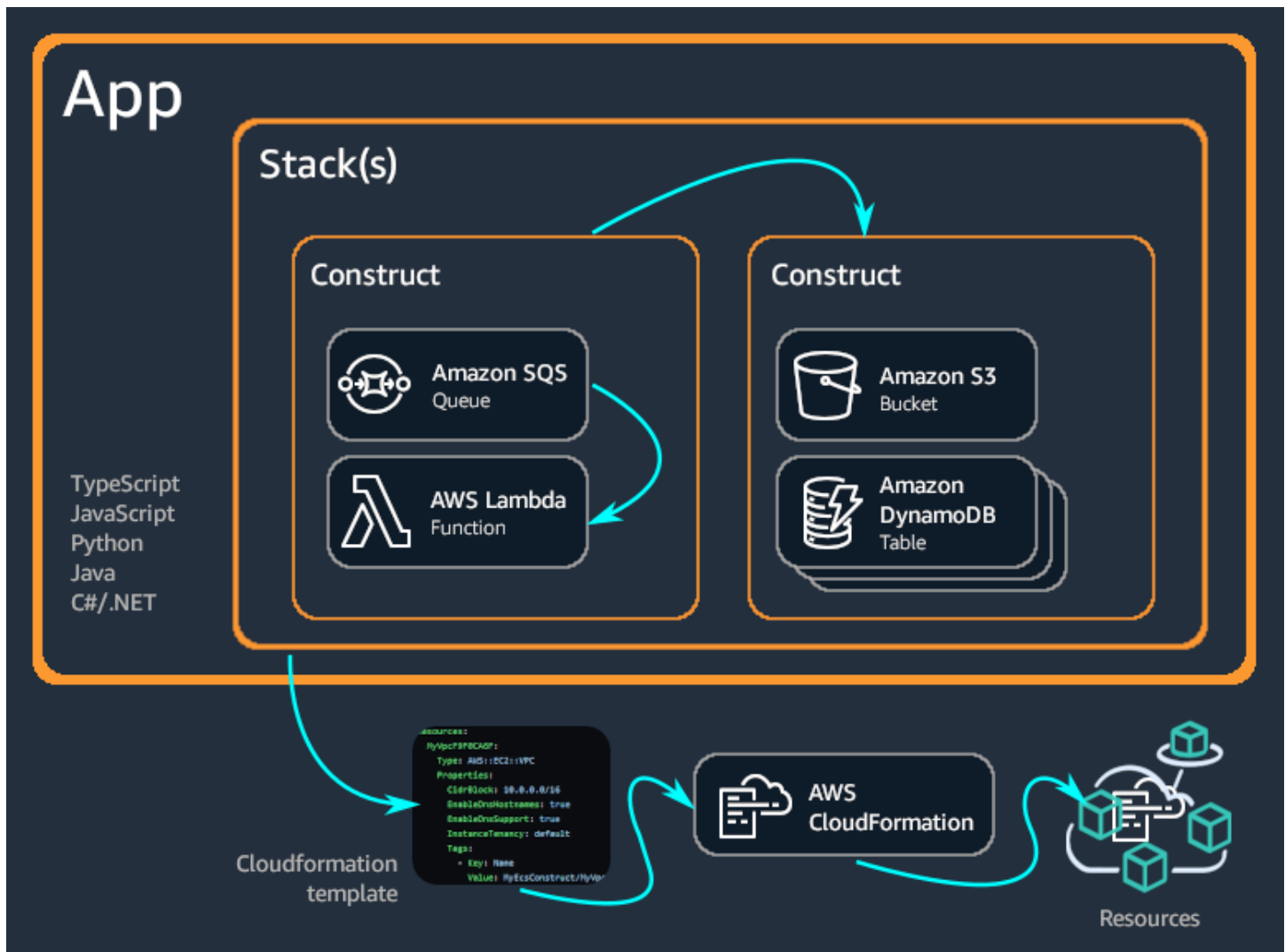
Was ist der AWS CDK?

Das AWS Cloud Development Kit (AWS CDK) ist ein Open-Source-Framework für die Softwareentwicklung, mit dem die Cloud-Infrastruktur im Code definiert und bereitgestellt werden kann. AWS CloudFormation

Das AWS CDK besteht aus zwei Hauptteilen:

- [AWS CDK Construct Library](#) — Eine Sammlung von vorgefertigten modularen und wiederverwendbaren Codeteilen, sogenannten Konstrukten, die Sie verwenden, ändern und integrieren können, um Ihre Infrastruktur schnell zu entwickeln. Das Ziel der AWS CDK Construct Library besteht darin, die Komplexität zu reduzieren, die erforderlich ist, um AWS Dienste gemeinsam zu definieren und zu integrieren, wenn darauf Anwendungen erstellt werden. AWS
- [AWS CDK Toolkit](#) — Ein Befehlszeilentool für die Interaktion mit CDK-Apps. Verwenden Sie das AWS CDK Toolkit, um Ihre Projekte zu erstellen, zu verwalten und bereitzustellen. AWS CDK

Die AWS CDK Stützen TypeScript, JavaScript, Python, Java, C#, .Net, und Go. Sie können jede dieser unterstützten Programmiersprachen verwenden, um wiederverwendbare Cloud-Komponenten, sogenannte [Konstrukte](#), zu definieren. [Sie stellen diese zusammen zu Stacks und Apps zusammen.](#) Anschließend stellen Sie Ihre CDK-Anwendungen bereit, AWS CloudFormation um Ihre Ressourcen bereitzustellen oder zu aktualisieren.



Themen

- [Vorteile des AWS CDK](#)
- [Beispiel für AWS CDK](#)
- [AWS CDK features](#)
- [Nächste Schritte](#)
- [Weitere Informationen](#)

Vorteile des AWS CDK

Verwenden Sie die, AWS CDK um zuverlässige, skalierbare und kostengünstige Anwendungen in der Cloud mit der beachtlichen Ausdruckskraft einer Programmiersprache zu entwickeln. Dieser Ansatz bietet viele Vorteile, darunter:

Entwickeln und verwalten Sie Ihre Infrastruktur als Code (IaC)

Üben Sie Infrastruktur als Code, um Infrastruktur auf programmatische, beschreibende und deklarative Weise zu erstellen, bereitzustellen und zu verwalten. Mit IaC behandeln Sie Infrastruktur genauso wie Entwickler Code behandeln. Dies führt zu einem skalierbaren und strukturierten Ansatz für die Verwaltung der Infrastruktur. Weitere Informationen zu IaC finden Sie unter [Infrastruktur als Code](#) im AWS Whitepaper Einführung DevOps zu.

Mit dem können Sie Ihre Infrastruktur AWS CDK, Ihren Anwendungscode und Ihre Konfiguration an einem zentralen Ort speichern und so sicherstellen, dass Sie bei jedem Meilenstein über ein vollständiges, in der Cloud bereitstellbares System verfügen. Nutzen Sie bewährte Methoden der Softwareentwicklung wie Codeüberprüfungen, Komponententests und Quellcodeverwaltung, um Ihre Infrastruktur robuster zu machen.

Definieren Sie Ihre Cloud-Infrastruktur mithilfe von Allzweck-Programmiersprachen

Mit dem können Sie jede der folgenden Programmiersprachen verwenden AWS CDK, um Ihre Cloud-Infrastruktur zu definieren: TypeScript, JavaScript, Python JavaC#/.Net, und Go. Wählen Sie Ihre bevorzugte Sprache und verwenden Sie Programmiererelemente wie Parameter, Bedingungen, Schleifen, Zusammensetzung und Vererbung, um das gewünschte Ergebnis Ihrer Infrastruktur zu definieren.

Verwenden Sie dieselbe Programmiersprache, um Ihre Infrastruktur und Ihre Anwendungslogik zu definieren.

Nutzen Sie die Vorteile der Infrastrukturentwicklung in Ihrer bevorzugten IDE (Integrated Development Environment), z. B. Syntaxhervorhebung und intelligente Codevervollständigung.

```

TS my_ecs_construct-stack.ts 1, M
lib > TS my_ecs_construct-stack.ts > MyEcsConstructStack > constructor > taskImageOptions > image
1 import { Stack, StackProps } from 'aws-cdk-lib';
2 import { Construct } from 'constructs';
3 // import * as sqs from 'aws-cdk-lib/aws-sqs';
4 import * as ec2 from "aws-cdk-lib/aws-ec2";
5 import * as ecs from "aws-cdk-lib/aws-ecs";
6 import * as ecs_patterns from "aws-cdk-lib/aws-ecs-patterns";
7
8 export class MyEcsConstructStack extends Stack {
9   constructor(scope: Construct, id: string, props?: StackProps) {
10    super(scope, id, props);
11
12    const vpc = new ec2.Vpc(this, "MyVpc", {
13      maxAzs: 3 // Default is all AZs in region
14    });
15
16    const cluster = new ecs.Cluster(this, "MyCluster", {
17      vpc: vpc
18    });
19
20    // Create a load-balanced Fargate service and make it public
21    new ecs_patterns.ApplicationLoadBalancedFargateService(this, "MyFargateService", {
22      cluster: cluster, // Required
23      cpu: 512, // Default is 256
24      desiredCount: 6, // Default is 1
25      taskImageOptions: { image: ecs.ContainerImage.from },
26      memoryLimitMiB: 2048, // Default is 512
27      publicLoadBalancer: true // Default is false
28    });
29  }
30 }
31 }
32

```

Stellen Sie die Infrastruktur bereit durch AWS CloudFormation

AWS CDK integriert sich in AWS CloudFormation, um Ihre Infrastruktur bereitzustellen und bereitzustellen AWS. AWS CloudFormation ist ein verwaltetes System AWS-Service, das umfassende Unterstützung von Ressourcen- und Eigenschaftskonfigurationen für die Bereitstellung von Diensten bietet. AWS Mit AWS CloudFormation können Sie Infrastrukturbereitstellungen vorhersehbar und wiederholt durchführen und bei einem Fehler einen Rollback durchführen. Wenn Sie bereits mit dem vertraut sind AWS CloudFormation, müssen Sie sich nicht mit einem neuen IaC-Verwaltungsservice vertraut machen, wenn Sie mit dem beginnen. AWS CDK

Beginnen Sie schnell mit der Entwicklung Ihrer Anwendung mithilfe von Konstrukten

Entwickeln Sie schneller, indem Sie wiederverwendbare Komponenten, sogenannte Konstrukte, verwenden und gemeinsam nutzen. Verwenden Sie Konstrukte auf niedriger Ebene, um einzelne AWS CloudFormation Ressourcen und ihre Eigenschaften zu definieren. Verwenden Sie

Konstrukte auf hoher Ebene, um schnell größere Komponenten Ihrer Anwendung mit sinnvollen, sicheren Standardeinstellungen für Ihre AWS Ressourcen zu definieren und so mehr Infrastruktur mit weniger Code zu definieren.

Erstellen Sie Ihre eigenen Konstrukte, die auf Ihre individuellen Anwendungsfälle zugeschnitten sind, und teilen Sie sie in Ihrem Unternehmen oder sogar mit der Öffentlichkeit.

Beispiel für AWS CDK

Im Folgenden finden Sie ein Beispiel für die Verwendung der AWS CDK Constructs Library zur Erstellung eines Amazon Elastic Container Service (Amazon ECS) -Service mit AWS Fargate (Fargate) Starttyp. Weitere Informationen zu diesem Beispiel finden Sie unter [the section called “ECS”](#).

TypeScript

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

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

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

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

JavaScript

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

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

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

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

module.exports = { MyEcsConstructStack }
```

Python

```
class MyEcsConstructStack(Stack):

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

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

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

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

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

Java

```
public class MyEcsConstructStack extends Stack {

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

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

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

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

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

C#

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

```

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

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

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

```

Go

```

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

    var sprops awscdk.StackProps

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

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

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

```



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

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

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

  return stack
}
```

Diese Klasse erzeugt eine AWS CloudFormation [Vorlage mit mehr als 500 Zeilen](#). Durch die Bereitstellung der AWS CDK App werden mehr als 50 Ressourcen der folgenden Typen erzeugt.

- [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

Das AWS CDK GitHub Repository

Das offizielle AWS CDK GitHub Repository finden Sie unter [aws-cdk](#). Hier können Sie [Probleme](#) einreichen, unsere [Lizenz](#) einsehen, [Veröffentlichungen verfolgen und vieles mehr](#).

Da es sich um AWS CDK ein Open-Source-Tool handelt, ermutigt das Team Sie, dazu beizutragen, es zu einem noch besseren Tool zu machen. Einzelheiten finden Sie unter [Beitrag zum AWS Cloud Development Kit \(AWS CDK\)](#).

Die AWS CDK API-Referenz

Die AWS CDK Construct Library stellt APIs bereit, mit denen Sie Ihre CDK-Anwendung definieren und der Anwendung CDK-Konstrukte hinzufügen können. Weitere Informationen finden Sie in der [AWS CDK -API-Referenz](#).

Das Construct-Programmiermodell

Das Construct Programming Model (CPM) erweitert die dahinter stehenden Konzepte auf weitere AWS CDK Bereiche. Zu den anderen Tools, die das CPM verwenden, gehören:

- [CDK für Terraform](#) (cdkTF)
- CDK für Kubernetes ([CDK8s](#))
- [Projen, zum Erstellen von Projektkonfigurationen](#)

Der Construct Hub

Der [Construct Hub](#) ist eine Online-Registrierung, in der Sie AWS CDK Open-Source-Bibliotheken finden, veröffentlichen und teilen können.

Nächste Schritte

Erste Schritte mit der Verwendung von finden Sie AWS CDK unter [Erste Schritte mit der AWS CDK](#).

Weitere Informationen

Weitere Informationen zum AWS CDK finden Sie im Folgenden:

- [Lernen Sie die AWS CDK Kernkonzepte kennen](#) — Wichtige Konzepte und Begriffe für die AWS CDK.
- [AWS CDK Workshop](#) — Praktischer Workshop zum Erlernen und Anwenden AWS CDK von.
- [AWS CDK Patterns](#) — Open-Source-Sammlung von AWS serverlosen Architekturmustern, die von Experten speziell für Sie entwickelt wurden. AWS CDK AWS
- [AWS CDK Codebeispiele](#) — GitHub Sammlung von AWS CDK Beispielprojekten.
- [cdk.dev](#) — [Von](#) der Community betriebener Hub für die AWS CDK, einschließlich eines Community-Workspace. Slack
- [Fantastisches CDK](#) — GitHub Repository mit einer kuratierten Liste von AWS CDK Open-Source-Projekten, Leitfäden, Blogs und anderen Ressourcen.
- [AWS Lösungskonstrukte — Geprüfte](#) Muster für die Konfiguration der Infrastruktur als Code (IaC), die einfach zu produktionsreifen Anwendungen zusammengefügt werden können.
- AWS Blog zu [Entwicklertools — Blogbeiträge](#), gefiltert nach dem. AWS CDK
- [AWS CDK on Stack Overflow](#) — Fragen, die mit aws-cdk on markiert sind. Stack Overflow
- [AWS CDK Tutorial für AWS Cloud9](#) — Tutorial zur Verwendung von AWS CDK mit der AWS Cloud9 Entwicklungsumgebung.

Weitere Informationen zu verwandten Themen zum AWS CDK finden Sie im Folgenden:

- [AWS CloudFormation Konzepte](#) — Da der so konzipiert AWS CDK ist, dass Sie damit arbeiten können AWS CloudFormation, empfehlen wir Ihnen, sich mit den wichtigsten AWS CloudFormation Konzepten vertraut zu machen und sie zu verstehen.

- [AWS Glossar](#) — Definitionen der wichtigsten Begriffe, die überall AWS verwendet werden.

Weitere Informationen zu Tools im Zusammenhang mit der AWS CDK , mit denen die Entwicklung und Bereitstellung serverloser Anwendungen vereinfacht werden kann, finden Sie im Folgenden:

- [AWS Serverless Application Model](#)— Ein Open-Source-Entwicklertool, das das Erstellen und Ausführen von serverlosen Anwendungen vereinfacht und verbessert. AWS
- [AWSChalice](#)— Ein Framework zum Schreiben serverloser Apps. Python

Lernen Sie die AWS CDK Kernkonzepte kennen

Lernen Sie die Kernkonzepte kennen, die hinter dem stehen AWS Cloud Development Kit (AWS CDK).

AWS CDK und IaC

Das AWS CDK ist ein Open-Source-Framework, mit dem Sie Ihre AWS Infrastruktur mithilfe von Code verwalten können. Dieser Ansatz wird Infrastructure as Code (IaC) genannt. Indem Sie Ihre Infrastruktur als Code verwalten und bereitstellen, behandeln Sie Ihre Infrastruktur genauso, wie Entwickler Code behandeln. Dies bietet viele Vorteile, wie z. B. Versionskontrolle und Skalierbarkeit. Weitere Informationen zu IaC finden Sie unter [Was ist Infrastruktur als Code?](#)

AWS CDK und AWS CloudFormation

Das AWS CDK ist eng integriert mit AWS CloudFormation. AWS CloudFormation ist ein vollständig verwalteter Service, mit dem Sie Ihre Infrastruktur verwalten und bereitstellen können AWS. Mit AWS CloudFormation definieren Sie Ihre Infrastruktur in Vorlagen und stellen diese bereit AWS CloudFormation. Der AWS CloudFormation Service stellt dann Ihre Infrastruktur gemäß der in Ihren Vorlagen definierten Konfiguration bereit.

AWS CloudFormation Vorlagen sind deklarativ, d. h. sie deklarieren den gewünschten Zustand oder das gewünschte Ergebnis Ihrer Infrastruktur. Mithilfe von JSON oder YAML deklarieren Sie Ihre AWS Infrastruktur, indem Sie AWS Ressourcen und Eigenschaften definieren. Ressourcen stellen die vielen verfügbaren Dienste dar AWS und Eigenschaften stellen Ihre gewünschte Konfiguration dieser Dienste dar. Wenn Sie Ihre Vorlage auf bereitstellen AWS CloudFormation, werden Ihre Ressourcen und ihre konfigurierten Eigenschaften wie in Ihrer Vorlage beschrieben bereitgestellt.

Mit dem AWS CDK können Sie Ihre Infrastruktur mithilfe von Allzweck-Programmiersprachen zwingend verwalten. Anstatt nur einen gewünschten Zustand deklarativ zu definieren, können Sie die Logik oder Reihenfolge definieren, die erforderlich ist, um den gewünschten Zustand zu erreichen. Sie können beispielsweise `if` Anweisungen oder bedingte Schleifen verwenden, die bestimmen, wie ein gewünschter Endzustand für Ihre Infrastruktur erreicht werden kann.

Die mit dem erstellte Infrastruktur AWS CDK wird schließlich übersetzt oder in AWS CloudFormation Vorlagen zusammengefasst und mithilfe des AWS CloudFormation Dienstes bereitgestellt. Der

AWS CDK bietet zwar einen anderen Ansatz zur Erstellung Ihrer Infrastruktur, Sie profitieren aber dennoch von den Vorteilen AWS CloudFormation, wie z. B. umfassender Unterstützung bei der AWS Ressourcenkonfiguration und robusten Bereitstellungsprozessen.

Weitere Informationen AWS CloudFormation dazu finden Sie unter [Was ist AWS CloudFormation?](#) im AWS CloudFormation Benutzerhandbuch.

AWS CDK und Abstraktionen

Mit AWS CloudFormation müssen Sie jedes Detail der Konfiguration Ihrer Ressourcen definieren. Dies bietet den Vorteil, dass Sie die vollständige Kontrolle über Ihre Infrastruktur haben. Dies erfordert jedoch, dass Sie sich mit robusten Vorlagen vertraut machen, diese verstehen und erstellen, die Details zur Ressourcenkonfiguration und Beziehungen zwischen Ressourcen enthalten, z. B. Berechtigungen und ereignisgesteuerte Interaktionen.

Mit dem AWS CDK können Sie die gleiche Kontrolle über Ihre Ressourcenkonfigurationen haben. Das bietet jedoch AWS CDK auch leistungsstarke Abstraktionen, die den Infrastrukturentwicklungsprozess beschleunigen und vereinfachen können. AWS CDK Dazu gehören beispielsweise Konstrukte, die sinnvolle Standardkonfigurationen bereitstellen, und Hilfsmethoden, die Standardcode für Sie generieren. Das bietet AWS CDK auch Tools wie die AWS CDK Befehlszeilenschnittstelle (AWS CDK CLI), die Infrastrukturverwaltungsaktionen für Sie ausführen.

Erfahren Sie mehr über die AWS CDK Kernkonzepte

Interaktion mit dem AWS CDK

Bei der AWS CDK Verwendung von interagieren Sie hauptsächlich mit der AWS Construct-Bibliothek und dem AWS CDK CLI.

Entwickeln mit dem AWS CDK

Das AWS CDK kann in jeder [unterstützten Programmiersprache](#) geschrieben werden. Sie beginnen mit einem [CDK-Projekt](#), das eine Struktur von Ordnern und Dateien, einschließlich [Assets](#), enthält. Innerhalb des Projekts erstellen Sie eine [CDK-Anwendung](#). Innerhalb der App definieren Sie einen [Stack](#), der direkt einen CloudFormation Stack darstellt. Innerhalb des Stacks definieren Sie Ihre AWS Ressourcen und Eigenschaften mithilfe von [Konstrukten](#).

Bereitstellung mit dem AWS CDK

Sie stellen CDK-Apps in einer AWS [Umgebung](#) bereit. Vor der Bereitstellung müssen Sie ein einmaliges [Bootstrapping](#) durchführen, um Ihre Umgebung vorzubereiten.

Weitere Informationen

Weitere Informationen zu den AWS CDK Kernkonzepten finden Sie in den Themen in diesem Abschnitt.

Unterstützte Programmiersprachen

Der AWS Cloud Development Kit (AWS CDK) bietet erstklassige Unterstützung für die folgenden Allzweck-Programmiersprachen:

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

Theoretisch können auch andere JVM .NET CLR Sprachen verwendet werden, aber wir bieten derzeit keinen offiziellen Support an.

Note

Dieses Handbuch enthält derzeit keine Anleitungen oder Codebeispiele für Go abgesehen von [the section called “In Go”](#).

AWS CDK Das wurde in einer Sprache entwickelt, TypeScript. Um die anderen Sprachen zu unterstützen, AWS CDK verwendet der ein Tool, das aufgerufen wird [JSII](#), Sprachbindungen zu generieren.

Wir versuchen, die üblichen Konventionen jeder Sprache anzubieten, um die Entwicklung AWS CDK so natürlich und intuitiv wie möglich zu gestalten. Zum Beispiel verteilen wir AWS Construct Library-

Module über das Standard-Repository Ihrer bevorzugten Sprache, und Sie installieren sie mit dem Standard-Paketmanager der Sprache. Methoden und Eigenschaften werden ebenfalls nach den von Ihrer Sprache empfohlenen Benennungsmustern benannt.

Im Folgenden finden Sie einige Codebeispiele:

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

Diese Codefragmente dienen nur der Veranschaulichung. Sie sind unvollständig und werden nicht so ausgeführt, wie sie sind.

Die AWS Construct-Bibliothek wird mit den Standard-Paketverwaltungstools jeder Sprache verteilt: NPM, einschließlich PyPi, Maven, und NuGet. Wir bieten auch eine Version der [AWS CDK API-Referenz](#) für jede Sprache an.

Um Ihnen bei der Verwendung von AWS CDK in Ihrer bevorzugten Sprache zu helfen, enthält dieser Leitfaden die folgenden Themen für unterstützte Sprachen:

- [the section called “In TypeScript”](#)
- [the section called “In JavaScript”](#)
- [the section called “In Python”](#)
- [the section called “In Java”](#)
- [the section called “In C#”](#)
- [the section called “In Go”](#)

TypeScript war die erste Sprache AWS CDK, die von unterstützt wurde, und ein Großteil des AWS CDK Beispielcodes ist in dieser Sprache geschrieben. Dieses Handbuch enthält ein Thema, das speziell zeigt, wie TypeScript AWS CDK Code für die Verwendung mit den anderen

unterstützten Sprachen angepasst werden kann. Weitere Informationen finden Sie unter [Vergleich von AWS CDK in TypeScript mit anderen Sprachen](#).

AWS CDK Projekte

Ein - AWS Cloud Development Kit (AWS CDK) Projekt stellt die Dateien und Ordner dar, die Ihren CDK-Code enthalten. Die Inhalte variieren je nach Programmiersprache.

Sie können Ihr AWS CDK Projekt manuell oder mit dem AWS CDK Befehl Command Line Interface (AWS CDK CLI) erstellen `cdk init`. In diesem Thema beziehen wir uns auf die Projektstruktur und Benennungskonventionen von Dateien und Ordnern, die von der AWS-CDK-CLI erstellt wurden. Sie können Ihre CDK-Projekte an Ihre Bedürfnisse anpassen und organisieren.

Note

Die vom erstellte Projektstruktur kann im Laufe der AWS CDK CLI Zeit je nach Version variieren.

Themen

- [Universale Dateien und Ordner](#)
- [Sprachspezifische Dateien und Ordner](#)

Universale Dateien und Ordner

`.git`

Wenn Sie `git` installiert haben, initialisiert die AWS CDK CLI automatisch ein `GitRepository` für Ihr Projekt. Das `.git` Verzeichnis enthält Informationen über das Repository.

`.gitignore`

Textdatei, die von verwendet wird `Git`, um Dateien und Ordner anzugeben, die ignoriert werden sollen.

`README.md`

Textdatei, die Ihnen grundlegende Anleitungen und wichtige Informationen zur Verwaltung Ihres AWS CDK Projekts bietet. Ändern Sie diese Datei nach Bedarf, um wichtige Informationen zu Ihrem CDK-Projekt zu dokumentieren.

cdk.json

Konfigurationsdatei für die AWS CDK. Diese Datei enthält Anweisungen an die AWS CDK CLI zur Ausführung Ihrer App.

Sprachspezifische Dateien und Ordner

Die folgenden Dateien und Ordner sind für jede unterstützte Programmiersprache eindeutig.

TypeScript

Im Folgenden finden Sie ein Beispielprojekt, das mit dem `cdk init --language typescript` Befehl im `-my-cdk-ts-project` Verzeichnis erstellt wurde:

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

.npmignore

Datei, die angibt, welche Dateien und Ordner beim Veröffentlichen eines Pakets in der npm Registrierung ignoriert werden sollen. Diese Datei ist ähnlich wie `.gitignore`, aber spezifisch für `-npm` Pakete.

bin/my-cdk-ts-project.ts

Die Anwendungsdatei definiert Ihre CDK-App. CDK-Projekte können eine oder mehrere Anwendungsdateien enthalten. Anwendungsdateien werden im `bin` Ordner gespeichert.

Im Folgenden finden Sie ein Beispiel für eine einfache Anwendungsdatei, die eine CDK-App definiert:

```
#!/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

Konfigurationsdatei für Jest. Jest ist ein beliebtes JavaScript Test-Framework.

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

Die Stack-Datei definiert Ihren CDK-Stack. Innerhalb Ihres Stacks definieren AWS Sie Ressourcen und Eigenschaften mithilfe von Konstrukten.

Im Folgenden finden Sie ein Beispiel für eine einfache Stack-Datei, die einen CDK-Stack definiert:

```
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

Gemeinsamer Ordner in Node.js Projekten, die Abhängigkeiten für Ihr Projekt enthalten.

package-lock.json

Metadatendatei, die mit der `-package.json` Datei zur Verwaltung von Versionen von Abhängigkeiten funktioniert.

package.json

Metadatei, die häufig in Node.js Projekten verwendet wird. Diese Datei enthält Informationen über Ihr CDK-Projekt, z. B. den Projektnamen, Skriptdefinitionen, Abhängigkeiten und andere Informationen auf Projektebene importieren.

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

Ein Testordner wird erstellt, um Tests für Ihr CDK-Projekt zu organisieren. Es wird auch eine Beispieltestdatei erstellt.

Sie können Tests in schreiben TypeScript und verwenden Jest, um Ihren TypeScript Code zu kompilieren, bevor Sie Tests ausführen.

tsconfig.json

Konfigurationsdatei, die in TypeScript Projekten verwendet wird, die Compiler-Optionen und Projekteinstellungen angeben.

JavaScript

Im Folgenden finden Sie ein Beispielprojekt, das mit dem `cdk init --language javascript` Befehl im `my-cdk-js-project` Verzeichnis erstellt wurde:

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

.npmignore

Datei, die angibt, welche Dateien und Ordner beim Veröffentlichen eines Pakets in der npm Registrierung ignoriert werden sollen. Diese Datei ist ähnlich wie `.gitignore`, aber spezifisch für -npmPakete.

bin/my-cdk-js-project.js

Die Anwendungsdatei definiert Ihre CDK-App. CDK-Projekte können eine oder mehrere Anwendungsdateien enthalten. Anwendungsdateien werden im `bin` Ordner gespeichert.

Im Folgenden finden Sie ein Beispiel für eine einfache Anwendungsdatei, die eine CDK-App definiert:

```
#!/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

Konfigurationsdatei für Jest. Jest ist ein beliebtes JavaScript Test-Framework.

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

Die Stack-Datei definiert Ihren CDK-Stack. Innerhalb Ihres Stacks definieren AWS Sie Ressourcen und Eigenschaften mithilfe von Konstrukten.

Im Folgenden finden Sie ein Beispiel für eine einfache Stack-Datei, die einen CDK-Stack definiert:

```
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

Gemeinsamer Ordner in Node.js Projekten, die Abhängigkeiten für Ihr Projekt enthalten.

package-lock.json

Metadatendatei, die mit der `package.json` Datei zur Verwaltung von Versionen von Abhängigkeiten funktioniert.

package.json

Metadatendatei, die häufig in Node.js Projekten verwendet wird. Diese Datei enthält Informationen über Ihr CDK-Projekt, z. B. den Projektnamen, Skriptdefinitionen, Abhängigkeiten und andere Informationen auf Projektebene importieren.

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

Ein Testordner wird erstellt, um Tests für Ihr CDK-Projekt zu organisieren. Es wird auch eine Beispieltestdatei erstellt.

Sie können Tests in schreiben JavaScript und verwenden Jest, um Ihren JavaScript Code zu kompilieren, bevor Sie Tests ausführen.

Python

Im Folgenden finden Sie ein Beispielprojekt, das mit dem `cdk init --language python` Befehl im `my-cdk-py-project` Verzeichnis erstellt wurde:

```
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

Das CDK erstellt CLI automatisch eine virtuelle Umgebung für Ihr Projekt. Das `.venv` Verzeichnis bezieht sich auf diese virtuelle Umgebung.

app.py

Die Anwendungsdatei definiert Ihre CDK-App. CDK-Projekte können eine oder mehrere Anwendungsdateien enthalten.

Im Folgenden finden Sie ein Beispiel für eine einfache Anwendungsdatei, die eine CDK-App definiert:

```
#!/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_projekt

Verzeichnis, das Ihre Stack-Dateien enthält. Das CDK CLI erstellt hier Folgendes:

- `__init__.py` – Eine leere Python Paketdefinitionsdatei.
- `my_cdk_py_project` – Datei, die Ihren CDK-Stack definiert. Anschließend definieren Sie AWS Ressourcen und Eigenschaften innerhalb des Stacks mithilfe von Konstrukten.

Im Folgenden finden Sie ein Beispiel für eine Stack-Datei:

```
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

Datei ähnlich wie `requirements.txt`, wird aber verwendet, um Abhängigkeiten speziell für Entwicklungszwecke und nicht für die Produktion zu verwalten.

requirements.txt

Geläufige Datei, die in Python Projekten verwendet wird, um Projektabhängigkeiten anzugeben und zu verwalten.

source.bat

Batch-Datei für Windows, die zum Einrichten der Python virtuellen Umgebung verwendet wird.

Tests

Verzeichnis, das Tests für Ihr CDK-Projekt enthält.

Im Folgenden finden Sie ein Beispiel für einen Komponententest:

```
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

Im Folgenden finden Sie ein Beispielprojekt, das mit dem `-cdk init --language java` Befehl im `-my-cdk-java-project` Verzeichnis erstellt wurde:

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

pom.xml

Datei, die Konfigurationsinformationen und Metadaten zu Ihrem CDK-Projekt enthält. Diese Datei ist Teil von Maven.

src/main

Verzeichnis, das Ihre Anwendungs- und Stack-Dateien enthält.

Im Folgenden finden Sie ein Beispiel für eine Anwendungsdatei:

```
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();
    }
}
```

Im Folgenden finden Sie ein Beispiel für eine Stack-Datei:

```
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

Verzeichnis, das Ihre Testdateien enthält. Im Folgenden wird ein Beispiel gezeigt:

```
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#

Im Folgenden finden Sie ein Beispielprojekt, das mit dem `cdk init --language csharp` Befehl im `-my-cdk-csharp-project` Verzeichnis erstellt wurde:

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

src/MyCdkCsharpProject

Verzeichnis, das Ihre Anwendungs- und Stack-Dateien enthält.

Im Folgenden finden Sie ein Beispiel für eine Anwendungsdatei:

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

Im Folgenden finden Sie ein Beispiel für eine Stack-Datei:

```
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
        }
    }
}
```

Dieses Verzeichnis enthält auch Folgendes:

- `GlobalSuppressions.cs` – Datei, die verwendet wird, um bestimmte Compiler-Warnungen oder -Fehler in Ihrem gesamten Projekt zu unterdrücken.
- `.csproj` – XML-basierte Datei, die zur Definition von Projekteinstellungen, Abhängigkeiten und Build-Konfigurationen verwendet wird.

`src/MyCdkCsharpProject.sln`

Microsoft Visual Studio Solution File wird verwendet, um verwandte Projekte zu organisieren und zu verwalten.

Go

Im Folgenden finden Sie ein Beispielprojekt, das mit dem `-cdk init --language go` Befehl im `-my-cdk-go-project` Verzeichnis erstellt wurde:

```
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

Datei, die Modulinformationen enthält und zur Verwaltung von Abhängigkeiten und Versioning für Ihr Go Projekt verwendet wird.

my-cdk-go-project.go

Datei, die Ihre CDK-Anwendung und -Stacks definiert.

Im Folgenden wird ein Beispiel gezeigt:

```
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

Datei, die einen Beispieltest definiert.

Im Folgenden wird ein Beispiel gezeigt:

```
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 Apps

Die AWS Cloud Development Kit (AWS CDK) Anwendung oder App ist eine Sammlung von einem oder mehreren CDK-[Stacks](#). Stacks sind eine Sammlung von einem oder mehreren [Konstrukten](#), die

AWS Ressourcen und Eigenschaften definieren. Daher wird die Gesamtgruppierung Ihrer Stacks und Konstrukte als CDK-App bezeichnet.

Themen

- [Definieren von Apps](#)
- [Der Konstruktbaum](#)
- [Der Anwendungslebenszyklus](#)

Definieren von Apps

Sie erstellen eine App, indem Sie eine App-Instance in der Anwendungsdatei Ihres [Projekts](#) definieren. Dazu importieren und verwenden Sie das [App](#)-Konstrukt aus der AWS Construct Library. Für das AppKonstrukt sind keine Initialisierungsargumente erforderlich. Es ist das einzige Konstrukt, das als Stamm verwendet werden kann.

Die [Stack](#)-Klassen [App](#) und aus der AWS Construct Library sind eindeutige Konstrukte. Im Vergleich zu anderen Konstrukten konfigurieren sie keine AWS Ressourcen selbst. Stattdessen werden sie verwendet, um Kontext für Ihre anderen Konstrukte bereitzustellen. Alle Konstrukte, die AWS Ressourcen darstellen, müssen direkt oder indirekt im Rahmen eines Stack Konstrukts definiert werden. Stack Konstrukte werden im Rahmen eines AppKonstrukts definiert.

Apps werden dann synthetisiert, um AWS CloudFormation Vorlagen für Ihre Stacks zu erstellen. Im Folgenden wird ein Beispiel gezeigt:

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)
```

Stacks innerhalb einer einzigen App können sich problemlos auf die Ressourcen und Eigenschaften des jeweils anderen beziehen. Die AWS CDK leitet Abhängigkeiten zwischen Stacks ab, sodass sie in der richtigen Reihenfolge bereitgestellt werden können. Sie können einen oder alle Stacks innerhalb einer App mit einem einzigen `cdk deploy` Befehl bereitstellen.

Der Konstruktbaum

Konstrukte werden innerhalb anderer `scope` Konstrukte mit dem Argument definiert, das an jedes Konstrukt übergeben wird, wobei die App Klasse das Stammverzeichnis ist. Auf diese Weise definiert eine AWS CDK App eine Hierarchie von Konstrukten, die als Konstruktbaum bezeichnet wird.

Das Stammverzeichnis dieses Baums ist Ihre App, eine Instance der -AppKlasse. Innerhalb der App instanzieren Sie einen oder mehrere Stacks. Innerhalb von Stacks instanzieren Sie Konstrukte, die selbst Ressourcen oder andere Konstrukte instanzieren können usw.

Konstrukte werden immer explizit im Rahmen eines anderen Konstrukts definiert, wodurch Beziehungen zwischen Konstrukten entstehen. Fast immer sollten Sie `this` (in Python, `self`) als Bereich übergeben, was darauf hinweist, dass das neue Konstrukt ein untergeordnetes Element des aktuellen Konstrukts ist. Das beabsichtigte Muster besteht darin, dass Sie Ihr Konstrukt von ableiten [Construct](#) und dann die Konstrukte instanzieren, die es in seinem Konstruktor verwendet.

Durch die explizite Übergabe des Bereichs kann jedes Konstrukt sich selbst zum Baum hinzufügen, wobei dieses Verhalten vollständig in der [Construct Basisklasse](#) enthalten ist. Es funktioniert in jeder Sprache, die von unterstützt wird, auf die gleiche Weise AWS CDK und erfordert keine zusätzliche Anpassung.

Important

Technisch gesehen ist es möglich, einen anderen Bereich als `this` beim Instanzieren eines Konstrukts zu übergeben. Sie können Konstrukte überall im Baum oder sogar in einem anderen Stack in derselben App hinzufügen. Sie könnten beispielsweise eine Funktion im Mixin-Stil schreiben, die Konstrukte zu einem Bereich hinzufügt, der als Argument übergeben wird. Die praktische Schwierigkeiten besteht darin, dass Sie nicht einfach sicherstellen können, dass die IDs, die Sie für Ihre Konstrukte auswählen, innerhalb des Geltungsbereichs einer anderen Person eindeutig sind. Die Praxis macht es Ihrem Code auch schwieriger, ihn zu verstehen, zu warten und wiederzuverwenden. Es ist fast immer besser, eine Möglichkeit zu finden, Ihre Absicht auszudrücken, ohne das `scope` Argument zu missbrauchen.

Der AWS CDK verwendet die IDs aller Konstrukte im Pfad vom Stamm des Baums zu jedem untergeordneten Konstrukt, um die eindeutigen IDs zu generieren, die von benötigt werden AWS CloudFormation. Dieser Ansatz bedeutet, dass Konstrukt-IDs nur innerhalb ihres Bereichs und nicht innerhalb des gesamten Stacks wie in nativen eindeutig sein müssen AWS CloudFormation. Wenn Sie ein Konstrukt jedoch in einen anderen Bereich verschieben, ändert sich seine generierte Stack-AWS CloudFormation eindeutige ID und betrachtet sie nicht als dieselbe Ressource.

Der Konstruktbaum ist getrennt von den Konstrukten, die Sie in Ihrem AWS CDK Code definieren. Es ist jedoch über das `node` Konstruktattribut zugänglich, das ein Verweis auf den Knoten ist, der dieses Konstrukt im Baum darstellt. Jeder Knoten ist eine [Node](#) Instance, deren Attribute Zugriff auf

den Stamm des Baums sowie auf die übergeordneten Bereiche und untergeordneten Elemente des Knotens gewähren.

1. `node.children` – Die direkten untergeordneten Elemente des Konstrukts.
2. `node.id` – Die Kennung des Konstrukts innerhalb seines Bereichs.
3. `node.path` – Der vollständige Pfad des Konstrukts, einschließlich der IDs aller übergeordneten Elemente.
4. `node.root` – Der Stamm des Konstruktbaums (die App).
5. `node.scope` – Der Bereich (übergeordnet) des Konstrukts oder nicht definiert, wenn der Knoten der Stamm ist.
6. `node.scopes` – Alle übergeordneten Elemente des Konstrukts bis zum Stamm.
7. `node.uniqueId` – Die eindeutige alphanumerische Kennung für dieses Konstrukt innerhalb des Baums (standardmäßig generiert von `node.path` und einem Hash).

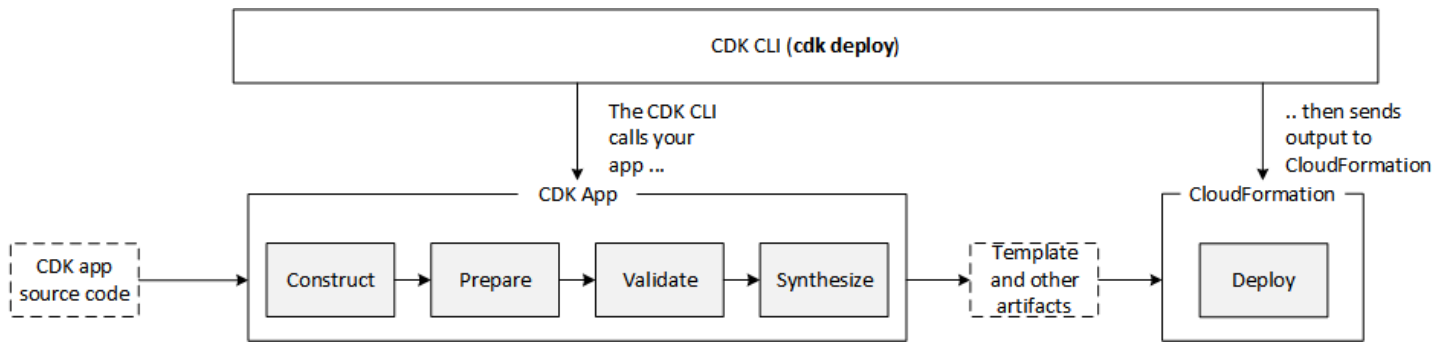
Der Konstruktbaum definiert eine implizite Reihenfolge, in der Konstrukte mit Ressourcen in der endgültigen AWS CloudFormation Vorlage synthetisiert werden. Wo eine Ressource vor einer anderen erstellt werden muss AWS CloudFormation oder die AWS Construct Library die Abhängigkeit im Allgemeinen ableitet. Anschließend stellen sie sicher, dass die Ressourcen in der richtigen Reihenfolge erstellt werden.

Sie können auch eine explizite Abhängigkeit zwischen zwei Knoten hinzufügen, indem Sie verwenden `node.addDependency()`. Weitere Informationen finden Sie unter [Abhängigkeiten](#) in der API AWS CDK -Referenz zu .

Die AWS CDK bietet eine einfache Möglichkeit, jeden Knoten in der Konstruktstruktur zu besuchen und einen Vorgang für jeden Knoten durchzuführen. Weitere Informationen finden Sie unter [the section called "Aspekte"](#).

Der Anwendungslebenszyklus

Wenn Sie Ihre CDK-App bereitstellen, finden die folgenden Phasen statt. Dies wird als Anwendungslebenszyklus bezeichnet:



Eine AWS CDK App durchläuft die folgenden Phasen ihres Lebenszyklus.

- **Konstruktion (oder Initialisierung)** – Ihr Code instanziiert alle definierten Konstrukte und verknüpft sie dann miteinander. In dieser Phase werden alle Konstrukte (App, Stacks und ihre untergeordneten Konstrukte) instanziiert und die Konstruktorkette wird ausgeführt. Der Großteil Ihres App-Codes wird in dieser Phase ausgeführt.
- **Vorbereitung** – Alle Konstrukte, die die `prepare` Methode implementiert haben, nehmen an einer abschließenden Runde von Änderungen teil, um ihren endgültigen Status einzurichten. Die Vorbereitungsphase erfolgt automatisch. Als Benutzer sehen Sie kein Feedback aus dieser Phase. Es ist selten, dass Sie den Hook „Vorbereiten“ verwenden müssen, was im Allgemeinen nicht empfohlen wird. Seien Sie während dieser Phase sehr vorsichtig, wenn Sie den Konstruktbaum mutieren, da sich die Reihenfolge der Operationen auf das Verhalten auswirken könnte.
- **Validierung** – Alle Konstrukte, die die `validate` Methode implementiert haben, können sich selbst validieren, um sicherzustellen, dass sie sich in einem Zustand befinden, der korrekt bereitgestellt wird. Sie werden über alle Validierungsfehler informiert, die während dieser Phase auftreten. Im Allgemeinen empfehlen wir, die Validierung so schnell wie möglich durchzuführen (in der Regel sobald Sie einige Eingaben erhalten) und Ausnahmen so früh wie möglich auszulösen. Die frühzeitige Durchführung einer Validierung verbessert die Zuverlässigkeit, da Stack-Traces genauer sein werden, und stellt sicher, dass Ihr Code weiterhin sicher ausgeführt werden kann.
- **Synthese** – Dies ist die letzte Phase der Ausführung Ihrer AWS CDK App. Sie wird durch einen Aufruf von `ausgelöstapp.synth()`, durchläuft den Konstruktbaum und ruft die `synthesize` Methode für alle Konstrukte auf. Konstrukte, die implementieren, `synthesize` können an der Synthetisierung beteiligt sein und Bereitstellungsartefakte an die resultierende Cloud-Baugruppe ausgeben. Zu diesen Artefakten gehören AWS CloudFormation Vorlagen, AWS Lambda Anwendungspakete, Datei- und Docker Image-Komponenten sowie andere Bereitstellungsartefakte. [the section called “Cloud-Komponenten”](#) beschreibt die Ausgabe dieser Phase. In den meisten Fällen müssen Sie die `synthesize` Methode nicht implementieren.

- **Bereitstellung** – In dieser Phase nimmt das AWS CDK CLI die von der -Synthetikphase erzeugten Bereitstellungsartefakte der Cloud-Baugruppe und stellt sie in einer AWS Umgebung bereit. Es lädt Komponenten auf Amazon S3 und Amazon ECR hoch, oder wo immer sie hingehören müssen. Anschließend wird eine - AWS CloudFormation Bereitstellung gestartet, um die Anwendung bereitzustellen und die Ressourcen zu erstellen.

Zum Zeitpunkt des Starts der AWS CloudFormation Bereitstellungsphase ist Ihre AWS CDK App bereits abgeschlossen und beendet. Dies hat folgende Auswirkungen:

- Die AWS CDK App kann nicht auf Ereignisse reagieren, die während der Bereitstellung auftreten, z. B. auf eine Ressource, die erstellt wird, oder auf die gesamte Bereitstellung, die abgeschlossen wird. Um Code während der Bereitstellungsphase auszuführen, müssen Sie ihn als [benutzerdefinierte Ressource](#) in die AWS CloudFormation Vorlage einfügen. Weitere Informationen zum Hinzufügen einer benutzerdefinierten Ressource zu Ihrer App finden Sie im [AWS CloudFormation Modul](#) oder im Beispiel [für benutzerdefinierte Ressourcen](#).
- Die AWS CDK App muss möglicherweise mit Werten arbeiten, die zum Zeitpunkt ihrer Ausführung nicht bekannt sind. Wenn die AWS CDK App beispielsweise einen Amazon S3-Bucket mit einem automatisch generierten Namen definiert und Sie das Attribut `bucket.bucketName` (Python: `bucket_name`) abrufen, ist dieser Wert nicht der Name des bereitgestellten Buckets. Stattdessen erhalten Sie einen -TokenWert. Um festzustellen, ob ein bestimmter Wert verfügbar ist, rufen Sie auf `cdk.isUnresolved(value)` (Python: `is_unresolved`). Details dazu finden Sie unter [the section called “Token”](#).

Cloud-Komponenten

Der Aufruf an `app.synth()` weist den an AWS CDK, eine Cloud-Baugruppe aus einer App zu synthetisieren. In der Regel interagieren Sie nicht direkt mit Cloud-Komponenten. Es handelt sich um Dateien, die alles enthalten, was für die Bereitstellung Ihrer App in einer Cloud-Umgebung erforderlich ist. Sie enthält beispielsweise eine - AWS CloudFormation Vorlage für jeden Stack in Ihrer App. Sie enthält auch eine Kopie aller Dateikomponenten oder Docker-Images, auf die Sie in Ihrer App verweisen.

Einzelheiten zur Formatierung von [Cloud-Komponenten finden Sie in der Spezifikation](#) für Cloud-Komponenten.

Um mit der Cloud-Baugruppe zu interagieren, die Ihre AWS CDK App erstellt, verwenden Sie normalerweise die AWS CDK CLI. Jedes Tool, das das Cloud-Assembly-Format lesen kann, kann jedoch verwendet werden, um Ihre App bereitzustellen.

Ausführen Ihrer App

Das CDK CLI muss wissen, wie Sie Ihre AWS CDK App ausführen. Wenn Sie das Projekt mit dem `cdk init` Befehl aus einer Vorlage erstellt haben, enthält die `-cdk.json` Datei Ihrer App einen `-app` Schlüssel. Dieser Schlüssel gibt den erforderlichen Befehl für die Sprache an, in der die App geschrieben ist. Wenn Ihre Sprache kompiliert werden muss, führt die Befehlszeile diesen Schritt aus, bevor Sie die App ausführen, sodass Sie dies nicht vergessen können.

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"
}
```

Wenn Sie Ihr Projekt nicht mit dem CDK erstellt haben, können Sie die in angegebene Befehlszeile überschreiben. Wenn Sie die in angegebene Befehlszeile überschreiben möchten, können Sie bei der Ausgabe des `cdk` Befehls die `--app` Option verwenden.

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

Der *ausführbare* Teil des Befehls gibt den Befehl an, der zum Ausführen Ihrer CDK-Anwendung ausgeführt werden soll. Verwenden Sie Anführungszeichen wie gezeigt, da solche Befehle Leerzeichen enthalten. Der Befehl *cdk-command* ist ein Unterbefehl wie `synth` oder `deploy`, der dem CDK mitteilt, CLI was Sie mit Ihrer App machen möchten. Folgen Sie diesem Schritt mit allen zusätzlichen Optionen, die für diesen Unterbefehl erforderlich sind.

Die AWS CDK CLI kann auch direkt mit einer bereits synthetisierten Cloud-Baugruppe interagieren. Übergeben Sie dazu das Verzeichnis, in dem die Cloud-Baugruppe gespeichert ist. Im folgenden Beispiel werden die Stacks aufgelistet, die in der unter gespeicherten Cloud-Baugruppe definiert sind. `./my-cloud-assembly`.

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

Stacks

Ein AWS Cloud Development Kit (AWS CDK) Stack ist eine Sammlung von einem oder mehreren Konstrukten, die Ressourcen definieren. Jeder CDK-Stack steht für einen AWS CloudFormation Stack in Ihrer CDK-App. Bei der Bereitstellung werden Konstrukte innerhalb eines Stacks als eine einzige Einheit bereitgestellt, die als Stack bezeichnet wird. Weitere Informationen zu AWS CloudFormation Stacks finden Sie unter [Arbeiten mit Stacks im Benutzerhandbuch](#). AWS CloudFormation

Da CDK-Stacks über AWS CloudFormation Stacks implementiert werden, AWS CloudFormation gelten Kontingente und Einschränkungen. [Weitere Informationen finden Sie unter Kontingente.AWS CloudFormation](#)

Themen

- [Stapel definieren](#)
- [Arbeiten mit -Stacks](#)

Stapel definieren

Stacks werden im Kontext einer App definiert. Sie definieren einen Stack mithilfe der [Stack](#) Klasse aus der AWS Construct-Bibliothek. Stacks können auf eine der folgenden Arten definiert werden:

- Direkt im Rahmen der App.
- Indirekt durch ein beliebiges Konstrukt innerhalb des Baums.

Das folgende Beispiel definiert eine CDK-App, die zwei Stacks enthält:

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();
```

Das folgende Beispiel ist ein gängiges Muster für die Definition eines Stacks in einer separaten Datei. Hier erweitern oder erben wir die Stack Klasse und definieren einen Konstruktor, der `scopeid`, und akzeptiert. `props` Dann rufen wir den Stack Basisklassenkonstruktor `super` mit dem empfangenen `scope`, und auf. `id` `props`

TypeScript

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

    //...
  }
}
```

JavaScript

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

    //...
  }
}
```

Python

```
class HelloCdkStack(Stack):

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

        # ...
```

Java

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

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

        // ...
    }
}
```

C#

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

```
    }  
  }  
}
```

Go

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

Im folgenden Beispiel wird eine Stack-Klasse mit dem Namen `MyFirstStack`, die einen einzelnen Amazon S3 S3-Bucket enthält.

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)
awsdk.Stack {
    var sprops awsdk.StackProps
    if props != nil {
        sprops = props.StackProps
    }
}
```

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

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

Dieser Code hat jedoch nur einen Stack deklariert. Damit der Stack tatsächlich zu einer AWS CloudFormation Vorlage synthetisiert und bereitgestellt werden kann, muss er instanziiert werden. Und wie alle CDK-Konstrukte muss er in einem bestimmten Kontext instanziiert werden. Das App ist dieser Kontext.

Wenn Sie die AWS CDK Standard-Entwicklungsvorlage verwenden, werden Ihre Stacks in derselben Datei instanziiert, in der Sie das Objekt instanziiieren. App

TypeScript

Die nach Ihrem Projekt benannte Datei (z. B. `hello-cdk.ts`) im Ordner Ihres Projekts. `bin`

JavaScript

Die nach Ihrem Projekt benannte Datei (z. B. `hello-cdk.js`) im `bin` Ordner Ihres Projekts.

Python

Die Datei `app.py` im Hauptverzeichnis Ihres Projekts.

Java

Die Datei mit dem Namen ist `ProjectNameApp.java` `HelloCdkApp.java` beispielsweise tief unter dem `src/main` Verzeichnis verschachtelt.

C#

Die Datei mit dem Namen `Program.cs` unter `src\ProjectName`, zum Beispiels `src\HelloCdk\Program.cs`.

Die Stack-API

Das [Stack-Objekt](#) bietet eine umfangreiche API, einschließlich der folgenden:

- `Stack.of(construct)`— Eine statische Methode, die den Stack zurückgibt, in dem ein Konstrukt definiert ist. Dies ist nützlich, wenn Sie innerhalb eines wiederverwendbaren

Konstrukts mit einem Stapel interagieren müssen. Der Aufruf schlägt fehl, wenn ein Stack im Gültigkeitsbereich nicht gefunden werden kann.

- `stack.stackName(Python:stack_name)` — Gibt den physischen Namen des Stacks zurück. Wie bereits erwähnt, haben alle AWS CDK Stacks einen physikalischen Namen, den sie während der Synthese auflösen AWS CDK können.
- `stack.region` und `stack.account` — Gibt die AWS Region bzw. das Konto zurück, in dem dieser Stack bereitgestellt werden soll. Diese Eigenschaften geben einen der folgenden Werte zurück:
 - Das Konto oder die Region, das bei der Definition des Stacks explizit angegeben wurde
 - Ein string-codiertes Token, das in die AWS CloudFormation Pseudo-Parameter für Account und Region aufgelöst wird, um anzuzeigen, dass dieser Stack umgebungsunabhängig ist

Hinweise dazu, wie Umgebungen für Stacks bestimmt werden, finden Sie unter [the section called "Umgebungen"](#)

- `stack.addDependency(stack)` (Python: `stack.add_dependency(stack)`) — Kann verwendet werden, um die Reihenfolge der Abhängigkeiten zwischen zwei Stacks explizit zu definieren. Diese Reihenfolge wird vom `cdk deploy` Befehl respektiert, wenn mehrere Stacks gleichzeitig bereitgestellt werden.
- `stack.tags` — Gibt einen zurück [TagManager](#), den Sie zum Hinzufügen oder Entfernen von Tags auf Stackebene verwenden können. Dieser Tag-Manager markiert alle Ressourcen innerhalb des Stacks und markiert auch den Stack selbst, wenn er durch diesen erstellt wird. AWS CloudFormation
- `stack.partition`, `stack.urlSuffix` (Python: `url_suffix`), `stack.stackId` (Python: `stack_id`) und `stack.notificationArn` (Python: `notification_arn`) — Gibt Tokens zurück, die in die jeweiligen AWS CloudFormation Pseudo-Parameter aufgelöst werden, wie `{ "Ref": "AWS::Partition" }` z. Diese Token sind dem spezifischen Stack-Objekt zugeordnet, sodass das AWS CDK Framework stapelübergreifende Referenzen identifizieren kann.
- `stack.availabilityZones` (Python: `availability_zones`) — Gibt den Satz von Availability Zones zurück, die in der Umgebung verfügbar sind, in der dieser Stack bereitgestellt wird. Bei umgebungsunabhängigen Stacks wird dadurch immer ein Array mit zwei Availability Zones zurückgegeben. Bei umgebungsspezifischen Stacks AWS CDK fragt das System die Umgebung ab und gibt genau den Satz von Availability Zones zurück, die in der von Ihnen angegebenen Region verfügbar sind.
- `stack.parseArn(arn)` und `stack.formatArn(comps)` (Python: `parse_arn`, `format_arn`) — Kann verwendet werden, um mit Amazon Resource Names (ARNs) zu arbeiten.

- `stack.toJsonString(obj)`(Python:`to_json_string`) — Kann verwendet werden, um ein beliebiges Objekt als JSON-String zu formatieren, der in eine AWS CloudFormation Vorlage eingebettet werden kann. Das Objekt kann Token, Attribute und Verweise enthalten, die nur während der Bereitstellung aufgelöst werden.
- `stack.templateOptions`(Python:`template_options`) — Wird verwendet, um AWS CloudFormation Vorlagenoptionen wie Transformation, Beschreibung und Metadaten für Ihren Stack anzugeben.

Arbeiten mit -Stacks

Verwenden Sie den Befehl, um alle Stacks in einer CDK-App aufzulisten. `cdk ls` Das vorherige Beispiel würde Folgendes ausgeben:

```
stack1
stack2
```

Stacks werden als Teil eines AWS CloudFormation Stacks in einer AWS [Umgebung](#) bereitgestellt. Die Umgebung deckt ein bestimmtes AWS-Konto und AWS-Region ab.

Wenn Sie den `cdk synth` Befehl für eine App mit mehreren Stacks ausführen, enthält die Cloud-Assembly eine separate Vorlage für jede Stack-Instanz. Selbst wenn es sich bei den beiden Stacks um Instanzen derselben Klasse handelt, werden sie als zwei einzelne AWS CDK Vorlagen ausgegeben.

Sie können jede Vorlage synthetisieren, indem Sie den Stacknamen im Befehl angeben. `cdk synth` Das folgende Beispiel synthetisiert die Vorlage für `stack1`.

```
$ cdk synth stack1
```

[Dieser Ansatz unterscheidet sich konzeptionell von der üblichen Verwendung von AWS CloudFormation Vorlagen, bei der eine Vorlage mehrfach bereitgestellt und über Parameter parametrisiert werden kann.](#) [AWS CloudFormation](#) AWS CloudFormation Parameter können zwar in der definiert werden AWS CDK, es wird jedoch generell davon abgeraten, da AWS CloudFormation Parameter erst während der Bereitstellung aufgelöst werden. Das bedeutet, dass Sie ihren Wert nicht in Ihrem Code bestimmen können.

Um beispielsweise eine Ressource auf der Grundlage eines Parameterwerts bedingt in Ihre App aufzunehmen, müssen Sie eine [AWS CloudFormation Bedingung](#) einrichten und die Ressource damit

kennzeichnen. Dabei wird AWS CDK ein Ansatz verfolgt, bei dem konkrete Vorlagen zum Zeitpunkt der Synthese aufgelöst werden. Daher können Sie eine `if`-Anweisung verwenden, um den Wert zu überprüfen, um festzustellen, ob eine Ressource definiert oder ein bestimmtes Verhalten angewendet werden sollte.

Note

AWS CDK Dadurch wird während der Synthesedauer so viel Auflösung wie möglich erreicht, um eine idiomatische und natürliche Verwendung Ihrer Programmiersprache zu ermöglichen.

Wie jedes andere Konstrukt können Stapel zu Gruppen zusammengesetzt werden. Der folgende Code zeigt ein Beispiel für einen Service, der aus drei Stacks besteht: einer Steuerungsebene, einer Datenebene und Monitoring-Stacks. Das Dienstkonstrukt ist zweimal definiert: einmal für die Betaumgebung und einmal für die Produktionsumgebung.

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();
    }
}
```

Diese AWS CDK App besteht letztendlich aus sechs Stacks, drei für jede Umgebung:

```
$ cdk ls
```

```
betacpDA8372D3
betadataE23DB2BA
betamon632BD457
prodcp187264CE
proddataF7378CE5
prodmon631A1083
```

Die physikalischen Namen der AWS CloudFormation Stapel werden automatisch AWS CDK anhand des Konstruktpfads des Stacks im Baum bestimmt. Standardmäßig wird der Name eines Stacks von der Konstrukt-ID des Stack Objekts abgeleitet. Sie können jedoch einen expliziten Namen angeben, indem Sie die `stackName` Requisite (in Python, `stack_name`) wie folgt verwenden.

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"  
});
```

Verschachtelte Stacks

Das [NestedStack](#) Konstrukt bietet eine Möglichkeit, das Limit von AWS CloudFormation 500 Ressourcen für Stacks zu umgehen. Ein verschachtelter Stapel zählt nur als eine Ressource in dem Stapel, der ihn enthält. Er kann jedoch bis zu 500 Ressourcen enthalten, einschließlich zusätzlicher verschachtelter Stacks.

Der Gültigkeitsbereich eines verschachtelten Stacks muss ein Stack Oder-Konstrukt sein. `NestedStack` Der verschachtelte Stapel muss innerhalb seines übergeordneten Stacks nicht lexikalisch deklariert werden. Bei der Instanziierung des verschachtelten Stacks muss nur der

übergeordnete Stapel als ersten Parameter (`scope`) übergeben werden. Abgesehen von dieser Einschränkung funktioniert die Definition von Konstrukten in einem verschachtelten Stack genauso wie in einem normalen Stack.

Zum Zeitpunkt der Synthese wird der verschachtelte Stack zu einer eigenen AWS CloudFormation Vorlage synthetisiert, die bei der Bereitstellung in den AWS CDK Staging-Bucket hochgeladen wird. Verschachtelte Stacks sind an ihren übergeordneten Stapel gebunden und werden nicht als unabhängige Bereitstellungsartefakte behandelt. Sie sind nicht in der Liste aufgeführt und können auch nicht von `cdk list` bereitgestellt werden. `cdk deploy`

[Verweise zwischen übergeordneten Stacks und verschachtelten Stacks werden automatisch in Stack-Parameter und Ausgaben in den generierten AWS CloudFormation Vorlagen übersetzt, wie bei jeder stapelübergreifenden Referenz.](#)

Warning

Änderungen der Sicherheitslage werden vor der Bereitstellung für verschachtelte Stacks nicht angezeigt. Diese Informationen werden nur für Stacks der obersten Ebene angezeigt.

Konstrukte

Konstrukte sind die Grundbausteine von Anwendungen. AWS Cloud Development Kit (AWS CDK) Ein Konstrukt ist eine Komponente innerhalb Ihrer Anwendung, die eine oder mehrere AWS CloudFormation Ressourcen und deren Konfiguration darstellt. Sie erstellen Ihre Anwendung Stück für Stück, indem Sie Konstrukte importieren und konfigurieren.

Konstrukte sind Klassen, die Sie in Ihre CDK-Apps importieren. Konstrukte sind in der Construct-Bibliothek verfügbar. AWS Sie können auch Ihre eigenen Konstrukte erstellen und verteilen oder Konstrukte verwenden, die von Drittanbietern erstellt wurden.

Konstrukte sind Teil des Construct Programming Model (CPM). Sie können mit anderen Tools wie CDK for Terraform (CDKtf), CDK for (CDK8s) und verwendet werden. Kubernetes Projen

Themen

- [AWS Bibliothek erstellen](#)
- [Konstrukte definieren](#)
- [Mit Konstrukten arbeiten](#)

- [Arbeiten mit Konstrukten von Drittanbietern](#)
- [Weitere Informationen](#)

AWS Bibliothek erstellen

Die AWS Construct Library enthält eine Sammlung von Konstrukten, die von entwickelt und verwaltet AWS werden. Sie ist in verschiedene Module unterteilt, die Konstrukte enthalten, die alle verfügbaren Ressourcen repräsentieren. AWS Referenzinformationen finden Sie in der [AWS CDK API-Referenz](#).

Das CDK-Hauptpaket wird aufgerufen `aws-cdk-lib` und enthält den Großteil der AWS Construct-Bibliothek. Es enthält auch Basisklassen wie `Stack` und `App`.

Der tatsächliche Paketname des CDK-Hauptpakets variiert je nach Sprache.

TypeScript

Installieren	<pre>npm install aws-cdk-lib</pre>
Import	<pre>import * as cdk from 'aws-cdk-lib';</pre>

JavaScript

Installieren	<pre>npm install aws-cdk-lib</pre>
Import	<pre>const cdk = require('aws-cdk-lib');</pre>

Python

Installieren	<pre>python -m pip install aws-cdk-lib</pre>
Import	<pre>import aws_cdk as cdk</pre>

Java

Inpom.xml, füge hinzu

```
Gruppensoftware.amazon.awscdk ;  
Artefakt aws-cdk-lib
```

Import

```
import software.amazon.aw  
scdk.App;
```

C#

Installieren

```
dotnet add package Amazon.CDK.Lib
```

Import

```
using Amazon.CDK;
```

Go

Installieren

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

Import

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

Note

Wenn Sie ein CDK-Projekt mit `erstellt habencdk init`, müssen Sie es nicht manuell installieren.
`aws-cdk-lib`

Die AWS Construct-Bibliothek enthält auch das [constructs](#) Paket mit der `Construct` Basisklasse. Es ist in einem eigenen Paket enthalten, da es neben dem auch von anderen konstruktbasierenden Tools verwendet wird AWS CDK, darunter CDK für Terraform und CDK für Kubernetes.

Zahlreiche Drittanbieter haben auch Konstrukte veröffentlicht, die mit dem kompatibel sind. AWS CDK Besuchen Sie [Construct Hub](#), um das Ökosystem der AWS CDK Konstruktpartner zu erkunden.

Konstruiere Levels

Konstrukte aus der AWS Construct Library sind in drei Stufen unterteilt. Jede Ebene bietet eine zunehmende Abstraktionsebene. Je höher die Abstraktion, desto einfacher ist die Konfiguration und erfordert weniger Fachwissen. Je niedriger die Abstraktion, desto mehr Anpassungen sind verfügbar und erfordern mehr Fachwissen.

Konstrukte der Stufe 1 (L1)

L1-Konstrukte, auch CFN-Ressourcen genannt, sind das Konstrukt der untersten Ebene und bieten keine Abstraktion. Jedes L1-Konstrukt ist direkt einer einzelnen Ressource zugeordnet. AWS CloudFormation Bei L1-Konstrukten importieren Sie ein Konstrukt, das eine bestimmte Ressource darstellt. AWS CloudFormation Anschließend definieren Sie die Eigenschaften der Ressource in Ihrer Konstruktinstanz.

L1-Konstrukte eignen sich hervorragend, wenn Sie mit der Definition Ihrer AWS Ressourceneigenschaften vertraut sind AWS CloudFormation und vollständige Kontrolle darüber benötigen.

In der AWS Construct-Bibliothek werden L1-Konstrukte beginnend mit `benanntCfn`, gefolgt von einem Bezeichner für die AWS CloudFormation Ressource, für die sie stehen. Das [CfnBucket](#) Konstrukt ist beispielsweise ein L1-Konstrukt, das eine Ressource darstellt. [AWS::S3::Bucket](#) AWS CloudFormation

[L1-Konstrukte werden anhand der Ressourcenspezifikation generiert.](#) [AWS CloudFormation](#) Wenn eine Ressource in existiert AWS CloudFormation, ist sie AWS CDK als L1-Konstrukt verfügbar. Es kann bis zu einer Woche dauern, bis neue Ressourcen oder Eigenschaften in der AWS Construct-Bibliothek verfügbar sind. Weitere Informationen finden Sie in der [Referenz zu AWS Ressourcen- und Eigenschaftstypen](#) im AWS CloudFormation Benutzerhandbuch.

Konstrukte der Ebene 2 (L2)

L2-Konstrukte, auch als kuratierte Konstrukte bekannt, wurden vom CDK-Team sorgfältig entwickelt und sind in der Regel der am häufigsten verwendete Konstrukttyp. L2-Konstrukte werden, ähnlich wie L1-Konstrukte, direkt einzelnen Ressourcen zugeordnet. AWS CloudFormation Im Vergleich zu L1-Konstrukten bieten L2-Konstrukte über eine intuitive

absichtsbasierte API eine Abstraktion auf höherer Ebene. L2-Konstrukte beinhalten sinnvolle Standardkonfigurationen für Eigenschaften, bewährte Sicherheitsrichtlinien und generieren einen Großteil des Standardcodes und der Glue-Logik für Sie.

L2-Konstrukte bieten auch Hilfsmethoden für die meisten Ressourcen, mit denen Eigenschaften, Berechtigungen, ereignisbasierte Interaktionen zwischen Ressourcen und mehr einfacher und schneller definiert werden können.

Die [s3.Bucket](#) Klasse ist ein Beispiel für ein L2-Konstrukt für eine Bucket-Ressource von Amazon Simple Storage Service (Amazon S3).

Die AWS Construct-Bibliothek enthält L2-Konstrukte, die als stabil und produktionsbereit gekennzeichnet sind. L2-Konstrukte, die sich in der Entwicklung befinden, werden als experimentell eingestuft und in einem separaten Modul angeboten.

Konstrukte der Stufe 3 (L3)

L3-Konstrukte, auch Muster genannt, stellen die höchste Abstraktionsebene dar. Jedes L3-Konstrukt kann eine Sammlung von Ressourcen enthalten, die so konfiguriert sind, dass sie zusammenarbeiten, um eine bestimmte Aufgabe oder einen bestimmten Dienst innerhalb Ihrer Anwendung auszuführen. L3-Konstrukte werden verwendet, um ganze AWS Architekturen für bestimmte Anwendungsfälle in Ihrer Anwendung zu erstellen.

Um komplette Systementwürfe oder wesentliche Teile eines größeren Systems bereitzustellen, bieten L3-Konstrukte individuelle Standardkonfigurationen für Eigenschaften. Sie basieren auf einem bestimmten Ansatz zur Lösung eines Problems und zur Bereitstellung einer Lösung. Mit L3-Konstrukten können Sie schnell und mit dem geringsten Aufwand an Eingabe und Code mehrere Ressourcen erstellen und konfigurieren.

Die [ecsPatterns.ApplicationLoadBalancedFargateService](#) Klasse ist ein Beispiel für ein L3-Konstrukt, das einen AWS Fargate Service darstellt, der auf einem Amazon Elastic Container Service (Amazon ECS) -Cluster ausgeführt wird und von einem Application Load Balancer unterstützt wird.

Ähnlich wie L2-Konstrukte sind L3-Konstrukte, die für den produktiven Einsatz bereit sind, in der Construct Library enthalten. AWS Diejenigen, die sich in der Entwicklung befinden, werden in separaten Modulen angeboten.

Konstrukte definieren

Composition

Die Komposition ist das Schlüsselmuster für die Definition von Abstraktionen auf höherer Ebene durch Konstrukte. Ein Konstrukt auf hoher Ebene kann aus einer beliebigen Anzahl von Konstrukten niedrigerer Ebenen zusammengesetzt werden. Von unten nach oben betrachtet, verwenden Sie Konstrukte, um die einzelnen AWS Ressourcen zu organisieren, die Sie bereitstellen möchten. Sie verwenden die für Ihren Zweck geeigneten Abstraktionen mit so vielen Ebenen, wie Sie benötigen.

Bei der Komposition definieren Sie wiederverwendbare Komponenten und teilen sie wie jeden anderen Code. Ein Team kann beispielsweise ein Konstrukt definieren, das die Best Practices des Unternehmens für eine Amazon DynamoDB-Tabelle implementiert, einschließlich Backup, globaler Replikation, automatischer Skalierung und Überwachung. Das Team kann das Konstrukt intern mit anderen Teams oder öffentlich teilen.

Teams können Konstrukte wie jedes andere Bibliothekspaket verwenden. Wenn die Bibliothek aktualisiert wird, erhalten Entwickler Zugriff auf die Verbesserungen und Bugfixes der neuen Version, ähnlich wie bei jeder anderen Codebibliothek.

Initialisierung

Konstrukte werden in Klassen implementiert, die die [Construct](#)-Basisklasse erweitern. Sie definieren ein Konstrukt, indem Sie die Klasse instanziiieren. Alle Konstrukte benötigen drei Parameter, wenn sie initialisiert werden:

- Gültigkeitsbereich — Das übergeordnete Objekt oder der Eigentümer des Konstrukts. Dies kann entweder ein Stapel oder ein anderes Konstrukt sein. Der Geltungsbereich bestimmt die Position des [Konstrukts im Konstruktbaum](#). Normalerweise sollten Sie für den Gültigkeitsbereich `this` (`self` in Python) übergeben, was das aktuelle Objekt darstellt.
- `id` — Ein [Bezeichner](#), der innerhalb des Bereichs eindeutig sein muss. Der Bezeichner dient als Namespace für alles, was innerhalb des Konstrukts definiert ist. Er wird verwendet, um eindeutige Bezeichner wie [Ressourcennamen](#) und AWS CloudFormation logische IDs zu generieren.

Identifikatoren müssen nur innerhalb eines Bereichs eindeutig sein. Auf diese Weise können Sie Konstrukte instanziiieren und wiederverwenden, ohne sich Gedanken über die darin enthaltenen Konstrukte und Bezeichner machen zu müssen. Außerdem können Sie Konstrukte zu Abstraktionen auf höherer Ebene zusammensetzen. Darüber hinaus ermöglichen es Bereiche, auf

Gruppen von Konstrukten gleichzeitig zu verweisen. Beispiele hierfür sind das [Markieren](#) oder die Angabe, wo die Konstrukte eingesetzt werden sollen.

- `props` — Je nach Sprache eine Reihe von Eigenschaften oder Schlüsselwortargumenten, die die ursprüngliche Konfiguration des Konstrukts definieren. Konstrukte auf höherer Ebene bieten mehr Standardwerte, und wenn alle Prop-Elemente optional sind, können Sie den Parameter `props` komplett weglassen.

Konfiguration

Die meisten Konstrukte akzeptieren `props` als drittes Argument (oder in Python Schlüsselwortargumente) eine Name/Wertesammlung, die die Konfiguration des Konstrukts definiert. Das folgende Beispiel definiert einen Bucket mit aktivierter AWS Key Management Service (AWS KMS) -Verschlüsselung und aktiviertem statischem Website-Hosting. Da es nicht explizit einen Verschlüsselungsschlüssel spezifiziert, definiert das Bucket Konstrukt einen neuen `kms.Key` und ordnet ihn dem Bucket zu.

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"),
})
```

Interaktion mit Konstrukten

Konstrukte sind Klassen, die die Basisklasse [Construct](#) erweitern. Nachdem Sie ein Konstrukt instanziiert haben, stellt das Konstruktobjekt eine Reihe von Methoden und Eigenschaften zur Verfügung, mit denen Sie mit dem Konstrukt interagieren und es als Referenz an andere Teile des Systems weitergeben können.

Das AWS CDK Framework schränkt die APIs von Konstrukten nicht ein. Autoren können jede beliebige API definieren. Die AWS Konstrukte, die in der AWS Construct-Bibliothek enthalten sind `s3.Bucket`, folgen jedoch Richtlinien und gängigen Mustern. Dadurch wird ein einheitliches Benutzererlebnis für alle AWS Ressourcen gewährleistet.

Die meisten AWS Konstrukte verfügen über eine Reihe von [Erteilungsmethoden](#), mit denen Sie einem Prinzipal AWS Identity and Access Management (IAM) Berechtigungen für dieses Konstrukt gewähren können. Im folgenden Beispiel wird der IAM-Gruppe die `data-science` Berechtigung erteilt, aus dem Amazon S3 S3-Bucket `raw-data` zu lesen.

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)
```

Ein anderes gängiges Muster besteht darin, dass AWS Konstrukte eines der Ressourcenattribute anhand von Daten festlegen, die an anderer Stelle bereitgestellt wurden. Zu den Attributen können Amazon-Ressourcennamen (ARNs), Namen oder URLs gehören.

Der folgende Code definiert eine AWS Lambda Funktion und ordnet sie über die URL der Warteschlange in einer Umgebungsvariablen einer Amazon Simple Queue Service (Amazon SQS) - Warteschlange zu.

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()),
    },
})

```

Informationen zu den gängigsten API-Mustern in der AWS Construct-Bibliothek finden Sie unter [the section called “Ressourcen”](#).

Das App- und Stack-Konstrukt

Die [Stack](#) Klassen [App](#) und aus der AWS Construct-Bibliothek sind einzigartige Konstrukte. Im Vergleich zu anderen Konstrukten konfigurieren sie AWS Ressourcen nicht eigenständig. Stattdessen werden sie verwendet, um Kontext für Ihre anderen Konstrukte bereitzustellen. Alle Konstrukte, die AWS Ressourcen darstellen, müssen direkt oder indirekt im Rahmen eines Stack Konstrukts definiert werden. StackKonstrukte werden innerhalb des Gültigkeitsbereichs eines App Konstrukts definiert.

Weitere Informationen zu CDK-Apps finden Sie unter. [AWS CDK Apps](#) Weitere Informationen zu CDK-Stacks finden Sie unter. [Stacks](#)

Das folgende Beispiel definiert eine App mit einem einzigen Stack. Innerhalb des Stacks wird ein L2-Konstrukt verwendet, um eine Amazon S3 S3-Bucket-Ressource zu konfigurieren.

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

In der `HelloCdkStack.java` Datei definierter Stack:

```
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();
    }
}
```

In der `HelloCdkApp.java` Datei definierte App:

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

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

return stack
}
```

Mit Konstrukten arbeiten

Mit L1-Konstrukten arbeiten

L1-Konstrukte sind direkt einzelnen Ressourcen zugeordnet. AWS CloudFormation Sie müssen die für die Ressource erforderliche Konfiguration angeben.

In diesem Beispiel erstellen wir ein bucket Objekt mit dem CfnBucket L1-Konstrukt:

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"),  
})
```

Konstrukteigenschaften, bei denen es sich nicht um einfache Boolesche Werte, Zeichenketten, Zahlen oder Container handelt, werden in den unterstützten Sprachen unterschiedlich behandelt.

TypeScript

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

JavaScript

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

```
    ]]  
  }  
});
```

Python

In Python werden diese Eigenschaften durch Typen repräsentiert, die als innere Klassen des L1-Konstrukts definiert sind. Beispielsweise `CfnBucket` erfordert die optionale Eigenschaft `cors_configuration` von a einen Wrapper des Typs `CfnBucket.CorsConfigurationProperty`. Hier definieren wir für `cors_configuration` eine `CfnBucket` Instanz.

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

Java

In Java werden diese Eigenschaften durch Typen repräsentiert, die als innere Klassen des L1-Konstrukts definiert sind. Beispielsweise `CfnBucket` erfordert die optionale Eigenschaft `corsConfiguration` von a einen Wrapper des Typs `CfnBucket.CorsConfigurationProperty`. Hier definieren wir für `corsConfiguration` eine `CfnBucket` Instanz.

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

C#

In C# werden diese Eigenschaften durch Typen repräsentiert, die als innere Klassen des L1-Konstrukts definiert sind. Beispielsweise `CfnBucket` erfordert die optionale Eigenschaft `CorsConfiguration` von a einen Wrapper des Typs `CfnBucket.CorsConfigurationProperty`. Hier definieren wir für `CorsConfiguration` eine `CfnBucket` Instanz.

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

Go

In Go werden diese Typen mit dem Namen des L1-Konstrukts, einem Unterstrich und dem Eigenschaftsnamen benannt. Beispielsweise `CfnBucket` erfordert die optionale Eigenschaft `CorsConfiguration` von a einen Wrapper des Typs `CfnBucket_CorsConfigurationProperty`. Hier definieren wir für `CorsConfiguration` eine `CfnBucket` Instanz.

```
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

Sie können L2-Eigenschaftstypen nicht mit L1-Konstrukten verwenden oder umgekehrt. Verwenden Sie bei der Arbeit mit L1-Konstrukten immer die Typen, die für das verwendete L1-Konstrukt definiert sind. Verwenden Sie keine Typen aus anderen L1-Konstrukten (einige haben möglicherweise denselben Namen, sind aber nicht vom gleichen Typ).

Einige unserer sprachspezifischen API-Referenzen enthalten derzeit Fehler in den Pfaden zu L1-Eigenschaftstypen oder dokumentieren diese Klassen überhaupt nicht. Wir hoffen, das bald beheben zu können. Denken Sie in der Zwischenzeit daran, dass solche Typen immer innere Klassen des L1-Konstrukts sind, mit dem sie verwendet werden.

Mit L2-Konstrukten arbeiten

Im folgenden Beispiel definieren wir einen Amazon S3 S3-Bucket, indem wir ein Objekt aus dem [Bucket](#)L2-Konstrukt erstellen:

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` ist nicht der Name des Buckets, der AWS CloudFormation erstellt wird. Es ist eine logische Kennung, die dem neuen Konstrukt im Kontext Ihrer CDK-App zugewiesen wurde. Der Wert [PhysicalName](#) wird verwendet, um die Ressource zu benennen. AWS CloudFormation

Arbeiten mit Konstrukten von Drittanbietern

[Construct Hub](#) ist eine Ressource, die Ihnen hilft AWS, weitere Konstrukte von Drittanbietern und der Open-Source-CDK-Community zu entdecken.

Schreiben Sie Ihre eigenen Konstrukte

Sie können nicht nur bestehende Konstrukte verwenden, sondern auch Ihre eigenen Konstrukte schreiben und sie von jedem in seinen Apps verwenden lassen. Alle Konstrukte sind gleich in der. AWS CDK Konstrukte aus der AWS Konstruktbibliothek werden genauso behandelt wie Konstrukte aus einer Drittanbieterbibliothek, die über NPM Maven, oder veröffentlicht wurden. PyPI Konstrukte, die im internen Paket-Repository Ihres Unternehmens veröffentlicht wurden, werden ebenfalls auf die gleiche Weise behandelt.

Um ein neues Konstrukt zu deklarieren, erstellen Sie im `constructs` Paket eine Klasse, die die [Construct-Basisklasse](#) erweitert, und folgen Sie dann dem Muster für Initialisierungsargumente.

Das folgende Beispiel zeigt, wie ein Konstrukt deklariert wird, das einen Amazon S3 S3-Bucket darstellt. Der S3-Bucket sendet jedes Mal, wenn jemand eine Datei in ihn hochlädt, eine Amazon Simple Notification Service (Amazon SNS) -Benachrichtigung.

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

Unser `NotifyingBucket` Konstrukt erbt nicht von `Bucket` sondern von `Construct`. Wir verwenden Komposition, nicht Vererbung, um einen Amazon S3-Bucket und ein Amazon SNS Thema zu bündeln. Im Allgemeinen wird bei der Entwicklung von AWS CDK Konstrukten die Komposition der Vererbung vorgezogen.

Der `NotifyingBucket` Konstruktor hat eine typische Konstruktursignatur: `scope`, `id`, und `props`. Das letzte Argument, `props`, ist optional (erhält den Standardwert `{}`), da alle Requisiten optional sind. (Die `Construct` Basisklasse akzeptiert kein `props` Argument.) Sie könnten eine Instanz dieses Konstrukts in Ihrer App definieren `props`, ohne zum Beispiel:

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)
```

Oder Sie könnten props (in Java einen zusätzlichen Parameter) verwenden, um das Pfadpräfix anzugeben, nach dem gefiltert werden soll, zum Beispiel:

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

Go

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

In der Regel möchten Sie auch einige Eigenschaften oder Methoden für Ihre Konstrukte verfügbar machen. Es ist nicht sehr nützlich, ein Thema hinter Ihrem Konstrukt zu verbergen, da Benutzer Ihres Konstrukts es nicht abonnieren können. Durch das Hinzufügen einer `topic` Eigenschaft können Verbraucher auf das innere Thema zugreifen, wie im folgenden Beispiel gezeigt wird:

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

Um dies in Go zu tun, benötigen wir ein wenig zusätzliche Klempnerarbeiten. Unsere ursprüngliche `NewNotifyingBucket` Funktion gab eine `awss3.Bucket` zurück. Wir müssen die Erweiterung `Bucket` um ein `topic` Mitglied erweitern, indem wir eine `NotifyingBucket` Struktur erstellen. Unsere Funktion gibt dann diesen Typ zurück.

```

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
}

```

Jetzt können Verbraucher das Thema abonnieren, zum Beispiel:

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))
```

Weitere Informationen

Das folgende Video bietet einen umfassenden Überblick über CDK-Konstrukte und erklärt, wie Sie sie in Ihren CDK-Apps verwenden können.

[CDK-Konstrukte erklärt](#)

Umgebungen

Eine Umgebung besteht aus dem AWS-Konto und AWS-Region , in dem Sie einen AWS Cloud Development Kit (AWS CDK) Stack bereitstellen.

AWS-Konto

Wenn Sie eine erstellen AWS-Konto, erhalten Sie eine Konto-ID. Diese ID ist eine 12-stellige Zahl, z. B. 012345678901, die Ihr Konto eindeutig identifiziert. Weitere Informationen finden Sie im Referenzhandbuch unter Identifikatoren [anzeigen AWS-Konto](#).AWS Account Management

Im Folgenden finden Sie ein Beispiel für die Angabe dieser Umgebungsvariablen in Ihrem CDK-Code:

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

Im Folgenden finden Sie ein Beispiel für die Übergabe von mit dem `user1` Profil verknüpften Werten aus Ihren Anmelde- und Konfigurationsdateien CLI mithilfe der Option `--profile` an das CDK. `--profile` Werte aus diesen Dateien werden an Ihre Umgebungsvariablen übergeben:

```
$ cdk deploy myStack --profile user1
```

Anstatt Werte aus den Anmeldeinformationen und Konfigurationsdateien zu verwenden, können Sie Umgebungswerte auch fest in Ihrem CDK-Code codieren. Im Folgenden wird ein Beispiel gezeigt:

```
const envEU = { account: '238383838383', region: 'eu-west-1' };
const envUSA = { account: '837873873873', region: 'us-west-2' };

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

Weitere Informationen

Erste Schritte mit der Verwendung von Umgebungen mit dem finden Sie AWS CDK unter [Konfigurieren Sie Umgebungen für die Verwendung mit dem AWS CDK](#)

Bootstrapping

Bootstrapping ist der Prozess, bei dem Ihre AWS Umgebung für die Verwendung mit dem vorbereitet wird. AWS Cloud Development Kit (AWS CDK) Bevor Sie einen CDK-Stack in einer AWS Umgebung bereitstellen, muss die Umgebung zunächst gebootet werden.

Themen

- [Was ist Bootstrapping?](#)
- [Wie funktioniert Bootstrapping?](#)

- [Weitere Informationen](#)

Was ist Bootstrapping?

Bootstrapping bereitet Ihre AWS Umgebung vor, indem es bestimmte AWS Ressourcen in Ihrer Umgebung bereitstellt, die von der verwendet werden. AWS CDK Diese umfasst die folgenden Funktionen:

- Amazon Simple Storage Service (Amazon S3) -Bucket — Wird zum Speichern Ihrer CDK-Projektdateien wie AWS Lambda Funktionscode und Ressourcen verwendet.
- Amazon Elastic Container Registry (Amazon ECR) -Repository — Wird hauptsächlich zum Speichern von Docker Bildern verwendet.
- AWS Identity and Access Management (IAM) -Rollen — Konfiguriert, um Berechtigungen zu gewähren, die für die Durchführung von Bereitstellungen AWS CDK erforderlich sind.

Wie funktioniert Bootstrapping?

Ressourcen und ihre Konfiguration, die vom CDK verwendet werden, sind in einer Vorlage definiert. AWS CloudFormation Diese Vorlage wird vom CDK-Team erstellt und verwaltet. Die neueste Version dieser Vorlage finden Sie [bootstrap-template.yaml](#) im GitHubaws-cdk-Repository.

Um eine Umgebung zu booten, verwenden Sie den AWS CDK Befehl Command Line Interface ().AWS CDK CLI `cdk bootstrap` Das CDK CLI ruft die Vorlage ab und stellt sie als Stack bereit, der AWS CloudFormation als Bootstrap-Stack bezeichnet wird. Standardmäßig lautet der Stack-Name. `CDKToolkit` Durch die Bereitstellung dieser Vorlage CloudFormation werden die Ressourcen in Ihrer Umgebung bereitgestellt. Nach der Bereitstellung wird der Bootstrap-Stack in der AWS CloudFormation Konsole Ihrer Umgebung angezeigt.

Sie können das Bootstrapping auch anpassen, indem Sie die Vorlage ändern oder die CLI CDK-Optionen mit dem Befehl verwenden. `cdk bootstrap`

AWS Umgebungen sind unabhängig. Für jede Umgebung, die Sie mit der verwenden möchten, AWS CDK muss zuerst ein Bootstrapping durchgeführt werden.

Weitere Informationen

Anweisungen zum Bootstrapping Ihrer Umgebung finden Sie unter. [Bootstrap Ihre Umgebung für die Verwendung mit dem AWS CDK](#)

Ressourcen

Ressourcen sind das, was Sie für die Verwendung AWS-Services in Ihren Anwendungen konfigurieren. Ressourcen sind ein Feature von AWS CloudFormation. Indem Sie Ressourcen und ihre Eigenschaften in einer AWS CloudFormation Vorlage konfigurieren, können Sie diese bereitstellen, AWS CloudFormation um Ihre Ressourcen bereitzustellen. Mit dem AWS Cloud Development Kit (AWS CDK) können Sie Ressourcen mithilfe von Konstrukten konfigurieren. Anschließend stellen Sie Ihre CDK-App bereit. Dazu müssen Sie eine AWS CloudFormation Vorlage synthetisieren und diese bereitstellen, AWS CloudFormation um Ihre Ressourcen bereitzustellen.

Themen

- [Konfiguration von Ressourcen mithilfe von Konstrukten](#)
- [Ressourcen referenzieren](#)
- [Physische Namen der Ressourcen](#)
- [Übergeben von eindeutigen Ressourcen-Identifikatoren](#)
- [Erteilen von Berechtigungen zwischen Ressourcen](#)
- [Metriken und Alarmer für Ressourcen](#)
- [Netzwerkdatenverkehr](#)
- [Behandlung von Ereignissen](#)
- [Richtlinien zur Entfernung](#)

Konfiguration von Ressourcen mithilfe von Konstrukten

Wie unter [beschrieben](#) [the section called "Konstrukte"](#), AWS CDK bietet eine umfangreiche Klassenbibliothek mit Konstrukten, sogenannten AWS Konstrukten, die alle Ressourcen repräsentieren. AWS

Um eine Instanz einer Ressource mit dem entsprechenden Konstrukt zu erstellen, übergeben Sie den Gültigkeitsbereich als erstes Argument, die logische ID des Konstrukts und eine Reihe von Konfigurationseigenschaften (Requisiten). Hier erfahren Sie beispielsweise, wie Sie mithilfe des [SQS.Queue-Konstrukts aus der Construct Library eine Amazon SQS-Warteschlange](#) mit AWS KMS Verschlüsselung erstellen. 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
});
```

Go

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



```
sqs.NewQueue(stack, jsii.String("MyQueue"), &sqs.QueueProps{
    Encryption: sqs.QueueEncryption_KMS_MANAGED,
})
```

Einige Konfigurationsrequisiten sind optional und haben in vielen Fällen Standardwerte. In einigen Fällen sind alle Requisiten optional, und das letzte Argument kann vollständig weggelassen werden.

Ressourcenattribute

Die meisten Ressourcen in der AWS Construct-Bibliothek stellen Attribute zur Verfügung, die bei der Bereitstellung von AWS CloudFormation aufgelöst werden. Attribute werden in Form von Eigenschaften für die Ressourcenklassen mit dem Typnamen als Präfix verfügbar gemacht. Das folgende Beispiel zeigt, wie die URL einer Amazon SQS SQS-Warteschlange mithilfe der Eigenschaft `queueUrl` (Python:`queue_url`) abgerufen wird.

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
```

Go

```
import (  
    "github.com/aws/aws-cdk-go/awscdk/v2"  
    "github.com/aws/jsii-runtime-go"  
    sqs "github.com/aws/aws-cdk-go/awscdk/v2/awssqs"  
)  
  
queue := sqs.NewQueue(stack, jsii.String("MyQueue"), &sqs.QueueProps{})  
url := queue.QueueUrl() // => A string representing a deploy-time value
```

Informationen darüber, wie die Bereitstellungszeitattribute als AWS CDK Zeichenketten kodiert, finden Sie unter [the section called "Token"](#).

Ressourcen referenzieren

Bei der Konfiguration von Ressourcen müssen Sie häufig auf Eigenschaften einer anderen Ressource verweisen. Im Folgenden sind einige Beispiele aufgeführt:

- Eine Amazon Elastic Container Service (Amazon ECS) -Ressource benötigt einen Verweis auf den Cluster, auf dem sie ausgeführt wird.
- Für eine CloudFront Amazon-Distribution ist ein Verweis auf den Amazon Simple Storage Service (Amazon S3) -Bucket erforderlich, der den Quellcode enthält.

Sie können auf eine der folgenden Arten auf Ressourcen verweisen:

- Indem Sie eine in Ihrer CDK-App definierte Ressource übergeben, entweder im selben Stack oder in einem anderen
- Indem Sie ein Proxy-Objekt übergeben, das auf eine in Ihrem AWS Konto definierte Ressource verweist und aus einer eindeutigen Kennung der Ressource (z. B. einem ARN) erstellt wurde

Wenn die Eigenschaft eines Konstrukts ein Konstrukt für eine andere Ressource darstellt, entspricht ihr Typ dem Schnittstellentyp des Konstrukts. Das Amazon ECS-Konstrukt nimmt beispielsweise eine Eigenschaft `cluster` vom Typ `anecs.ICluster`. Ein anderes Beispiel ist das CloudFront Verteilungskonstrukt, das eine Eigenschaft `sourceBucket` (Python:`source_bucket`) vom Typ `annimmts3.IBucket`.

Sie können jedes Ressourcenobjekt des richtigen Typs, das in derselben AWS CDK App definiert ist, direkt übergeben. Das folgende Beispiel definiert einen Amazon ECS-Cluster und verwendet ihn dann, um einen Amazon ECS-Service zu definieren.

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 });
```

Go

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

cluster := ecs.NewCluster(stack, jsii.String("MyCluster"), &ecs.ClusterProps{})
service := ecs.NewEc2Service(stack, jsii.String("MyService"), &ecs.Ec2ServiceProps{
    Cluster: cluster,
})
```

Referenzieren von Ressourcen in einem anderen Stack

Sie können auf Ressourcen in einem anderen Stack verweisen, sofern sie in derselben App definiert sind und sich in derselben AWS Umgebung befinden. Das folgende Muster wird im Allgemeinen verwendet:

- Speichern Sie einen Verweis auf das Konstrukt als Attribut des Stacks, der die Ressource erzeugt. (Um einen Verweis auf den Stack des aktuellen Konstrukts zu erhalten, verwenden Sie `Stack.of(this)`.)
- Übergeben Sie diese Referenz als Parameter oder Eigenschaft an den Konstruktor des Stacks, der die Ressource verbraucht. Der verbrauchende Stapel übergibt ihn dann als Eigenschaft an jedes Konstrukt, das ihn benötigt.

Das folgende Beispiel definiert einen Stapel `stack1`. Dieser Stack definiert einen Amazon S3 S3-Bucket und speichert einen Verweis auf das Bucket-Konstrukt als Attribut des Stacks. Dann definiert die App einen zweiten Stack `stack2`, der bei der Instanziierung einen Bucket akzeptiert. `stack2` könnte zum Beispiel eine AWS Glue Tabelle definieren, die den Bucket für die Datenspeicherung verwendet.

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});
```

Wenn der AWS CDK feststellt, dass sich die Ressource in derselben Umgebung, aber in einem anderen Stapel befindet, synthetisiert er automatisch AWS CloudFormation [Exporte](#) im produzierenden Stapel und ein [Fn:: ImportValue](#) im verbrauchenden Stapel, um diese Informationen von einem Stapel auf den anderen zu übertragen.

Behebung von Deadlocks bei Abhängigkeiten

Wenn Sie auf eine Ressource von einem Stapel in einem anderen Stapel verweisen, entsteht eine Abhängigkeit zwischen den beiden Stacks. Dadurch wird sichergestellt, dass sie in der richtigen Reihenfolge bereitgestellt werden. Nachdem die Stacks bereitgestellt wurden, ist diese Abhängigkeit konkret. Danach kann das Entfernen der Nutzung der gemeinsam genutzten Ressource aus dem verbrauchenden Stack zu einem unerwarteten Bereitstellungsfehler führen. Dies passiert, wenn zwischen den beiden Stacks eine weitere Abhängigkeit besteht, die eine Bereitstellung in derselben Reihenfolge erzwingt. Es kann auch ohne Abhängigkeit geschehen, wenn der produzierende Stack einfach vom CDK Toolkit so ausgewählt wird, dass er zuerst bereitgestellt wird. Der AWS CloudFormation Export wird aus dem produzierenden Stack entfernt, weil er nicht mehr benötigt wird. Die exportierte Ressource wird jedoch weiterhin im verbrauchenden Stack verwendet, da ihr Update noch nicht bereitgestellt wurde. Daher schlägt die Bereitstellung des Producer-Stacks fehl.

Um diesen Deadlock zu überwinden, entfernen Sie die Nutzung der gemeinsam genutzten Ressource aus dem verbrauchenden Stack. (Dadurch wird der automatische Export aus

dem produzierenden Stack entfernt.) Als Nächstes fügen Sie denselben Export manuell zum Produktionsstapel hinzu und verwenden dabei genau dieselbe logische ID wie der automatisch generierte Export. Entfernen Sie die Nutzung der gemeinsam genutzten Ressource im verbrauchenden Stack und stellen Sie beide Stapel bereit. Entfernen Sie dann den manuellen Export (und die gemeinsam genutzte Ressource, falls sie nicht mehr benötigt wird) und stellen Sie beide Stapel erneut bereit. Die [exportValue\(\)](#) Methode des Stacks ist eine bequeme Möglichkeit, den manuellen Export für diesen Zweck zu erstellen. (Sehen Sie sich das Beispiel in der verlinkten Methodenreferenz an.)

Verweisen auf Ressourcen in Ihrem Konto AWS

Angenommen, Sie möchten eine Ressource, die bereits in Ihrem AWS Konto verfügbar ist, in Ihrer AWS CDK App verwenden. Dies kann eine Ressource sein, die über die Konsole, ein AWS SDK, direkt mit AWS CloudFormation oder in einer anderen AWS CDK Anwendung definiert wurde. Sie können den ARN der Ressource (oder ein anderes identifizierendes Attribut oder eine Gruppe von Attributen) in ein Proxy-Objekt umwandeln. Das Proxy-Objekt dient als Referenz auf die Ressource, indem es eine statische Factory-Methode für die Klasse der Ressource aufruft.

Wenn Sie einen solchen Proxy erstellen, wird die externe Ressource nicht Teil Ihrer AWS CDK App. Daher wirken sich Änderungen, die Sie am Proxy in Ihrer AWS CDK App vornehmen, nicht auf die bereitgestellte Ressource aus. Der Proxy kann jedoch an jede AWS CDK Methode übergeben werden, die eine Ressource dieses Typs benötigt.

Das folgende Beispiel zeigt, wie Sie auf einen Bucket verweisen, der auf einem vorhandenen Bucket mit dem ARN `arn:aws:s3:::my-bucket-name` basiert, und auf eine Amazon Virtual Private Cloud, die auf einer vorhandenen VPC mit einer bestimmten ID basiert.

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"
});
```

Go

```
// Define a proxy for a bucket by its name (must be same account)
s3.Bucket_FromBucketName(stack, jsii.String("MyBucket"),
    jsii.String("MyBucketName"))

// Define a proxy for a bucket by its full ARN (can be another account)
s3.Bucket_FromBucketArn(stack, jsii.String("MyBucket"),
    jsii.String("arn:aws:s3:::my-bucket-name"))

// Define a proxy for an existing VPC from its attributes
ec2.Vpc_FromVpcAttributes(stack, jsii.String("MyVpc"), &ec2.VpcAttributes{
    VpcId: jsii.String("vpc-1234567890abcde"),
})
```

Schauen wir uns die Methode genauer an. [Vpc.fromLookup\(\)](#) Da das `ec2.Vpc` Konstrukt komplex ist, gibt es viele Möglichkeiten, die VPC auszuwählen, die mit Ihrer CDK-App verwendet werden soll. Um dieses Problem zu lösen, verfügt das VPC-Konstrukt über eine `fromLookup` statische Methode (Python: `from_lookup`), mit der Sie die gewünschte Amazon-VPC suchen können, indem Sie Ihr AWS Konto bei der Synthese abfragen.

Für die Verwendung muss das `SystemVpc.fromLookup()`, das den Stack synthetisiert, Zugriff auf das Konto haben, dem die Amazon VPC gehört. Das liegt daran, dass das CDK Toolkit das Konto abfragt, um zum Zeitpunkt der Synthese die richtige Amazon VPC zu finden.

Außerdem `Vpc.fromLookup()` funktioniert es nur in Stacks, die mit einem expliziten Konto und einer Region definiert sind (siehe). [the section called "Umgebungen"](#) Wenn AWS CDK versucht wird, eine Amazon-VPC aus einem [umgebungsunabhängigen Stack](#) zu suchen, weiß das CDK Toolkit nicht, welche Umgebung abgefragt werden muss, um die VPC zu finden.

Sie müssen ausreichende `Vpc.fromLookup()` Attribute angeben, um eine VPC in Ihrem AWS Konto eindeutig zu identifizieren. Beispielsweise kann es immer nur eine Standard-VPC geben, sodass es ausreichend ist, die VPC als Standard-VPC anzugeben.

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 });
```

Go

```
ec2.Vpc_FromLookup(this, jsii.String("DefaultVpc"), &ec2.VpcLookupOptions{
  IsDefault: jsii.Bool(true),
})
```

Sie können die `tags` Eigenschaft auch verwenden, um nach VPCs anhand eines Tags abzufragen. Sie können der Amazon VPC bei ihrer Erstellung Tags hinzufügen, indem Sie AWS CloudFormation oder den AWS CDK verwenden. Sie können Tags jederzeit nach der Erstellung bearbeiten, indem

Sie das AWS Management Console AWS CLI, oder ein AWS SDK verwenden. Zusätzlich zu allen Tags, die Sie selbst hinzufügen, fügt das AWS CDK automatisch die folgenden Tags zu allen VPCs hinzu, die es erstellt.

- Name — Der Name der VPC.
- aws-cdk:subnet-name — Der Name des Subnetzes.
- aws-cdk:subnet-type — Der Typ des Subnetzes: Öffentlich, Privat oder Isoliert.

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" });
```

Go

```
ec2.Vpc_FromLookup(this, jsii.String("DefaultVpc"), &ec2.VpcLookupOptions{
```

```
Tags: &map[string]*string{"aws-cdk:subnet-type": jsii.String("Public")},
})
```

Die Ergebnisse von `Vpc.fromLookup()` werden in der Projektdatei zwischengespeichert. `cdk.context.json` (Siehe [the section called "Kontext"](#)). Übergeben Sie diese Datei der Versionskontrolle, damit Ihre App weiterhin auf dieselbe Amazon VPC verweist. Dies funktioniert auch dann, wenn Sie später die Attribute Ihrer VPCs so ändern, dass eine andere VPC ausgewählt wird. Dies ist besonders wichtig, wenn Sie den Stack in einer Umgebung bereitstellen, die keinen Zugriff auf das AWS Konto hat, das die VPC definiert, z. B. [CDK](#) Pipelines.

Sie können eine externe Ressource zwar überall verwenden, wo Sie eine ähnliche, in Ihrer AWS CDK App definierte Ressource verwenden würden, aber Sie können sie nicht ändern. Zum Beispiel bewirkt der Aufruf von `addToResourcePolicy` (Python:`add_to_resource_policy`) auf einem externen `s3.Bucket` Gerät nichts.

Physische Namen der Ressourcen

Die logischen Namen der Ressourcen in AWS CloudFormation unterscheiden sich von den Namen der Ressourcen, die AWS Management Console nach ihrer Bereitstellung von angezeigt werden AWS CloudFormation. Er AWS CDK nennt diese endgültigen Namen physische Namen.

AWS CloudFormation Könnte beispielsweise den Amazon S3 S3-Bucket mit der logischen ID `Stack2MyBucket4DD88B4F` aus dem vorherigen Beispiel mit dem physischen Namen `erstellenstack2mybucket4dd88b4f-iuv1rbv9z3to`.

Sie können einen physikalischen Namen angeben, wenn Sie Konstrukte erstellen, die Ressourcen darstellen, indem Sie die Eigenschaft `<resourceType>Name` verwenden. Im folgenden Beispiel wird ein Amazon S3 S3-Bucket mit dem physischen Namen `erstelltmy-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" });
```

Go

```
bucket := s3.NewBucket(this, jsii.String("MyBucket"), &s3.BucketProps{  
    BucketName: jsii.String("my-bucket-name"),  
})
```

Die Zuweisung physischer Namen zu Ressourcen hat einige Nachteile in AWS CloudFormation. Am wichtigsten ist jedoch, dass alle Änderungen an bereitgestellten Ressourcen, die einen Austausch von Ressourcen erfordern, wie z. B. Änderungen an den Eigenschaften einer Ressource, die nach der Erstellung unveränderlich sind, fehlschlagen, wenn einer Ressource ein physischer Name zugewiesen wurde. Wenn Sie in diesem Zustand landen, besteht die einzige Lösung darin, den AWS CloudFormation Stack zu löschen und die AWS CDK App dann erneut bereitzustellen. Einzelheiten finden Sie in der [AWS CloudFormation Dokumentation](#).

In einigen Fällen, z. B. beim Erstellen einer AWS CDK App mit umgebungsübergreifenden Verweisen, sind physische Namen erforderlich, AWS CDK damit sie ordnungsgemäß funktioniert. In diesen Fällen können Sie, wenn Sie sich nicht selbst die Mühe machen möchten, einen physischen Namen zu finden, den AWS CDK Namen für Sie selbst festlegen lassen. Verwenden Sie dazu den speziellen Wert `PhysicalName.GENERATE_IF_NEEDED` wie folgt.

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 });
```

Go

```
bucket := s3.NewBucket(this, jsii.String("MyBucket"), &s3.BucketProps{  
  BucketName: awscdk.PhysicalName_GENERATE_IF_NEEDED(),  
})
```

Übergeben von eindeutigen Ressourcen-Identifikatoren

Wann immer möglich, sollten Sie Ressourcen als Referenz übergeben, wie im vorherigen Abschnitt beschrieben. Es gibt jedoch Fälle, in denen Sie keine andere Wahl haben, als mit einem ihrer Attribute auf eine Ressource zu verweisen. Zu den beispielhaften Anwendungsfällen gehören die folgenden:

- Wenn Sie AWS CloudFormation Ressourcen auf niedriger Ebene verwenden.

- Wenn Sie Ressourcen für die Laufzeitkomponenten einer AWS CDK Anwendung verfügbar machen müssen, z. B. wenn Sie über Umgebungsvariablen auf Lambda-Funktionen verweisen.

Diese Bezeichner sind als Attribute für die Ressourcen verfügbar, wie z. B. die folgenden.

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

Die AWS CDK Java-Bindung verwendet Getter-Methoden für Attribute.

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

C#

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

Go

```
bucket.BucketName()
```

```
fn.FunctionArn()
```

Das folgende Beispiel zeigt, wie ein generierter Bucket-Name an eine AWS Lambda Funktion übergeben wird.

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
    }
});
```

Go

```
bucket := s3.NewBucket(this, jsii.String("Bucket"), &s3.BucketProps{})
lambda.NewFunction(this, jsii.String("MyLambda"), &lambda.FunctionProps{
    Environment: &map[string]*string{"BUCKET_NAME": bucket.BucketName()},
})
```

Erteilen von Berechtigungen zwischen Ressourcen

Konstrukte auf höherer Ebene ermöglichen die Erzielung von Berechtigungen mit den geringsten Rechten, indem sie einfache, absichtsbasierte APIs anbieten, mit denen Berechtigungsanforderungen ausgedrückt werden können. Viele L2-Konstrukte bieten beispielsweise Erteilungsmethoden, mit denen Sie einer Entität (z. B. einer IAM-Rolle oder einem IAM-Benutzer) die Erlaubnis erteilen können, mit der Ressource zu arbeiten, ohne IAM-Berechtigungsanweisungen manuell erstellen zu müssen.

Im folgenden Beispiel werden die Berechtigungen erstellt, die es der Ausführungsrolle einer Lambda-Funktion ermöglichen, Objekte in einen bestimmten Amazon S3 S3-Bucket zu lesen und zu schreiben. Wenn der Amazon S3 S3-Bucket mit einem AWS KMS Schlüssel verschlüsselt ist, gewährt diese Methode auch der Ausführungsrolle der Lambda-Funktion die Erlaubnis, mit dem Schlüssel zu entschlüsseln.

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)  
{  
    // ...  
}
```

Go

```
if *bucket.GrantReadWrite(function, nil).Success() {  
    // ...  
}
```

Die Grant-Methoden geben ein `iam.Grant` Objekt zurück. Verwenden Sie das `success` Attribut des Grant Objekts, um festzustellen, ob der Zuschuss tatsächlich gewährt wurde (z. B. wurde er möglicherweise nicht auf [externe Ressourcen](#) angewendet). Sie können auch die Methode `assertSuccess` (Python:`assert_success`) des Grant Objekts verwenden, um zu erzwingen, dass der Zuschuss erfolgreich beantragt wurde.

Wenn eine bestimmte Grant-Methode für den jeweiligen Anwendungsfall nicht verfügbar ist, können Sie eine generische Grant-Methode verwenden, um einen neuen Zuschuss mit einer bestimmten Liste von Aktionen zu definieren.

Das folgende Beispiel zeigt, wie einer Lambda-Funktion Zugriff auf die Amazon DynamoDB `CreateBackup` DynamoDB-Aktion gewährt wird.

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");
```

Go

```
table := dynamodb.NewTable(this, jsii.String("MyTable"), &dynamodb.TableProps{  
table.Grant(function, jsii.String("dynamodb:CreateBackup"))
```

Für viele Ressourcen, wie z. B. Lambda-Funktionen, muss bei der Ausführung von Code eine Rolle übernommen werden. Mit einer Konfigurationseigenschaft können Sie eine `iam.IRole` angeben. Wenn keine Rolle angegeben ist, erstellt die Funktion automatisch eine Rolle speziell für diesen Zweck. Anschließend können Sie `Grant`-Methoden für die Ressourcen verwenden, um der Rolle Anweisungen hinzuzufügen.

Die `Grant`-Methoden werden mithilfe von APIs auf niedrigerer Ebene für den Umgang mit IAM-Richtlinien erstellt. Richtlinien werden als Objekte modelliert. [PolicyDocument](#) Fügen Sie mithilfe der Methode (Python:) Anweisungen direkt zu Rollen (oder der angehängten

Rolle eines Konstrukts `add_to_role_policy`) oder mit der `addToRolePolicy` Methode (Python:) zur Bucket Richtlinie einer Ressource `addToResourcePolicy` (z. B. einer Richtlinie `add_to_resource_policy`) hinzu.

Metriken und Alarme für Ressourcen

Viele Ressourcen geben CloudWatch Metriken aus, mit denen Überwachungs-Dashboards und Alarme eingerichtet werden können. Konstrukte auf höherer Ebene verfügen über metrische Methoden, mit denen Sie auf die Metriken zugreifen können, ohne nach dem richtigen Namen suchen zu müssen.

Das folgende Beispiel zeigt, wie ein Alarm definiert wird, wenn `ApproximateNumberOfMessagesNotVisible` der Wert einer Amazon SQS SQS-Warteschlange 100 überschreitet.

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,
    // ..
});

```

Go

```

import (
    "github.com/aws/aws-cdk-go/awscdk/v2"
    "github.com/aws/jsii-runtime-go"
    cw "github.com/aws/aws-cdk-go/awscdk/v2/awsccloudwatch"
    sqs "github.com/aws/aws-cdk-go/awscdk/v2/awssqs"
)

queue := sqs.NewQueue(this, jsii.String("MyQueue"), &sqs.QueueProps{})
metric := queue.MetricApproximateNumberOfMessagesNotVisible(&cw.MetricOptions{
    Label: jsii.String("Messages Visible (Approx)"),
    Period: awscdk.Duration_Minutes(jsii.Number(5)),
})

```

```
metric.CreateAlarm(this, jsii.String("TooManyMessagesAlarm"),
  &cw.CreateAlarmOptions{
    ComparisonOperator: cw.ComparisonOperator_GREATER_THAN_THRESHOLD,
    Threshold: jsii.Number(100),
  })
```

Wenn es für eine bestimmte Metrik keine Methode gibt, können Sie die allgemeine Metrikmethode verwenden, um den Metriknamen manuell anzugeben.

Metriken können auch zu CloudWatch Dashboards hinzugefügt werden. Siehe [CloudWatch](#).

Netzwerkdatenverkehr

In vielen Fällen müssen Sie Berechtigungen in einem Netzwerk aktivieren, damit eine Anwendung funktioniert, z. B. wenn die Recheninfrastruktur auf die Persistenzschicht zugreifen muss.

Ressourcen, die Verbindungen herstellen oder darauf warten, stellen Methoden zur Verfügung, die den Datenfluss ermöglichen, einschließlich der Festlegung von Sicherheitsgruppenregeln oder Netzwerk-ACLs.

[IConnectable-Ressourcen](#) verfügen über eine `connections` Eigenschaft, die das Gateway zur Konfiguration der Regeln für den Netzwerkverkehr darstellt.

Mithilfe `allow` von Methoden ermöglichen Sie den Datenfluss auf einem bestimmten Netzwerkpfad. Das folgende Beispiel aktiviert HTTPS-Verbindungen zum Internet und eingehende Verbindungen von der Amazon EC2 Auto Scaling Scaling-Gruppe `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());
```

Go

```
import (
    "github.com/aws/aws-cdk-go/awscdk/v2"
    "github.com/aws/jsii-runtime-go"
    autoscaling "github.com/aws/aws-cdk-go/awscdk/v2/awsautoscaling"
    ec2 "github.com/aws/aws-cdk-go/awscdk/v2/awsec2"
)

fleet1 := autoscaling.NewAutoScalingGroup(this, jsii.String("MyFleet1"),
    &autoscaling.AutoScalingGroupProps{})
fleet1.Connections().AllowTo(ec2.Peer_AnyIpv4(), ec2.NewPort(&ec2.PortProps{ FromPort:
    jsii.Number(443), ToPort: jsii.Number(443) }), jsii.String("secure web"))

fleet2 := autoscaling.NewAutoScalingGroup(this, jsii.String("MyFleet2"),
    &autoscaling.AutoScalingGroupProps{})
fleet1.Connections().AllowFrom(fleet2, ec2.Port_AllTraffic(), jsii.String("all
    traffic"))
```

Bestimmten Ressourcen sind Standardports zugeordnet. Beispiele hierfür sind der Listener eines Load Balancers auf dem öffentlichen Port und die Ports, auf denen die Datenbank-Engine Verbindungen für Instances einer Amazon RDS-Datenbank akzeptiert. In solchen Fällen können Sie eine strenge Netzwerkkontrolle erzwingen, ohne den Port manuell angeben zu müssen.

Verwenden Sie dazu die `allowToDefaultPort` Methoden `allowDefaultPortFrom` und (Python:`allow_default_port_from`,`allow_to_default_port`).

Das folgende Beispiel zeigt, wie Verbindungen von einer beliebigen IPV4-Adresse und eine Verbindung von einer Auto Scaling Scaling-Gruppe aus für den Zugriff auf eine Datenbank aktiviert werden.

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");
```

Go

```
listener.Connections().AllowDefaultPortFromAnyIpv4(jsii.String("Allow public  
Access"))
```

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

Behandlung von Ereignissen

Einige Ressourcen können als Ereignisquellen dienen. Verwenden Sie die `addEventNotification` Methode (Python: `add_event_notification`), um ein Ereignisziel für einen bestimmten Ereignistyp zu registrieren, der von der Ressource ausgegeben wird. Darüber hinaus bieten `addXxxNotification` Methoden eine einfache Möglichkeit, einen Handler für gängige Ereignistypen zu registrieren.

Das folgende Beispiel zeigt, wie eine Lambda-Funktion ausgelöst wird, wenn ein Objekt zu einem Amazon S3 S3-Bucket hinzugefügt wird.

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_not

handler = lambda_.Function(self, "Handler", ...)
bucket = s3.Bucket(self, "Bucket")
bucket.add_object_created_notification(s3_not.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));
```

Go

```
import (
    "github.com/aws/aws-cdk-go/awscdk/v2"
    "github.com/aws/jsii-runtime-go"
    s3 "github.com/aws/aws-cdk-go/awscdk/v2/awss3"
    s3nots "github.com/aws/aws-cdk-go/awscdk/v2/awss3notifications"
)

handler := lambda.NewFunction(this, jsii.String("MyFunction"),
    &lambda.FunctionProps{})
bucket := s3.NewBucket(this, jsii.String("Bucket"), &s3.BucketProps{})
bucket.AddObjectCreatedNotification(s3nots.NewLambdaDestination(handler), nil)
```

Richtlinien zur Entfernung

Für Ressourcen, die persistente Daten verwalten, wie Datenbanken, Amazon S3 S3-Buckets und Amazon ECR-Registries, gilt eine Entfernungsrichtlinie. Die Entfernungsrichtlinie gibt an, ob persistente Objekte gelöscht werden sollen, wenn der AWS CDK Stapel, der sie enthält, zerstört wird.

Die Werte, die die Entfernungsrichtlinie angeben, sind über die `RemovalPolicy` Aufzählung im `AWS CDK core` Modul verfügbar.

Note

Neben Ressourcen, die Daten dauerhaft speichern, können auch Ressourcen eine `removalPolicy` haben, die für einen anderen Zweck verwendet wird. Beispielsweise verwendet eine Lambda-Funktionsversion ein `removalPolicy` Attribut, um zu bestimmen, ob eine bestimmte Version beibehalten wird, wenn eine neue Version bereitgestellt wird. Diese haben im Vergleich zu den Entfernungsrichtlinien für einen Amazon S3 S3-Bucket oder eine DynamoDB-Tabelle unterschiedliche Bedeutungen und Standardeinstellungen.

Wert	Sinn
<code>RemovalPolicy.RETAIN</code>	Behält den Inhalt der Ressource bei, wenn der Stapel zerstört wird (Standard). Die Ressource ist aus dem Stapel verwaist und muss manuell gelöscht werden. Wenn Sie versuchen, den Stack erneut bereitzustellen, während die Ressource noch vorhanden ist, erhalten Sie aufgrund eines Namenskonflikts eine Fehlermeldung.
<code>RemovalPolicy.DESTROY</code>	Die Ressource wird zusammen mit dem Stapel zerstört.

AWS CloudFormation entfernt keine Amazon S3 S3-Buckets, die Dateien enthalten, auch wenn ihre Entfernungsrichtlinie auf `DESTROY` eingestellt ist. Der Versuch, dies zu tun, ist ein AWS CloudFormation Fehler. Damit alle Dateien aus dem Bucket AWS CDK gelöscht werden, bevor er zerstört wird, setzen Sie die `autoDeleteObjects` Eigenschaft des Buckets auf `true`.

Im Folgenden finden Sie ein Beispiel für die Erstellung eines Amazon S3 S3-Buckets mit `RemovalPolicy` der Einstellung of `DESTROY` und der `autoDeleteObjects` Einstellung auf `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
    });
}
```

Go

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

```
"github.com/aws/jsii-runtime-go"  
s3 "github.com/aws/aws-cdk-go/awscdk/v2/awss3"  
)  
  
s3.NewBucket(this, jsii.String("Bucket"), &s3.BucketProps{  
  RemovalPolicy: awscdk.RemovalPolicy_DESTROY,  
  AutoDeleteObjects: jsii.Bool(true),  
})
```

Mithilfe der `applyRemovalPolicy()` Methode können Sie eine Entfernungsrichtlinie auch direkt auf die zugrunde liegende AWS CloudFormation Ressource anwenden. Diese Methode ist für einige statusbehaftete Ressourcen verfügbar, die keine `removalPolicy` Eigenschaft in den Requisiten ihrer L2-Ressource haben. Beispiele sind unter anderem:

- AWS CloudFormation stapelt
- Amazon-Cognito-Benutzerpools
- Amazon DocumentDB DocumentDB-Datenbank-Instances
- Amazon EC2 EC2-Volumen
- Amazon OpenSearch Service-Domänen
- Amazon FSx-Dateisysteme
- Amazon SQS-Warteschlangen

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

Das „AWS CDK s“ RemovalPolicy heißt übersetzt „s“. AWS CloudFormationDeletionPolicy Die Standardeinstellung besteht jedoch AWS CDK darin, die Daten beizubehalten, was das Gegenteil der AWS CloudFormation Standardeinstellung ist.

IDs

Beim Erstellen von AWS Cloud Development Kit (AWS CDK) Apps verwenden Sie viele Arten von Kennungen und Namen. Um AWS CDK effektiv zu nutzen und Fehler zu vermeiden, ist es wichtig, die Arten von Bezeichnern zu verstehen.

IDs müssen innerhalb des Bereichs, in dem sie erstellt werden, eindeutig sein. Sie müssen in Ihrer AWS CDK Anwendung nicht global eindeutig sein.

Wenn Sie versuchen, eine Kennung mit demselben Wert innerhalb desselben Bereichs zu erstellen, AWS CDK löst die eine Ausnahme aus.

Themen

- [Konstrukt-IDs](#)
- [Pfade](#)
- [Eindeutige IDs](#)
- [Logische IDs](#)

Konstrukt-IDs

Die gängigste Kennung, `id`, ist die Kennung, die beim Instanzieren eines Konstruktobjekts als zweites Argument übergeben wird. Diese Kennung muss, wie alle Kennungen, nur innerhalb des Bereichs, in dem sie erstellt wird, eindeutig sein. Dies ist das erste Argument beim Instanzieren eines Konstruktobjekts.

Note

Die `id` eines Stacks ist auch die Kennung, mit der Sie in der darauf verweisen [the section called “AWS CDK Toolkit”](#).

Schauen wir uns ein Beispiel an, in dem wir zwei Konstrukte mit der ID `MyBucket` in unserer App haben. Die erste ist im Bereich des Stacks mit der Kennung `definiertStack1`. Die zweite ist im Bereich eines Stacks mit der Kennung `definiertStack2`. Da sie in verschiedenen Bereichen definiert sind, verursacht dies keinen Konflikt und sie können problemlos in derselben App vorhanden sein.

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");
    }
}
```

```
}  
}
```

Pfade

Die Konstrukte in einer AWS CDK Anwendung bilden eine Hierarchie, die sich in der `-App`-Klasse befindet. Wir bezeichnen die Sammlung von IDs aus einem bestimmten Konstrukt, seinem übergeordneten Konstrukt, seinem übergeordneten Konstrukt usw. zum Stamm des Konstruktbaums als Pfad .

zeigt in der AWS CDK Regel Pfade in Ihren Vorlagen als Zeichenfolge an. Die IDs der Ebenen werden durch Schrägstriche getrennt, beginnend mit dem Knoten unmittelbar unter der Stamm-AppInstance, bei der es sich normalerweise um einen Stack handelt. Die Pfade der beiden Amazon S3-Bucket-Ressourcen im vorherigen Codebeispiel sind beispielsweise `Stack1/MyBucket` und `Stack2/MyBucket`.

Sie können auf den Pfad jedes Konstrukts programmgesteuert zugreifen, wie im folgenden Beispiel gezeigt. Dadurch wird der Pfad von `myConstruct` (oder `my_construct`, wie Python-Entwickler ihn schreiben würden) abgerufen. Da IDs innerhalb des erstellten Bereichs eindeutig sein müssen, sind ihre Pfade innerhalb einer AWS CDK Anwendung immer eindeutig.

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;
```

Eindeutige IDs

AWS CloudFormation erfordert, dass alle logischen IDs in einer Vorlage eindeutig sind. Aus diesem Grund AWS CDK muss für jedes Konstrukt in einer Anwendung eine eindeutige Kennung generieren können. Ressourcen haben global eindeutige Pfade (die Namen aller Bereiche vom Stack zu einer bestimmten Ressource). Daher AWS CDK generiert die erforderlichen eindeutigen Kennungen, indem die Elemente des Pfads verkettet und ein achtstelliger Hash hinzugefügt werden. (Der Hash ist erforderlich, um unterschiedliche Pfade wie A/B/C und zu unterscheiden A/BC, die zu derselben AWS CloudFormation Kennung führen würden. AWS CloudFormation -Kennungen sind alphanumerisch und dürfen keine Schrägstriche oder andere Trennzeichen enthalten.) Der AWS CDK ruft diese Zeichenfolge die eindeutige ID des Konstrukts auf.

Im Allgemeinen sollte Ihre AWS CDK App nichts über eindeutige IDs wissen müssen. Sie können jedoch programmgesteuert auf die eindeutige ID eines Konstrukts zugreifen, wie im folgenden Beispiel gezeigt.

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);
```

Die Adresse ist eine andere Art von eindeutiger Kennung, die CDK-Ressourcen eindeutig unterscheidet. Abgeleitet vom SHA-1-Hash des Pfads ist er nicht für Menschen lesbar. Die konstante, relativ kurze Länge (also immer 42 hexadezimale Zeichen) macht sie jedoch in Situationen nützlich, in denen die „traditionelle“ eindeutige ID möglicherweise zu lang ist. Einige Konstrukte verwenden möglicherweise die Adresse in der synthetisierten AWS CloudFormation Vorlage anstelle der eindeutigen ID. Auch hier sollte Ihre App im Allgemeinen nichts über die Adressen ihrer Konstrukte wissen müssen, aber Sie können die Adresse eines Konstrukts wie folgt abrufen.

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;
```

Logische IDs

Eindeutige IDs dienen als logische Kennungen (oder logische Namen) von Ressourcen in den generierten AWS CloudFormation Vorlagen für Konstrukte, die AWS Ressourcen darstellen.

Beispielsweise Stack2 führt der Amazon S3-Bucket im vorherigen Beispiel, der in erstellt wurde, zu einer `-AWS::S3::Bucket`Ressource. Die logische ID der Ressource befindet sich Stack2MyBucket4DD88B4F in der resultierenden AWS CloudFormation Vorlage. (Details dazu, wie diese Kennung generiert wird, finden Sie unter [the section called “Eindeutige IDs”](#).)

Stabilität der logischen ID

Vermeiden Sie es, die logische ID einer Ressource zu ändern, nachdem sie erstellt wurde. AWS CloudFormation identifiziert Ressourcen anhand ihrer logischen ID. Wenn Sie also die logische ID einer Ressource ändern, AWS CloudFormation erstellt eine neue Ressource mit der neuen logischen ID und löscht dann die vorhandene. Je nach Art der Ressource kann dies zu Serviceunterbrechungen, Datenverlust oder beidem führen.

Token

Token stellen Werte dar, die nur zu einem späteren Zeitpunkt im [App-Lebenszyklus](#) aufgelöst werden können. Beispielsweise wird der Name eines Amazon Simple Storage Service (Amazon S3)-Buckets, den Sie in Ihrer CDK-App definieren, nur zugewiesen, wenn die AWS CloudFormation Vorlage synthetisiert wird. Wenn Sie das `bucket.bucketName` Attribut drucken, bei dem es sich um eine Zeichenfolge handelt, wird Folgendes angezeigt:

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

Auf diese Weise AWS CDK kodiert die ein Token, dessen Wert zum Zeitpunkt der Konstruktion noch nicht bekannt ist, aber später verfügbar wird. Die AWS CDK ruft diese Platzhalter-Token auf. In diesem Fall handelt es sich um ein Token, das als Zeichenfolge codiert ist.

Sie können diese Zeichenfolge so übergeben, als wäre sie der Name des Buckets. Im folgenden Beispiel wird der Bucket-Name als Umgebungsvariable für eine - AWS Lambda Funktion angegeben.

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
    }
});
```

Wenn die AWS CloudFormation Vorlage schließlich synthetisiert wird, wird das Token als AWS CloudFormation intrinsische gerendert{ "Ref": "MyBucket" }. Zum Zeitpunkt der Bereitstellung AWS CloudFormation ersetzt dieses intrinsische durch den tatsächlichen Namen des erstellten Buckets.

Themen

- [Token und Token-Kodierungen](#)
- [Zeichenfolgenkodierte Token](#)
- [Listencodierte Token](#)
- [Nummernkodierte Token](#)
- [Lazy Values](#)
- [Konvertieren in JSON](#)

Token und Token-Kodierungen

Token sind Objekte, die die [IResolvable](#)-Schnittstelle implementieren, die eine einzige `resolve` Methode enthält. Die AWS CDK ruft diese Methode während der Synthetisierung auf, um den endgültigen Wert für die AWS CloudFormation Vorlage zu erzeugen. Token nehmen am Synthetisierungsprozess teil, um beliebige Werte beliebiger Art zu erzeugen.

Note

Sie arbeiten selten direkt mit der `IResolvable`-Schnittstelle. Sie werden wahrscheinlich nur zeichenfolgenkodierte Versionen von Token sehen.

Andere Funktionen akzeptieren in der Regel nur Argumente grundlegender Typen wie `string` oder `number`. Um Token in diesen Fällen zu verwenden, können Sie sie in einen von drei Typen codieren, indem Sie statische Methoden für die Klasse [cdk.Token](#) verwenden.

- [Token.asString](#) , um eine Zeichenfolgenkodierung zu generieren (oder `.toString()` für das Token-Objekt aufzurufen)
- [Token.asList](#) zum Generieren einer Listenkodierung
- [Token.asNumber](#) , um eine numerische Kodierung zu generieren

Diese verwenden einen beliebigen Wert, der ein sein kann `IResolvable`, und codieren sie in einen primitiven Wert des angegebenen Typs.

Important

Da einer der vorherigen Typen möglicherweise ein codiertes Token sein kann, sollten Sie beim Parsen oder Versuch, ihren Inhalt zu lesen, vorsichtig sein. Wenn Sie beispielsweise versuchen, eine Zeichenfolge zu analysieren, um einen Wert daraus zu extrahieren, und es sich bei der Zeichenfolge um ein codiertes Token handelt, schlägt die Analyse fehl. Wenn Sie versuchen, die Länge eines Arrays abzufragen oder mathematische Operationen mit einer Zahl durchzuführen, müssen Sie zunächst überprüfen, ob es sich nicht um codierte Token handelt.

Um zu überprüfen, ob ein Wert ein nicht aufgelöstes Token enthält, rufen Sie die Methode `Token.isUnresolved` (Python: `is_unresolved`) auf.

Im folgenden Beispiel wird überprüft, ob ein Zeichenfolgenwert, der ein Token sein könnte, nicht mehr als 10 Zeichen lang ist.

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");
```

Wenn Name ein Token ist, wird keine Validierung durchgeführt, und in einer späteren Phase des Lebenszyklus kann ein Fehler auftreten, z. B. während der Bereitstellung.

Note

Sie können Token-Kodierungen verwenden, um das Typsystem zu maskieren. Sie könnten beispielsweise ein Token, das einen Zahlenwert zur Synthetisierungszeit erzeugt, mit einer Zeichenfolge kodieren. Wenn Sie diese Funktionen verwenden, liegt es in Ihrer Verantwortung sicherzustellen, dass Ihre Vorlage nach der Generierung in einen nutzbaren Zustand aufgelöst wird.

Zeichenfolgenkodierte Token

Zeichenfolgenkodierte Token sehen wie folgt aus.

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

Sie können wie reguläre Zeichenfolgen übergeben und verkettet werden, wie im folgenden Beispiel gezeigt.

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";
```

Sie können auch die Zeichenfolgeninterpolation verwenden, wenn Ihre Sprache dies unterstützt, wie im folgenden Beispiel gezeigt.

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";
```

Vermeiden Sie es, die Zeichenfolge auf andere Weise zu bearbeiten. Wenn Sie beispielsweise eine Teilzeichenfolge einer Zeichenfolge nehmen, wird das Zeichenfolgen-Token wahrscheinlich unterbrochen.

Listencodierte Token

Listenkodierte Token sehen wie folgt aus:

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

Die einzige sichere Möglichkeit für diese Listen besteht darin, sie direkt an andere Konstrukte zu übergeben. Token in Form von Zeichenfolgenlisten können nicht verkettet werden und es kann auch kein Element aus dem Token übernommen werden. Die einzige sichere Möglichkeit, sie zu bearbeiten, ist die Verwendung AWS CloudFormation intrinsischer Funktionen wie [Fn.select](#).

Nummernkodierte Token

Zahlenkodierte Token sind eine Reihe von sehr negativen Gleitkommazahlen, die wie folgt aussehen.

```
-1.8881545897087626e+289
```

Wie bei Listen-Token können Sie den Zahlenwert nicht ändern, da dies wahrscheinlich das Zahlen-Token beschädigen wird. Die einzige zulässige Operation besteht darin, den Wert an ein anderes Konstrukt zu übergeben.

Lazy Values

Zusätzlich zur Darstellung von Bereitstellungszeitwerten, wie z. B. AWS CloudFormation [Parametern](#), werden Token häufig auch verwendet, um Lazy-Werte für die Synthetisierungszeit darzustellen. Dies sind Werte, für die der endgültige Wert bestimmt wird, bevor die Synthetisierung abgeschlossen ist, aber nicht an dem Punkt, an dem der Wert erstellt wird. Verwenden Sie Token, um eine Literalzeichenfolge oder einen Zahlenwert an ein anderes Konstrukt zu übergeben, während der tatsächliche Wert zur Generierungszeit von einer Berechnung abhängen kann, die noch nicht durchgeführt wurde.

Sie können Token, die synth-time Lazy-Werte darstellen, mit statischen Methoden in der Lazy Klasse erstellen, z. B. [Lazy.string](#) und [Lazy.number](#). Diese Methoden akzeptieren ein Objekt, dessen `produce` Eigenschaft eine Funktion ist, die ein Kontextargument akzeptiert und beim Aufruf den endgültigen Wert zurückgibt.

Im folgenden Beispiel wird eine Auto Scaling-Gruppe erstellt, deren Kapazität nach ihrer Erstellung bestimmt wird.

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;
```

Konvertieren in JSON

Manchmal möchten Sie eine JSON-Zeichenfolge beliebiger Daten erstellen und wissen möglicherweise nicht, ob die Daten Token enthalten. Verwenden Sie den Methoden-[StacktoJsonString](#), um eine Datenstruktur unabhängig davon, ob sie Token enthält, ordnungsgemäß mit JSON zu codieren, wie im folgenden Beispiel gezeigt.

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 stringValue = stack.ToJsonString(new Dictionary<string, string>
{
    ["value"] = bucket.BucketName
});
```

Parameter

Parameter sind benutzerdefinierte Werte, die bei der Bereitstellung bereitgestellt werden. [Parameter](#) sind ein Feature von AWS CloudFormation. Da es AWS CloudFormation Vorlagen AWS Cloud Development Kit (AWS CDK) synthetisiert, bietet es auch Unterstützung für Bereitstellungszeitparameter.

Themen

- [Über Parameter](#)
- [Parameter definieren](#)
- [Verwenden von Parametern](#)
- [Bereitstellung mit Parametern](#)

Über Parameter

Mithilfe von können Sie Parameter definieren, die dann in den Eigenschaften der von Ihnen erstellten Konstrukte verwendet werden können. AWS CDK Sie können auch Stacks bereitstellen, die Parameter enthalten.

Wenn Sie die AWS CloudFormation Vorlage mit dem AWS CDK Toolkit bereitstellen, geben Sie die Parameterwerte in der Befehlszeile an. Wenn Sie die Vorlage über die AWS CloudFormation Konsole bereitstellen, werden Sie zur Eingabe der Parameterwerte aufgefordert.

Im Allgemeinen empfehlen wir, keine AWS CloudFormation Parameter mit dem zu verwenden AWS CDK. Die üblichen Methoden, Werte an AWS CDK Apps zu übergeben, sind [Kontextwerte](#) und Umgebungsvariablen. Da sie zum Zeitpunkt der Synthese nicht verfügbar sind, können Parameterwerte nicht einfach für die Flusskontrolle und andere Zwecke in Ihrer CDK-App verwendet werden.

Note

Um die Ablaufsteuerung mit Parametern durchzuführen, können Sie [CfnCondition](#)-Konstrukte verwenden, obwohl dies im Vergleich zu systemeigenen Anweisungen umständlich ist. `if`

Bei der Verwendung von Parametern müssen Sie darauf achten, wie sich der Code, den Sie schreiben, bei der Bereitstellung und auch bei der Synthese verhält. Dies macht es schwieriger, Ihre AWS CDK Anwendung zu verstehen und zu begründen, was in vielen Fällen kaum von Nutzen ist.

Im Allgemeinen ist es besser, wenn Ihre CDK-App die erforderlichen Informationen auf klar definierte Weise akzeptiert und sie direkt zur Deklaration von Konstrukten in Ihrer CDK-App verwendet. Eine ideale AWS CDK— generierte AWS CloudFormation Vorlage ist konkret, sodass zum Zeitpunkt der Bereitstellung keine Werte mehr spezifiziert werden müssen.

Es gibt jedoch Anwendungsfälle, für die AWS CloudFormation Parameter eindeutig geeignet sind. Wenn Sie beispielsweise über separate Teams verfügen, die die Infrastruktur definieren und bereitstellen, können Sie Parameter verwenden, um die generierten Vorlagen breiter nutzbar zu machen. Da der AWS CloudFormation Parameter AWS CDK unterstützt, können Sie den auch AWS CDK mit AWS Diensten verwenden, die AWS CloudFormation Vorlagen verwenden (z. B. Service Catalog). Diese AWS Dienste verwenden Parameter, um die Vorlage zu konfigurieren, die bereitgestellt wird.

Parameter definieren

Verwenden Sie die [CfnParameter](#)-Klasse, um einen Parameter zu definieren. Für die meisten Parameter sollten Sie mindestens einen Typ und eine Beschreibung angeben, obwohl beide technisch gesehen optional sind. Die Beschreibung wird angezeigt, wenn der Benutzer aufgefordert wird, den Wert des Parameters in der AWS CloudFormation Konsole einzugeben. Weitere Informationen zu den verfügbaren Typen finden Sie unter [Typen](#).

Note

Sie können Parameter in jedem Bereich definieren. Wir empfehlen jedoch, Parameter auf Stack-Ebene zu definieren, damit sich ihre logische ID nicht ändert, wenn Sie Ihren Code umgestalten.

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"
});
```

Verwenden von Parametern

Eine `CfnParameter` Instanz stellt ihren Wert Ihrer AWS CDK App über ein [Token](#) zur Verfügung. Wie alle Token wird das Token des Parameters zur Synthesezeit aufgelöst. Es wird jedoch in einen Verweis auf den in der AWS CloudFormation Vorlage definierten Parameter (der zum Zeitpunkt der Bereitstellung aufgelöst wird) und nicht auf einen konkreten Wert aufgelöst.

Sie können das Token als Instanz der Token Klasse oder als Zeichenfolge, Zeichenkettenliste oder numerische Kodierung abrufen. Ihre Wahl hängt von der Art des Werts ab, der für die Klasse oder Methode erforderlich ist, mit der Sie den Parameter verwenden möchten.

TypeScript

Eigenschaft	Art des Werts
<code>value</code>	TokenKlasseninstanz
<code>valueAsList</code>	Das als Stringliste dargestellte Token
<code>valueAsNumber</code>	Das als Zahl dargestellte Token
<code>valueAsString</code>	Das als Zeichenfolge dargestellte Token

JavaScript

Eigenschaft	Art von Wert
<code>value</code>	TokenKlasseninstanz
<code>valueAsList</code>	Das als Stringliste dargestellte Token
<code>valueAsNumber</code>	Das als Zahl dargestellte Token
<code>valueAsString</code>	Das als Zeichenfolge dargestellte Token

Python

Eigenschaft	Art von Wert
<code>value</code>	TokenKlasseninstanz
<code>value_as_list</code>	Das als Stringliste dargestellte Token
<code>value_as_number</code>	Das als Zahl dargestellte Token
<code>value_as_string</code>	Das als Zeichenfolge dargestellte Token

Java

Eigenschaft	Art von Wert
<code>getValue()</code>	TokenKlasseninstanz
<code>getValueAsList()</code>	Das als Stringliste dargestellte Token
<code>getValueAsNumber()</code>	Das als Zahl dargestellte Token
<code>getValueAsString()</code>	Das als Zeichenfolge dargestellte Token

C#

Eigenschaft	Art von Wert
<code>Value</code>	TokenKlasseninstanz
<code>ValueAsList</code>	Das als Stringliste dargestellte Token
<code>ValueAsNumber</code>	Das als Zahl dargestellte Token
<code>ValueAsString</code>	Das als Zeichenfolge dargestellte Token

Um beispielsweise einen Parameter in einer Bucket Definition zu verwenden:

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  
};
```

Bereitstellung mit Parametern

Wenn Sie eine generierte AWS CloudFormation Vorlage über die AWS CloudFormation Konsole bereitstellen, werden Sie aufgefordert, die Werte für jeden Parameter anzugeben.

Sie können Parameterwerte auch mit dem CLI `cdk deploy` CDK-Befehl oder durch Angabe von Parameterwerten in der Stack-Datei Ihres CDK-Projekts angeben.

Bereitstellung von Parameterwerten mit `cdk deploy`

Wenn Sie die Bereitstellung mit dem CLI `cdk deploy` CDK-Befehl durchführen, können Sie bei der Bereitstellung mit der `--parameters` Option Parameterwerte angeben.

Im Folgenden finden Sie ein Beispiel für die `cdk deploy` Befehlsstruktur:

```
$ cdk deploy stack-logical-id --parameters stack-name:parameter-name=parameter-value
```

Wenn Ihre CDK-App einen einzelnen Stack enthält, müssen Sie weder das logische Stack-ID-Argument noch den `stack-name` Wert in der `--parameters` Option angeben. Das CDK CLI findet diese Werte automatisch und stellt sie bereit. Im Folgenden finden Sie ein Beispiel, das einen `uploadbucket` Wert für den `uploadBucketName` Parameter des einzelnen Stacks in unserer CDK-App angibt:

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

Bereitstellung von Parameterwerten mit `cdk deploy` für Multi-Stack-Anwendungen

Im Folgenden finden Sie ein Beispiel für eine CDK-Anwendung `TypeScript`, die zwei CDK-Stacks enthält. Jeder Stack enthält eine Amazon S3 S3-Bucket-Instance und einen Parameter zum Festlegen des Amazon S3 S3-Bucket-Namens:

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

// Define the CDK app
const app = new cdk.App();

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

    // Set a default parameter name
    const bucketNameParam = new cdk.CfnParameter(this, 'bucketNameParam', {
      type: 'String',
      default: 'myfirststackdefaultbucketname'
    });

    // Define an S3 bucket
    new s3.Bucket(this, 'MyFirstBucket', {
      bucketName: bucketNameParam.valueAsString
    });
  }
}
```



```
}

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

    // Set a default parameter name
    const bucketNameParam = new cdk.CfnParameter(this, 'bucketNameParam', {
      type: 'String',
      default: 'mysecondstackdefaultbucketname'
    });

    // Define an S3 bucket
    new s3.Bucket(this, 'MySecondBucket', {
      bucketName: bucketNameParam.valueAsString
    });
  }
}

// Instantiate the stacks
new MyFirstStack(app, 'MyFirstStack', {
  stackName: 'MyFirstDeployedStack',
});

new MySecondStack(app, 'MySecondStack', {
  stackName: 'MySecondDeployedStack',
});
```

Für CDK-Apps, die mehrere Stacks enthalten, können Sie Folgendes tun:

- Einen Stack mit Parametern bereitstellen — Um einen einzelnen Stack aus einer Multi-Stack-Anwendung bereitzustellen, geben Sie die logische Stack-ID als Argument an.

Im Folgenden finden Sie ein Beispiel für die Bereitstellung `MySecondStack` mit `mynewbucketname` als Parameterwert für: `bucketNameParam`

```
$ cdk deploy MySecondStack --parameters bucketNameParam='mynewbucketname'
```

- Alle Stacks bereitstellen und Parameterwerte für jeden Stack angeben — Geben Sie den `'*'` Platzhalter oder die `--all` Option zur Bereitstellung aller Stapel an. Geben Sie die `--`

parameters Option mehrmals in einem einzigen Befehl an, um Parameterwerte für jeden Stack anzugeben. Im Folgenden wird ein Beispiel gezeigt:

```
$ cdk deploy '*' --
parameters MyFirstDeployedStack:bucketNameParam='mynewfirststackbucketname' --
parameters MySecondDeployedStack:bucketNameParam='mynewsecondstackbucketname'
```

- Alle Stapel bereitstellen und Parameterwerte für einen einzelnen Stapel angeben — Geben Sie den '*' Platzhalter oder die --all Option zum Bereitstellen aller Stapel an. Geben Sie dann in der Option den Stack an, für den der Parameter definiert werden soll. --parameters Im Folgenden finden Sie Beispiele, die alle Stacks in einer CDK-App bereitstellen und einen Parameterwert für den Stack angeben. MySecondDeployedStack AWS CloudFormation Alle anderen Stacks werden den Standardparameterwert bereitstellen und verwenden:

```
$ cdk deploy '*' --parameters MySecondDeployedStack:bucketNameParam='mynewbucketname'
$ cdk deploy --all --
parameters MySecondDeployedStack:bucketNameParam='mynewbucketname'
```

Bereitstellung von Parameterwerten cdk deploy für Anwendungen mit verschachtelten Stacks

Das CLI CDK-Verhalten bei der Arbeit mit Anwendungen, die verschachtelte Stacks enthalten, ähnelt dem von Multi-Stack-Anwendungen. Der Hauptunterschied besteht darin, dass Sie den Platzhalter verwenden müssen, wenn Sie alle verschachtelten Stacks bereitstellen möchten. '*' Der '*' Platzhalter stellt alle Stacks bereit, verteilt aber keine verschachtelten Stacks. Der '*' Platzhalter stellt alle Stapel bereit, einschließlich verschachtelter Stacks.

Das folgende Beispiel zeigt, wie verschachtelte Stacks bereitgestellt und gleichzeitig der Parameterwert für einen verschachtelten Stapel angegeben wird:

```
$ cdk deploy '*' --parameters MultiStackCdkApp/
SecondStack:bucketNameParam='mysecondstackbucketname'
```

Weitere Informationen zu cdk deploy Befehlsoptionen finden Sie unter [cdk deploy](#)

Tagging

Tags sind informative Schlüsselwertelemente, die Sie zu Konstrukten in Ihrer App hinzufügen können. AWS CDK Ein Tag, der auf ein bestimmtes Konstrukt angewendet wird, gilt auch für alle seine markierbaren untergeordneten Objekte. Tags sind in der AWS CloudFormation Vorlage enthalten, die aus Ihrer App synthetisiert wurde, und werden auf die AWS Ressourcen angewendet, die sie bereitstellt. Sie können Tags verwenden, um Ressourcen für die folgenden Zwecke zu identifizieren und zu kategorisieren:

- Vereinfachung der Verwaltung
- Kostenzuordnung
- Zugriffskontrolle
- Alle anderen Zwecke, die Sie sich ausdenken

Tip

Weitere Informationen darüber, wie Sie Tags mit Ihren AWS Ressourcen verwenden können, finden Sie im Whitepaper unter [Best Practices für das AWS Tagging von AWS Ressourcen](#).

Themen

- [Verwenden von Markierungen](#)
- [Tag-Prioritäten](#)
- [Optionale Eigenschaften](#)
- [Beispiel](#)
- [Markieren einzelner Konstrukte](#)

Verwenden von Markierungen

Die [Tags](#)-Klasse umfasst die statische Methode `of()`, mit der Sie dem angegebenen Konstrukt Tags hinzufügen oder Tags daraus entfernen können.

- [Tags.of\(SCOPE\).add\(\)](#) wendet ein neues Tag auf das angegebene Konstrukt und alle seine untergeordneten Objekte an.

- `Tags.of(SCOPE).remove()` entfernt ein Tag aus dem angegebenen Konstrukt und allen seinen untergeordneten Objekten, einschließlich Tags, die ein untergeordnetes Konstrukt möglicherweise auf sich selbst angewendet hat.

Note

Das Tagging wird implementiert mit [the section called "Aspekte"](#). Aspekte sind eine Möglichkeit, eine Operation (z. B. Tagging) auf alle Konstrukte in einem bestimmten Bereich anzuwenden.

Im folgenden Beispiel wird der Tag-Schlüssel mit dem Wertwert auf ein Konstrukt angewendet.

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");
```

Go

```
awscdk.Tags_Of(myConstruct).Add(jsii.String("key"), jsii.String("value"),  
&awscdk.TagProps{}
```

Im folgenden Beispiel wird der Tag-Schlüssel aus einem Konstrukt gelöscht.

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");
```

Go

```
awsdk.Tags_Of(myConstruct).Remove(jsii.String("key"), &awsdk.TagProps{})
```

Wenn Sie Stage Konstrukte verwenden, wenden Sie das Tag auf der Stage Ebene oder darunter an. Beschriftungen werden nicht Stage grenzübergreifend angewendet.

Tag-Prioritäten

Der AWS CDK wendet Tags rekursiv an und entfernt sie. Bei Konflikten gewinnt der Tagging-Vorgang mit der höchsten Priorität. (Prioritäten werden mithilfe der optionalen `priority` Eigenschaft festgelegt.) Wenn die Prioritäten von zwei Operationen identisch sind, gewinnt die Tagging-Operation, die sich am Ende des Konstruktionsbaums am nächsten befindet. Standardmäßig hat das Anwenden eines Tags eine Priorität von 100 (mit Ausnahme von direkt zu einer AWS CloudFormation Ressource hinzugefügten Tags, die eine Priorität von 50 haben). Die Standardpriorität für das Entfernen eines Tags ist 200.

Im Folgenden wird ein Tag mit der Priorität 300 auf eine Konstruktion angewendet.

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 });
```

Go

```
awsdk.Tags_Of(myConstruct).Add(jsii.String("key"), jsii.String("value"),  
&awsdk.TagProps{  
  Priority: jsii.Number(300),  
})
```

Optionale Eigenschaften

Mithilfe von [properties](#) Stichwörtern können Sie genau festlegen, wie Tags auf Ressourcen angewendet oder aus Ressourcen entfernt werden. Alle anderen Eigenschaften sind optional.

`applyToLaunchedInstances`(Python:`apply_to_launched_instances`)

Nur für `add()` verfügbar. Standardmäßig werden Tags auf Instances angewendet, die in einer Auto Scaling Scaling-Gruppe gestartet wurden. Setzen Sie diese Eigenschaft auf `false`, um Instances zu ignorieren, die in einer Auto Scaling Scaling-Gruppe gestartet wurden.

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

Verwenden Sie diese, um Tags nur für eine Teilmenge von Ressourcen zu manipulieren, basierend auf AWS CloudFormation Ressourcentypen. Standardmäßig wird die Operation auf alle Ressourcen im Konstruktionsunterbaum angewendet. Dies kann jedoch geändert werden, indem bestimmte Ressourcentypen ein- oder ausgeschlossen werden. `Exclude` hat Vorrang vor `Include`, wenn beide angegeben sind.

`priority`

Verwenden Sie diese Option, um die Priorität dieser Operation im Vergleich zu anderen Tags.`add()` Tags.`remove()` AND-Vorgängen festzulegen. Höhere Werte haben Vorrang vor niedrigeren Werten. Die Standardeinstellung ist 100 für Hinzufügevorgänge (50 für direkt auf AWS CloudFormation Ressourcen angewendete Tags) und 200 für Entfernungsvorgänge.

Im folgenden Beispiel wird der Tagname mit dem Wert Wert und der Priorität 100 auf Ressourcen des Typs `AWS::Xxx::Yyy` im Konstrukt angewendet. Es wendet das Tag nicht auf Instances an, die in einer Amazon EC2 Auto Scaling Scaling-Gruppe gestartet wurden, oder auf Ressourcen des Typs `AWS::Xxx::Zzz`. (Dies sind Platzhalter für zwei beliebige, aber unterschiedliche AWS CloudFormation Ressourcentypen.)

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("tagname", "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
});

```

Go

```

awscdk.Tags_Of(myConstruct).Add(jsii.String("tagname"), jsii.String("value"),
    &awscdk.TagProps{
        ApplyToLaunchedInstances: jsii.Bool(false),
        IncludeResourceTypes:      &[*string]{jsii.String("AWS::Xxx::Yyy")},
        ExcludeResourceTypes:      &[*string]{jsii.String("AWS::Xxx::Zzz")},
        Priority:                   jsii.Number(100),
    })

```


Im folgenden Beispiel wird das Tag `tagname` mit der Priorität 200 aus Ressourcen des Typs `AWS::Xxx::Yyy` entfernt, aber nicht aus Ressourcen des Typs `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
});
```

Go

```
awscdk.Tags_Of(myConstruct).Remove(jsii.String("tagname"), &awscdk.TagProps{
    IncludeResourceTypes: &[*string]{jsii.String("AWS::Xxx:Yyy")},
    ExcludeResourceTypes: &[*string]{jsii.String("AWS::Xxx:Zzz")},
    Priority:              jsii.Number(200),
})
```

Beispiel

Im folgenden Beispiel wird der Tag-Schlüssel `StackType` mit Wert `TheBest` zu jeder Ressource hinzugefügt, die innerhalb des Stack Namens `MarketingSystem` erstellt wurde. Dann entfernt es es wieder aus allen Ressourcen außer Amazon EC2 VPC-Subnetzen. Das Ergebnis ist, dass das Tag nur auf die Subnetze angewendet wurde.

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"]
});
```

Go

```
import "github.com/aws/aws-cdk-go/awscdk/v2"
app := awscdk.NewApp(nil)
theBestStack := awscdk.NewStack(app, jsii.String("MarketingSystem"),
    &awscdk.StackProps{})

// Add a tag to all constructs in the stack
awscdk.Tags_Of(theBestStack).Add(jsii.String("StackType"), jsii.String("TheBest"),
    &awscdk.TagProps{})

// Remove the tag from all resources except subnet resources
awscdk.Tags_Of(theBestStack).Add(jsii.String("StackType"), jsii.String("TheBest"),
    &awscdk.TagProps{
        ExcludeResourceTypes: &[*string]{jsii.String("AWS::EC2::Subnet")},
    })
```

Der folgende Code erzielt dasselbe Ergebnis. Überlegen Sie, welcher Ansatz (Inklusion oder Ausschluss) Ihre Absicht klarer macht.

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"]  
});
```

Go

```
awscdk.Tags_Of(theBestStack).Add(jsii.String("StackType"), jsii.String("TheBest"),  
&awscdk.TagProps{  
    IncludeResourceTypes: &[*string{jsii.String("AWS::EC2::Subnet")}],  
})
```

Markieren einzelner Konstrukte

`Tags.of(scope).add(key, value)` ist die Standardmethode zum Hinzufügen von Tags zu Konstrukten in der AWS CDK. Das Verhalten beim Tree-Walking, bei dem rekursiv alle Ressourcen, die mit Tags versehen werden können, innerhalb des angegebenen Bereichs markiert werden, ist fast immer das, was Sie wollen. Manchmal müssen Sie jedoch ein bestimmtes, beliebiges Konstrukt (oder mehrere Konstrukte) taggen.

Ein solcher Fall beinhaltet das Anwenden von Tags, deren Wert von einer Eigenschaft des zu markierenden Konstrukts abgeleitet wird. Der Standardansatz für Tagging wendet rekursiv denselben Schlüssel und Wert auf alle passenden Ressourcen im Bereich an. Hier könnte der Wert jedoch für jedes markierte Konstrukt unterschiedlich sein.

Tags werden mithilfe von [Aspekten](#) implementiert, und das CDK ruft die `visit()` Methode des `Tags` für jedes Konstrukt innerhalb des von Ihnen angegebenen `Tags.of(scope)` Bereichs auf. Wir können `Tag.visit()` direkt aufrufen, um ein Tag auf ein einzelnes Konstrukt anzuwenden.

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);
```

Go

```
awscdk.NewTag(key, value, &awscdk.TagProps{}).Visit(scope)
```

Sie können alle Konstrukte innerhalb eines Gültigkeitsbereichs taggen, die Werte der Tags jedoch von den Eigenschaften der einzelnen Konstrukte ableiten lassen. Schreiben Sie dazu einen Aspekt und wenden Sie das Tag in der `visit()` Methode des Aussehens an, wie im vorherigen Beispiel gezeigt. Fügen Sie dann den Aspekt mit dem gewünschten Bereich hinzu `Aspects.of(scope).add(aspect)`.

Im folgenden Beispiel wird jeder Ressource in einem Stapel, der den Pfad der Ressource enthält, ein Tag zugewiesen.

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

Die Logik der bedingten Kennzeichnung, einschließlich Prioritäten, Ressourcentypen usw., ist in die Tag Klasse integriert. Sie können diese Funktionen verwenden, wenn Sie Tags auf beliebige Ressourcen anwenden. Das Tag wird nicht angewendet, wenn die Bedingungen nicht erfüllt sind. Außerdem kennzeichnet die Tag Klasse nur Ressourcen, die mit Tags versehen werden können, sodass Sie nicht testen müssen, ob ein Konstrukt markierbar ist, bevor Sie ein Tag anwenden.

Objekte

Assets sind lokale Dateien, Verzeichnisse oder Docker-Images, die in AWS CDK Bibliotheken und Apps gebündelt werden können. Eine Komponente könnte beispielsweise ein Verzeichnis sein, das den Handler-Code für eine - AWS Lambda Funktion enthält. Assets können jedes Artefakt darstellen, das die App für den Betrieb benötigt.

Das folgende Tutorial-Video bietet einen umfassenden Überblick über CDK-Komponenten und erklärt, wie Sie sie in Ihrer Infrastruktur als Code (IaC) verwenden können.

[Erklärte CDK-Assets](#)

Sie fügen Komponenten über APIs hinzu, die durch bestimmte AWS Konstrukte verfügbar gemacht werden. Wenn Sie beispielsweise ein [Lambda.Function](#)-Konstrukt definieren, können Sie mit der [Code](#)-Eigenschaft eine [Komponente](#) (Verzeichnis) übergeben. `Function` verwendet Komponenten, um den Inhalt des Verzeichnisses zu bündeln und für den Code der Funktion zu verwenden. Ebenso verwendet [ecs.ContainerImage.fromAsset](#) ein Docker-Image, das aus einem lokalen Verzeichnis erstellt wurde, wenn eine Amazon-ECS-Aufgabendefinition definiert wird.

Komponenten im Detail

Wenn Sie auf eine Komponente in Ihrer App verweisen, enthält die [Cloud-Baugruppe](#), die aus Ihrer Anwendung synthetisiert wird, Metadateninformationen mit Anweisungen für die AWS CDK CLI. Die Anweisungen enthalten, wo die Komponente auf der lokalen Festplatte zu finden ist und

welche Art von Bündelung basierend auf dem Komponententyp ausgeführt werden soll, z. B. ein zu komprimierendes Verzeichnis (zip) oder ein zu erstellendes Docker-Image.

Der AWS CDK generiert einen Quell-Hash für Komponenten. Dies kann zur Konstruktionszeit verwendet werden, um festzustellen, ob sich der Inhalt einer Komponente geändert hat.

Standardmäßig AWS CDK erstellt die eine Kopie der Komponente im Cloud-Assembly-Verzeichnis, das standardmäßig `istcdk.out`, unter dem Quell-Hash. Auf diese Weise ist die Cloud-Baugruppe eigenständig. Wenn sie zur Bereitstellung auf einen anderen Host verschoben wird, kann sie dennoch bereitgestellt werden. Details dazu finden Sie unter [the section called “Cloud-Komponenten”](#).

Wenn die eine App AWS CDK bereitstellt, die auf Komponenten verweist (entweder direkt durch den Anwendungscode oder über eine Bibliothek), bereitet die AWS CDK CLI die Komponenten zunächst vor und veröffentlicht sie in einem Amazon S3-Bucket oder Amazon-ECR-Repository. (Der S3-Bucket oder das Repository wird während des Bootstrappings erstellt.) Erst dann werden die im Stack definierten Ressourcen bereitgestellt.

In diesem Abschnitt werden die im Framework verfügbaren Low-Level-APIs beschrieben.

Komponententypen

Die AWS CDK unterstützt die folgenden Arten von Komponenten:

Amazon S3-Komponenten

Dies sind lokale Dateien und Verzeichnisse, die der in Amazon S3 AWS CDK hochlädt.

Docker-Image

Dies sind Docker-Images, die der auf Amazon ECR AWS CDK hochlädt.

Diese Komponententypen werden in den folgenden Abschnitten erläutert.

Amazon S3-Komponenten

Sie können lokale Dateien und Verzeichnisse als Assets definieren, die AWS CDK Pakete und laden sie über das Modul [aws-s3-assets](#) in Amazon S3 hoch.

Im folgenden Beispiel werden eine lokale Verzeichniskomponente und eine Dateikomponente definiert.

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")),
  })
```

In den meisten Fällen müssen Sie die APIs im `aws-s3-assets` Modul nicht direkt verwenden. Module, die Komponenten unterstützen, wie z. B. `aws-lambda`, verfügen über praktische Methoden, mit denen Sie Komponenten verwenden können. Bei Lambda-Funktionen können Sie mit der statischen Methode [fromAsset \(\)](#) ein Verzeichnis oder eine ZIP-Datei im lokalen Dateisystem angeben.

Beispiel für eine Lambda-Funktion

Ein häufiger Anwendungsfall ist das Erstellen von Lambda-Funktionen mit dem Handler-Code als Amazon S3-Komponente.

Im folgenden Beispiel wird eine Amazon S3-Komponente verwendet, um einen Python-Handler im lokalen Verzeichnis zu definieren `handler`. Außerdem wird eine Lambda-Funktion mit der lokalen Verzeichniskomponente als `-code`Eigenschaft erstellt. Im Folgenden finden Sie den Python-Code für den Handler.

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

Der Code für die Haupt AWS CDK -App sollte wie folgt aussehen.

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
}
```

Die `Function` Methode verwendet Komponenten, um den Inhalt des Verzeichnisses zu bündeln und ihn für den Code der Funktion zu verwenden.

Tip

Java-`.jar` Dateien sind ZIP-Dateien mit einer anderen Erweiterung. Diese werden unverändert in Amazon S3 hochgeladen, aber wenn sie als Lambda-Funktion bereitgestellt werden, werden die darin enthaltenen Dateien extrahiert, was Sie möglicherweise nicht wünschen. Um dies zu vermeiden, platzieren Sie die `.jar` Datei in einem Verzeichnis und geben Sie dieses Verzeichnis als Komponente an.

Beispiel für die Bereitstellung von Zeitattributen

Amazon S3 [Komponententypen stellen auch Bereitstellungszeitattribute](#) bereit, auf die in AWS CDK Bibliotheken und Apps verwiesen werden kann. Der AWS CDK CLI-Befehl `cdk synth` zeigt Komponenteneigenschaften als AWS CloudFormation Parameter an.

Im folgenden Beispiel werden Bereitstellungszeitattribute verwendet, um den Speicherort einer Image-Komponente als Umgebungsvariablen an eine Lambda-Funktion zu übergeben. (Die Art der Datei spielt keine Rolle; das hier verwendete PNG-Image ist nur ein Beispiel.)

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(),
    },
  })

```

Berechtigungen

Wenn Sie Amazon S3-Komponenten direkt über das Modul [aws-s3-assets](#), IAM-Rollen, Benutzer oder Gruppen verwenden und Komponenten zur Laufzeit lesen müssen, gewähren Sie diesen Komponenten IAM-Berechtigungen über die Methode [asset.grantRead](#).

Im folgenden Beispiel wird einer IAM-Gruppe Leseberechtigungen für eine Dateikomponente erteilt.

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/awss3assets"
    "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 := awssiam.NewGroup(stack, jsii.String("MyUserGroup"), &awssiam.GroupProps{})

asset.GrantRead(group)
```

Docker-Image-Komponenten

unterstützt AWS CDK das Bündeln lokaler Docker-Images als Komponenten über das [-aws-ecr-assets](#) Modul.

Das folgende Beispiel definiert ein Docker-Image, das lokal erstellt und an Amazon ECR übertragen wird. Images werden aus einem lokalen Docker-Kontextverzeichnis (mit einer Docker-Datei) erstellt

und von der AWS CDK CLI oder der CI/CD-Pipeline Ihrer App in Amazon ECR hochgeladen. Die Bilder können natürlich in Ihrer AWS CDK App referenziert werden.

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")),
    })
```

Das `my-image` Verzeichnis muss ein Dockerfile enthalten. Die AWS CDK CLI erstellt ein Docker-Image aus `my-image`, überträgt es in ein Amazon ECR-Repository und gibt den Namen des Repositorys als Parameter AWS CloudFormation für Ihren Stack an. [Docker-Image-Komponententypen stellen Bereitstellungszeitattribute](#) bereit, auf die in AWS CDK Bibliotheken und Apps verwiesen werden kann. Der AWS CDK CLI-Befehl `cdk synth` zeigt Komponenteneigenschaften als AWS CloudFormation Parameter an.

Beispiel für eine Amazon-ECS-Aufgabendefinition

Ein häufiger Anwendungsfall besteht darin, ein Amazon ECS zum Ausführen von Docker-Containern [TaskDefinition](#) zu erstellen. Das folgende Beispiel gibt den Speicherort einer Docker-Image-Komponente an, die die lokal AWS CDK erstellt und an Amazon ECR überträgt.

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),
    })
```

Beispiel für die Bereitstellung von Zeitattributen

Das folgende Beispiel zeigt, wie Sie die Bereitstellungszeitattribute `repository` und `imageUri`, um eine Amazon-ECS-Aufgabendefinition mit dem AWS Fargate Starttyp zu erstellen. Beachten Sie, dass für die Amazon-ECR-Repo-Suche das Tag des Images erforderlich ist, nicht der URI. Daher schneiden wir es vom Ende des URI der Komponente aus.

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()),  
    })  
})
```

Beispiel für Build-Argumente

Sie können benutzerdefinierte Build-Argumente für den Docker-Build bereitstellen, indem Sie die Eigenschaftsoption `buildArgs` (Python: `build_args`) durchlaufen, wenn die AWS CDK CLI das Image während der Bereitstellung erstellt.

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"),
},
})
```

Berechtigungen

Wenn Sie ein Modul verwenden, das Docker-Image-Assets unterstützt, z. B. [aws-ecs](#), AWS CDK verwaltet die Berechtigungen für Sie, wenn Sie Assets direkt oder über verwenden [ContainerImage.fromEcrRepository](#) (Python: `from_ecr_repository`). Wenn Sie Docker-Image-Komponenten direkt verwenden, stellen Sie sicher, dass der verbrauchende Prinzipal über Berechtigungen zum Abrufen des Images verfügt.

In den meisten Fällen sollten Sie die Methode [asset.repository.grantPull](#) verwenden (Python: `grant_pull`). Dadurch wird die IAM-Richtlinie des Prinzipals geändert, damit dieser Images aus diesem Repository abrufen kann. Wenn sich der Prinzipal, der das Image abrufen, nicht im selben Konto befindet oder wenn es sich um einen - AWS Service handelt, der keine Rolle in Ihrem Konto annimmt (z. B. AWS CodeBuild), müssen Sie Pull-Berechtigungen für die Ressourcenrichtlinie und nicht für die Richtlinie des Prinzipals erteilen. Verwenden Sie die Methode [asset.repository.addToResourcePolicy](#) (Python: `add_to_resource_policy`), um die entsprechenden Prinzipalberechtigungen zu erteilen.

AWS CloudFormation Ressourcenmetadaten

Note

Dieser Abschnitt ist nur für Konstruktoren relevant. In bestimmten Situationen müssen Tools wissen, dass eine bestimmte CFN-Ressource eine lokale Komponente verwendet. Sie können beispielsweise die AWS SAM CLI verwenden, um Lambda-Funktionen lokal für Debugging-Zwecke aufzurufen. Details dazu finden Sie unter [the section called "AWS SAM - Integration"](#).

Um solche Anwendungsfälle zu aktivieren, konsultieren externe Tools eine Reihe von Metadateneinträgen für - AWS CloudFormation Ressourcen:

- `aws:asset:path` – zeigt auf den lokalen Pfad der Komponente.

- `aws:asset:property` – Der Name der Ressourceneigenschaft, in der die Komponente verwendet wird.

Mithilfe dieser beiden Metadateneinträge können Tools feststellen, dass Komponenten von einer bestimmten Ressource verwendet werden, und erweiterte lokale Erlebnisse ermöglichen.

Um diese Metadateneinträge einer Ressource hinzuzufügen, verwenden Sie die `add_resource_metadata` Methode `asset.addResourceMetadata` (Python:).

Berechtigungen

Die AWS Construct Library verwendet einige gängige, weit verbreitete Ausdrücke, um den Zugriff und die Berechtigungen zu verwalten. Das IAM-Modul stellt Ihnen die Tools zur Verfügung, die Sie für die Verwendung dieser Ausdrücke benötigen.

AWS CDK verwendet AWS CloudFormation, um Änderungen bereitzustellen. Jede Bereitstellung umfasst einen Akteur (entweder einen Entwickler oder ein automatisiertes System), der eine AWS CloudFormation Bereitstellung startet. In diesem Fall übernimmt der Akteur eine oder mehrere IAM-Identitäten (Benutzer oder Rollen) und übergibt optional eine Rolle an AWS CloudFormation.

Wenn Sie verwenden, AWS IAM Identity Center um sich als Benutzer zu authentifizieren, stellt der Single-Sign-On-Anbieter kurzlebige Sitzungsanmeldeinformationen bereit, mit denen Sie als vordefinierte IAM-Rolle fungieren können. Informationen dazu, wie die AWS Anmeldeinformationen von der IAM-Identity-Center-Authentifizierung AWS CDK erhält, finden Sie unter [Grundlegendes zur IAM-Identity-Center-Authentifizierung](#) im AWS Referenzhandbuch für SDKs und Tools.

Auftraggeber

Ein IAM-Prinzipal ist eine authentifizierte AWS Entität, die einen Benutzer, Service oder eine Anwendung darstellt, die - AWS APIs aufrufen kann. Die AWS Construct Library unterstützt die Angabe von Prinzipalen auf verschiedene flexible Weisen, um ihnen Zugriff auf Ihre - AWS Ressourcen zu gewähren.

Im Sicherheitskontext bezieht sich der Begriff „Prinzipal“ speziell auf authentifizierte Entitäten wie Benutzer. Objekte wie Gruppen und Rollen stellen keine Benutzer (und andere authentifizierte Entitäten) dar, sondern identifizieren sie indirekt, um Berechtigungen zu erteilen.

Wenn Sie beispielsweise eine IAM-Gruppe erstellen, können Sie der Gruppe (und damit ihren Mitgliedern) Schreibzugriff auf eine Amazon-RDS-Tabelle gewähren. Die Gruppe selbst ist jedoch

kein Prinzipal, da sie keine einzelne Entität darstellt (Sie können sich auch nicht bei einer Gruppe anmelden).

In der IAM-Bibliothek des CDK implementieren Klassen, die Prinzipale direkt oder indirekt identifizieren, die [IPrincipal](#) Schnittstelle, sodass diese Objekte in Zugriffsrichtlinien austauschbar verwendet werden können. Nicht alle sind jedoch Prinzipale im Sicherheitsbereich. Zu diesen Objekten gehören:

1. IAM-Ressourcen wie [Role](#), [User](#) und [Group](#)
2. Service-Prinzipale (`new iam.ServicePrincipal('service.amazonaws.com')`)
3. Verbundprinzipale (`new iam.FederatedPrincipal('cognito-identity.amazonaws.com')`)
4. Kontoprinzipale (`new iam.AccountPrincipal('0123456789012')`)
5. Kanonische Benutzerprinzipale (`new iam.CanonicalUserPrincipal('79a59d[...]7ef2be')`)
6. AWS Organizations -Prinzipale (`new iam.OrganizationPrincipal('org-id')`)
7. Beliebige ARN-Prinzipale (`new iam.ArnPrincipal(res.arn)`)
8. Ein `iam.CompositePrincipal(principal1, principal2, ...)`, um mehreren Prinzipalen zu vertrauen

Gewährungen

Jedes Konstrukt, das eine Ressource darstellt, auf die zugegriffen werden kann, z. B. ein Amazon S3-Bucket oder eine Amazon-DynamoDB-Tabelle, verfügt über Methoden, die einer anderen Entität Zugriff gewähren. Alle diese Methoden haben Namen, die mit der Erteilung beginnen.

Amazon S3-Buckets verfügen beispielsweise über die Methoden [grantRead](#) und [grantReadWrite](#) (Python: `grant_read`, `grant_read_write`), um Lese- bzw. Lese-/Schreibzugriff von einer Entität auf den Bucket zu ermöglichen. Die Entität muss nicht genau wissen, welche Amazon S3-IAM-Berechtigungen für diese Operationen erforderlich sind.

Das erste Argument einer Erteilungsmethode ist immer vom Typ [IGrantable](#). Diese Schnittstelle stellt Entitäten dar, denen Berechtigungen erteilt werden können. Das heißt, es stellt Ressourcen mit Rollen dar, z. B. die IAM-[Role](#)-Objekte [User](#), und [Group](#).

Anderen Entitäten können auch Berechtigungen erteilt werden. Später in diesem Thema zeigen wir beispielsweise, wie Sie einem CodeBuild Projekt Zugriff auf einen Amazon S3-Bucket gewähren. Im

Allgemeinen wird die zugehörige Rolle über eine `role` Eigenschaft der Entität abgerufen, der Zugriff gewährt wird.

Ressourcen, die Ausführungsrollen verwenden, wie z. B. [lambda.Function](#), implementieren auch `IGrantable`, sodass Sie ihnen Zugriff direkt gewähren können, anstatt Zugriff auf ihre Rolle zu gewähren. Wenn beispielsweise `bucket` ein Amazon S3-Bucket und eine Lambda-Funktion `function` ist, gewährt der folgende Code der Funktion Lesezugriff auf den Bucket.

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

Manchmal müssen Berechtigungen angewendet werden, während Ihr Stack bereitgestellt wird. Ein solcher Fall ist, wenn Sie einer AWS CloudFormation benutzerdefinierten Ressource Zugriff auf eine andere Ressource gewähren. Die benutzerdefinierte Ressource wird während der Bereitstellung aufgerufen, daher muss sie zum Zeitpunkt der Bereitstellung über die angegebenen Berechtigungen verfügen.

Ein weiterer Fall ist, wenn ein Service überprüft, ob auf die Rolle, die Sie ihm übergeben, die richtigen Richtlinien angewendet wurden. (Eine Reihe von AWS Services tun dies, um sicherzustellen, dass Sie nicht vergessen haben, die Richtlinien festzulegen.) In diesen Fällen schlägt die Bereitstellung möglicherweise fehl, wenn die Berechtigungen zu spät angewendet werden.

Um zu erzwingen, dass die Berechtigungen der Erteilung angewendet werden, bevor eine andere Ressource erstellt wird, können Sie eine Abhängigkeit von der Erteilung selbst hinzufügen, wie hier gezeigt. Obwohl der Rückgabewert von Erteilungsmethoden häufig verworfen wird, gibt jede Erteilungsmethode tatsächlich ein `iam.Grant` Objekt zurück.

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);
```

Rollen

Das IAM-Paket enthält ein [Role](#) Konstrukt, das IAM-Rollen darstellt. Der folgende Code erstellt eine neue Rolle, die dem Amazon EC2-Service vertraut.

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
});
```

Sie können einer Rolle Berechtigungen hinzufügen, indem Sie die `-addToPolicy` Methode der Rolle (Python: `add_to_policy`) aufrufen und eine übergeben `PolicyStatement`, die die hinzuzufügende Regel definiert. Die Anweisung wird der Standardrichtlinie der Rolle hinzugefügt. Wenn sie keine enthält, wird eine erstellt.

Im folgenden Beispiel wird der Rolle eine Deny Richtlinienanweisung für die Aktionen `ec2:SomeAction` und `s3:AnotherAction` für die Ressourcen `bucket` und `otherRole` (Python: `other_role`) unter der Bedingung hinzugefügt, dass der autorisierte Service ist 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"
        }
    }
}));

```

Im vorherigen Beispiel haben wir eine neue [PolicyStatement](#) Inline mit dem Aufruf [addToPolicy](#) (Python: `add_to_policy`) erstellt. Sie können auch eine vorhandene oder eine von Ihnen geänderte Richtlinienanweisung übergeben. Das [PolicyStatement](#)-Objekt verfügt über [zahlreiche Methoden](#) zum Hinzufügen von Prinzipalen, Ressourcen, Bedingungen und Aktionen.

Wenn Sie ein Konstrukt verwenden, für das eine Rolle ordnungsgemäß funktioniert, können Sie einen der folgenden Schritte ausführen:

- Übergeben Sie beim Instanzieren des Konstruktobjekts eine vorhandene Rolle.
- Lassen Sie das Konstrukt eine neue Rolle für Sie erstellen und vertrauen Sie dem entsprechenden Service-Prinzipal. Im folgenden Beispiel wird ein solches Konstrukt verwendet: ein CodeBuild Projekt.

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
});
```

Sobald das Objekt erstellt wurde, ist die Rolle (unabhängig davon, ob die übergebene Rolle oder die vom Konstrukt erstellte Standardrolle) als Eigenschaft verfügbar `role`. Diese Eigenschaft ist jedoch nicht für externe Ressourcen verfügbar. Daher haben diese Konstrukte eine `addToRolePolicy` (Python: `add_to_role_policy`)-Methode.

Die `-`-Methode tut nichts, wenn das Konstrukt eine externe Ressource ist, und ruft andernfalls die `-`-Methode `addToPolicy` (Python: `add_to_policy`) der `-role`-Eigenschaft auf. Dadurch haben Sie die Möglichkeit, den nicht definierten Fall explizit zu behandeln.

Das folgende Beispiel zeigt:

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  
}));
```

Ressourcenrichtlinien

Einige Ressourcen in AWS, wie Amazon S3-Buckets und IAM-Rollen, verfügen ebenfalls über eine Ressourcenrichtlinie. Diese Konstrukte haben eine `-addToResourcePolicyMethode` (Python: `add_to_resource_policy`), die ein [PolicyStatement](#) als Argument verwendet. Jede Richtlinienanweisung, die einer Ressourcenrichtlinie hinzugefügt wird, muss mindestens einen Prinzipal angeben.

Im folgenden Beispiel bucket erteilt der [Amazon S3-Bucket](#) eine Rolle mit der `-s3:SomeAction`Berechtigung für sich selbst.

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 }
}));
```

Verwenden externer IAM-Objekte

Wenn Sie einen IAM-Benutzer, Prinzipal, eine Gruppe oder Rolle außerhalb Ihrer AWS CDK App definiert haben, können Sie dieses IAM-Objekt in Ihrer AWS CDK App verwenden. Erstellen Sie dazu einen Verweis darauf mit seinem ARN oder seinem Namen. (Verwenden Sie den Namen für Benutzer, Gruppen und Rollen.) Die zurückgegebene Referenz kann dann verwendet werden, um Berechtigungen zu erteilen oder Richtlinienanweisungen zu erstellen, wie zuvor erläutert.

- Rufen Sie für Benutzer [User.fromUserArn\(\)](#) oder auf [User.fromUserName\(\)](#). [User.fromUserAttributes\(\)](#) ist ebenfalls verfügbar, bietet aber derzeit die gleiche Funktionalität wie [User.fromUserArn\(\)](#).
- Instanzieren Sie für Prinzipale ein [ArnPrincipal](#) Objekt.
- Rufen Sie für Gruppen [Group.fromGroupArn\(\)](#) oder auf [Group.fromGroupName\(\)](#).
- Rufen Sie für -Rollen [Role.fromRoleArn\(\)](#) oder auf [Role.fromRoleName\(\)](#).

Richtlinien (einschließlich verwalteter Richtlinien) können mit den folgenden Methoden auf ähnliche Weise verwendet werden. Sie können Verweise auf diese Objekte überall verwenden, wo eine IAM-Richtlinie erforderlich ist.

- [Policy.fromPolicyName](#)
- [ManagedPolicy.fromManagedPolicyArn](#)
- [ManagedPolicy.fromManagedPolicyName](#)
- [ManagedPolicy.fromAwsManagedPolicyName](#)

Note

Wie bei allen Verweisen auf externe AWS Ressourcen können Sie externe IAM-Objekte in Ihrer CDK-App nicht ändern.

Laufzeitkontext

Kontextwerte sind Schlüssel-Wert-Paare, die einer App, einem Stack oder einem Konstrukt zugeordnet werden können. Sie können Ihrer App aus einer Datei (normalerweise entweder `cdk.json` oder `cdk.context.json` in Ihrem Projektverzeichnis) oder über die Befehlszeile bereitgestellt werden.

Das CDK Toolkit verwendet Kontext, um Werte zwischenspeichern, die während der Synthese von Ihrem AWS Konto abgerufen wurden. Zu den Werten gehören die Availability Zones in Ihrem Konto oder die Amazon Machine Image (AMI) -IDs, die derzeit für Amazon EC2 EC2-Instances verfügbar sind. Da diese Werte von Ihrem AWS Konto bereitgestellt werden, können sie sich zwischen den Ausführungen Ihrer CDK-Anwendung ändern. Dies macht sie zu einer potenziellen Quelle unbeabsichtigter Änderungen. Das Caching-Verhalten des CDK Toolkits „friert“ diese Werte für Ihre CDK-App ein, bis Sie sich entscheiden, die neuen Werte zu akzeptieren.

Stellen Sie sich das folgende Szenario ohne Kontext-Caching vor. Nehmen wir an, Sie haben „aktuelles Amazon Linux“ als AMI für Ihre Amazon EC2 EC2-Instances angegeben und eine neue Version dieses AMI wurde veröffentlicht. Wenn Sie dann das nächste Mal Ihren CDK-Stack bereitstellen, würden Ihre bereits bereitgestellten Instances das veraltete („falsche“) AMI verwenden und müssten aktualisiert werden. Ein Upgrade würde dazu führen, dass all Ihre vorhandenen Instances durch neue ersetzt werden, was wahrscheinlich unerwartet und unerwünscht wäre.

Stattdessen zeichnet das CDK die verfügbaren AMIs Ihres Kontos in der `cdk.context.json` Datei Ihres Projekts auf und verwendet den gespeicherten Wert für future Synthesevorgänge. Auf diese Weise ist die Liste der AMIs keine potenzielle Änderungsquelle mehr. Sie können auch sicher sein, dass Ihre Stacks immer nach denselben AWS CloudFormation Vorlagen synthetisiert werden.

Nicht alle Kontextwerte sind zwischengespeicherte Werte aus Ihrer Umgebung. AWS [the section called “Feature-Flags”](#) sind auch Kontextwerte. Sie können auch Ihre eigenen Kontextwerte für Ihre Apps oder Konstrukte erstellen.

Kontextschlüssel sind Zeichenketten. Werte können jeden Typ haben, der von JSON unterstützt wird: Zahlen, Zeichenketten, Arrays oder Objekte.

Tip

Wenn Ihre Konstrukte ihre eigenen Kontextwerte erstellen, nehmen Sie den Paketnamen Ihrer Bibliothek in die Schlüssel auf, damit sie nicht mit den Kontextwerten anderer Pakete in Konflikt geraten.

Viele Kontextwerte sind einer bestimmten AWS Umgebung zugeordnet, und eine bestimmte CDK-App kann in mehr als einer Umgebung bereitgestellt werden. Der Schlüssel für solche Werte umfasst das AWS Konto und die Region, sodass Werte aus verschiedenen Umgebungen nicht miteinander in Konflikt geraten.

Der folgende Kontextschlüssel veranschaulicht das von der verwendete Format AWS CDK, einschließlich des Kontos und der Region.

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

Important

Zwischengespeicherte Kontextwerte werden von der AWS CDK und ihren Konstrukten verwaltet, einschließlich Konstrukten, die Sie schreiben können. Fügen Sie zwischengespeicherte Kontextwerte nicht hinzu oder ändern Sie sie nicht, indem Sie Dateien manuell bearbeiten. Es kann jedoch nützlich sein, `cdk.context.json` gelegentlich nachzuschauen, welche Werte zwischengespeichert werden. Kontextwerte, die keine zwischengespeicherten Werte darstellen, sollten unter dem `context` Schlüssel von gespeichert werden. `cdk.json` Auf diese Weise werden sie nicht gelöscht, wenn zwischengespeicherte Werte gelöscht werden.

Quellen von Kontextwerten

Kontextwerte können Ihrer AWS CDK App auf sechs verschiedene Arten zur Verfügung gestellt werden:

- Automatisch vom AWS Girokonto.
- Über die `--context` Option zum `cdk` Befehl. (Diese Werte sind immer Zeichenketten.)
- In der `cdk.context.json` Datei des Projekts.
- Im `context` Schlüssel der `cdk.json` Projektdatei.
- Im `context` Schlüssel Ihrer `~/cdk.json` Datei.
- In Ihrer AWS CDK App mit der `construct.node.setContext()` Methode.

In der Projektdatei `cdk.context.json` werden die aus Ihrem AWS Konto abgerufenen Kontextwerte AWS CDK zwischengespeichert. Durch diese Vorgehensweise werden unerwartete Änderungen an Ihren Bereitstellungen vermieden, wenn beispielsweise eine neue Availability Zone eingeführt wird. Der schreibt AWS CDK keine Kontextdaten in eine der anderen aufgelisteten Dateien.

Important

Weil sie Teil des Status Ihrer Anwendung sind `cdk.json` und zusammen mit dem Rest des Quellcodes Ihrer App der Quellcodeverwaltung unterliegen `cdk.context.json` müssen. Andernfalls können Bereitstellungen in anderen Umgebungen (z. B. einer CI-Pipeline) zu inkonsistenten Ergebnissen führen.

Kontextwerte beziehen sich auf das Konstrukt, mit dem sie erstellt wurden. Sie sind für untergeordnete Konstrukte sichtbar, nicht jedoch für Eltern oder Geschwister. Kontextwerte, die vom AWS CDK Toolkit (dem `cdk` Befehl) festgelegt werden, können automatisch, aus einer Datei oder aus der Option festgelegt werden. `--context` Kontextwerte aus diesen Quellen werden implizit für das Konstrukt App festgelegt. Daher sind sie für jedes Konstrukt in jedem Stapel in der App sichtbar.

Ihre App kann mithilfe der `construct.node.tryGetContext` Methode einen Kontextwert lesen. Wenn der angeforderte Eintrag im aktuellen Konstrukt oder einem seiner übergeordneten Objekte nicht gefunden wird, lautet das Ergebnis `undefined`. (Alternativ könnte das Ergebnis das Äquivalent Ihrer Sprache sein, z. B. `None` in Python.)

Context-Methoden

Das AWS CDK unterstützt mehrere Kontextmethoden, mit denen AWS CDK Apps Kontextinformationen aus der AWS Umgebung abrufen können. Mit der Methode [Stack.AvailabilityZones](#) können Sie beispielsweise eine Liste der Availability Zones abrufen, die in einem bestimmten AWS Konto und einer bestimmten Region verfügbar sind.

Im Folgenden sind die Kontextmethoden aufgeführt:

[HostedZone.fromLookup](#)

Ruft die gehosteten Zonen in Ihrem Konto ab.

[stack.AvailabilityZones](#)

Ruft die unterstützten Availability Zones ab.

[StringParameter.valueFromLookup](#)

Ruft einen Wert aus dem Amazon EC2 Systems Manager Manager-Parameterspeicher der aktuellen Region ab.

[vpc.FromLookup](#)

Ruft die vorhandenen Amazon Virtual Private Clouds in Ihren Konten ab.

[LookupMachineImage](#)

Sucht nach einem Maschinen-Image zur Verwendung mit einer NAT-Instance in einer Amazon Virtual Private Cloud.

Wenn ein erforderlicher Kontextwert nicht verfügbar ist, benachrichtigt die AWS CDK App das CDK Toolkit, dass die Kontextinformationen fehlen. Als Nächstes fragt die CLI das aktuelle AWS Konto nach den Informationen ab und speichert die resultierenden Kontextinformationen in der `cdk.context.json` Datei. Anschließend führt sie die AWS CDK App erneut mit den Kontextwerten aus.

Kontext anzeigen und verwalten

Verwenden Sie den `cdk context` Befehl, um die Informationen in Ihrer `cdk.context.json` Datei anzuzeigen und zu verwalten. Um diese Informationen zu sehen, verwenden Sie den `cdk context` Befehl ohne Optionen. Die Ausgabe sollte etwa wie folgt aussehen.

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`.

Um einen Kontextwert zu entfernen `cdk context --reset`, führen Sie den Befehl aus und geben Sie dabei den entsprechenden Schlüssel oder die Zahl des Werts an. Im folgenden Beispiel wird der Wert entfernt, der dem zweiten Schlüssel im vorherigen Beispiel entspricht. Dieser Wert stellt die Liste der Availability Zones in der Region Europa (Irland) dar.

```
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.
```

Wenn Sie also auf die neueste Version des Amazon Linux AMI aktualisieren möchten, verwenden Sie das vorherige Beispiel, um eine kontrollierte Aktualisierung des Kontextwerts durchzuführen und ihn zurückzusetzen. Synthetisieren Sie dann Ihre App und stellen Sie sie erneut bereit.

```
cdk synth
```

Um alle gespeicherten Kontextwerte für Ihre App zu löschen `cdk context --clear`, führen Sie Folgendes aus.

```
cdk context --clear
```

Nur in gespeicherte Kontextwerte `cdk.context.json` können zurückgesetzt oder gelöscht werden. Das AWS CDK berührt keine anderen Kontextwerte. Um zu verhindern, dass ein Kontextwert mithilfe dieser Befehle zurückgesetzt wird, können Sie den Wert daher nach kopieren `cdk.json`.

AWS CDK Toolkit-Flagge `--context`

Verwenden Sie die Option `--context` (kurz), um während der Synthese oder Bereitstellung Werte `-c` für den Laufzeitkontext an Ihre CDK-App zu übergeben.

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

Um mehrere Kontextwerte anzugeben, wiederholen Sie die `--context` Option beliebig oft und geben Sie jedes Mal ein Schlüssel-Wert-Paar an.

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

Bei der Synthese mehrerer Stapel werden die angegebenen Kontextwerte an alle Stapel übergeben. Um einzelnen Stacks unterschiedliche Kontextwerte zur Verfügung zu stellen, verwenden Sie entweder unterschiedliche Schlüssel für die Werte oder mehrere OR-Befehle. `cdk synth cdk deploy`

Von der Befehlszeile übergebene Kontextwerte sind immer Zeichenketten. Wenn ein Wert normalerweise von einem anderen Typ ist, muss Ihr Code darauf vorbereitet sein, den Wert zu konvertieren oder zu analysieren. Möglicherweise werden Kontextwerte, die keine Zeichenfolge sind, auf andere Weise bereitgestellt (z. B. `incdk.context.json`). Um sicherzustellen, dass diese Art von Wert wie erwartet funktioniert, stellen Sie vor der Konvertierung sicher, dass es sich bei dem Wert um eine Zeichenfolge handelt.

Beispiel

Im Folgenden finden Sie ein Beispiel für die Verwendung einer vorhandenen Amazon VPC mithilfe von AWS CDK Kontext.

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__(self, 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()
        });
    }
}

```

Sie können `cdk diff` verwenden, um die Auswirkungen der Übergabe eines Kontextwerts in der Befehlszeile zu sehen:

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

```
Stack ExistsvpcStack
```

```
Outputs
```

```
[+] Output publicsubnets publicsubnets:
```

```
{"Value":"subnet-06e0ea7dd302d3e8f,subnet-01fc0acfb58f3128f"}
```

Die resultierenden Kontextwerte können wie hier gezeigt angezeigt werden.

```
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-0cdded5da53180ebfa"
    ],
    "privateSubnetNames": [
      "Private"
    ],
    "privateSubnetRouteTableIds": [
      "rtb-0e955393ced0ada04",
      "rtb-05602e7b9f310e5b0"
    ],
    "publicSubnetIds": [
      "subnet-06e0ea7dd302d3e8f",
      "subnet-01fc0acfb58f3128f"
    ],
    "publicSubnetNames": [
      "Public"
    ],
    "publicSubnetRouteTableIds": [
      "rtb-00d1fdfd823c82289",
      "rtb-04bb1969b42969bcb"
    ]
  }
}
```

Feature-Flags

Die AWS CDK verwendet Feature-Flags, um potenziell störendes Verhalten in einer Version zu ermöglichen. Flags werden als [the section called “Kontext”](#) Werte in `cdk.json` (oder `~/cdk.json`) gespeichert. Sie werden nicht von den `cdk context --clear` Befehlen `cdk context --reset` oder entfernt.

Feature-Flags sind standardmäßig deaktiviert. Bestehende Projekte, die das -Flag nicht angeben, funktionieren mit späteren AWS CDK Versionen weiterhin wie zuvor. Neue Projekte, die mit `cdk init` erstellt wurden, enthalten Flags, die alle Funktionen aktivieren, die in der Version verfügbar sind, die das Projekt erstellt hat. Bearbeiten Sie `cdk.json` um alle Flags zu deaktivieren, für die Sie das frühere Verhalten bevorzugen. Sie können auch Flags hinzufügen, um nach dem Upgrade der neue Verhaltensweisen zu aktivieren AWS CDK.

Eine Liste aller aktuellen Feature-Flags finden Sie im AWS CDK GitHub Repository in [FEATURE_FLAGS.md](#). Eine Beschreibung aller neuen Feature-Flags, die in dieser Version hinzugefügt wurden, finden Sie unter CHANGELOG in einer bestimmten Version.

Zurücksetzen auf das Verhalten v1

In CDK v2 wurden die Standardwerte für einige Feature-Flags in Bezug auf v1 geändert. Sie können diese auf `false` setzen, um zu einem bestimmten AWS CDK v1-Verhalten zurückzukehren. Verwenden Sie den `cdk diff` Befehl, um die Änderungen an Ihrer synthetisierten Vorlage zu überprüfen, um festzustellen, ob eines dieser Flags erforderlich ist.

`@aws-cdk/core:newStyleStackSynthesis`

Verwenden Sie die neue Stack-Synthetics-Methode, die Bootstrap-Ressourcen mit bekannten Namen annimmt. Erfordert [ein modernes Bootstrapping](#), erlaubt jedoch CI/CD über [CDK-Pipelines](#) und kontoübergreifende Bereitstellungen sofort.

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

Wenn Ihre Anwendung mehrere Amazon API Gateway-API-Schlüssel verwendet und sie Nutzungsplänen zuordnet.

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

Wenn Ihre Anwendung Amazon-RDS-Datenbank-Instances oder -Datenbank-Cluster verwendet und explizit die ID für diese angibt.

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

Wenn Ihre Anwendung die Sicherheitsrichtlinie TLS_V1_2_2019 mit - Amazon CloudFront Verteilungen verwendet. CDK v2 verwendet standardmäßig die Sicherheitsrichtlinie TLSv1.2_2021.

@aws-cdk/core:stackRelativeExports

Wenn Ihre Anwendung mehrere Stacks verwendet und Sie auf Ressourcen aus einem Stack in einem anderen verweisen, bestimmt dies, ob ein absoluter oder relativer Pfad zum Erstellen von AWS CloudFormation Exporten verwendet wird.

@aws-cdk/aws-lambda:recognizeVersionProps

Wenn auf festgelegt `false`, enthält das CDK Metadaten, wenn erkannt wird, ob sich eine Lambda-Funktion geändert hat. Dies kann zu Bereitstellungsfehlern führen, wenn sich nur die Metadaten geändert haben, da doppelte Versionen nicht zulässig sind. Sie müssen dieses Flag nicht zurücksetzen, wenn Sie mindestens eine Änderung an allen Lambda-Funktionen in Ihrer Anwendung vorgenommen haben.

Die Syntax zum Zurücksetzen dieser Flags in `cdk.json` wird hier gezeigt.

```
{
  "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
  }
}
```

Aspekte

Aspekte sind eine Möglichkeit, eine Operation auf alle Konstrukte in einem bestimmten Bereich anzuwenden. Der -Aspekt könnte die Konstrukte ändern, z. B. durch Hinzufügen von Tags. Oder es könnte etwas über den Status der Konstrukte überprüfen, z. B. um sicherzustellen, dass alle Buckets verschlüsselt sind.

Um einen Aspekt auf ein Konstrukt und alle Konstrukte im selben Bereich anzuwenden, rufen Sie [Aspects.of\(SCOPE\).add\(\)](#) mit einem neuen Aspekt auf, wie im folgenden Beispiel gezeigt.

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(...))
```

Die AWS CDK verwendet Aspekte, um [Ressourcen zu markieren](#), aber das Framework kann auch für andere Zwecke verwendet werden. Sie können es beispielsweise verwenden, um die AWS CloudFormation Ressourcen zu validieren oder zu ändern, die von übergeordneten Konstrukten für Sie definiert werden.

Detailaspekte

Aspekte verwenden das [Besuchermuster](#). Ein Aspekt ist eine Klasse, die die folgende Schnittstelle implementiert.

TypeScript

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

JavaScript

JavaScript hat keine Schnittstellen als Sprachfunktion. Daher ist ein -Aspekt einfach eine Instance einer Klasse mit einer `visit` Methode, die den Knoten akzeptiert, auf dem betrieben werden soll.

Python

Python hat keine Schnittstellen als Sprachfunktion. Daher ist ein -Aspekt einfach eine Instance einer Klasse mit einer `visit` Methode, die den Knoten akzeptiert, auf dem betrieben werden soll.

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)  
}
```

Wenn Sie aufrufen `Aspects.of(SCOPE).add(...)`, fügt das Konstrukt den -Aspekt zu einer internen Liste von -Aspekten hinzu. Sie können die Liste mit `Aspects.of(SCOPE)` abrufen.

Während der [Vorbereitungsphase](#) AWS CDK ruft die die `visit` Methode des Objekts für das Konstrukt und jedes seiner untergeordneten Elemente in der Reihenfolge von oben nach unten auf.

Die `visit` Methode ist frei, alles im Konstrukt zu ändern. Konvertieren Sie in stark typisierten Sprachen das empfangene Konstrukt in einen spezifischeren Typ, bevor Sie auf konstruktsspezifische Eigenschaften oder Methoden zugreifen.

Aspekte werden nicht über StageKonstruktgrenzen weitergegeben, da sie nach der Definition eigenständig und unveränderlich Stages sind. Wenden Sie Aspekte auf das StageKonstrukt selbst (oder niedriger) an, wenn Sie möchten, dass sie Konstrukte innerhalb der `besuchenStage`.

Beispiel

Im folgenden Beispiel wird überprüft, ob für alle im Stack erstellten Buckets das Versioning aktiviert ist. Der -Aspekt fügt den Konstrukten, die die Validierung nicht bestehen, eine Fehleranmerkung hinzu. Dies führt dazu, dass der `synth` Vorgang fehlschlägt und die Bereitstellung der resultierenden Cloud-Baugruppe verhindert.

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());

```

Erste Schritte mit der AWS CDK

Beginnen Sie mit der , AWS Cloud Development Kit (AWS CDK) indem Sie die AWS CDK CLI installieren und Ihre erste CDK-App erstellen.

Themen

- [Voraussetzungen](#)
- [Schritt 1: Erstellen eines AWS-Konto](#)
- [Schritt 2: Programmgesteuerten Zugriff konfigurieren](#)
- [Schritt 3: Installieren der AWS CDKCLI](#)
- [Schritt 4: Bootstrappen Ihrer Umgebung](#)
- [Optionale AWS CDK Tools](#)
- [Nächste Schritte](#)
- [Weitere Informationen](#)
- [Deine erste AWS CDK App](#)

Voraussetzungen

Empfohlene Ressourcen

Bevor Sie mit der beginnen AWS CDK, empfehlen wir ein grundlegendes Verständnis der folgenden Punkte:

- Eine Einführung in die AWS CDK. Weitere Informationen hierzu finden Sie unter [Was ist der AWS CDK?](#).
- Kernkonzepte hinter dem AWS CDK. Weitere Informationen hierzu finden Sie unter [Lernen Sie die AWS CDK Kernkonzepte kennen](#).
- Die AWS-Services , die Sie mit der verwalten möchten AWS CDK.
- AWS Identity and Access Management. Weitere Informationen finden Sie unter [Was ist IAM?](#) und [Was ist IAM Identity Center?](#)
- AWS CloudFormation da den - AWS CloudFormation Service zur Bereitstellung von Ressourcen AWS CDK verwendet, die im CDK erstellt wurden. Weitere Informationen finden Sie unter [Was ist AWS CloudFormation?](#)

- Die unterstützte Programmiersprache, die Sie mit dem verwenden möchten AWS CDK.

Vorbereiten Ihrer lokalen Umgebung

Alle AWS CDK Entwickler benötigen unabhängig von Ihrer bevorzugten Sprache [Node.js](#) 14.15.0 oder höher. Alle unterstützten Programmiersprachen verwenden dasselbe Backend, das auf ausgeführt wird Node.js. Wir empfehlen eine [-Version mit aktiver langfristiger Unterstützung](#). Ihre Organisation hat möglicherweise eine andere Empfehlung.

Important

Die Node.js-Versionen 13.0.0 bis 13.6.0 sind AWS CDK aufgrund von Kompatibilitätsproblemen mit ihren Abhängigkeiten nicht mit dem kompatibel.

Andere Voraussetzungen hängen von der Sprache ab, in der Sie AWS CDK Anwendungen entwickeln, und lauten wie folgt.

TypeScript

- TypeScript 3.8 oder höher (`npm -g install typescript`)

JavaScript

Keine zusätzlichen Anforderungen

Python

- Python 3.7 oder höher, einschließlich `pip` und `virtualenv`

Java

- Java Development Kit (JDK) 8 (a.k.a. 1.8) oder höher
- Apache Maven 3.5 oder höher

Java IDE empfohlen (in einigen Beispielen in diesem Handbuch verwenden wir Eclipse).

Die IDE muss Maven-Projekte importieren können. Überprüfen Sie, ob Ihr Projekt für die Verwendung von Java 1.8 festgelegt ist. Legen Sie die Umgebungsvariable `JAVA_HOME` auf den Pfad fest, in dem Sie das JDK installiert haben.

C#

.NET Core 3.1 oder höher oder .NET 6.0 oder höher.

Visual Studio 2019 (jede Edition) oder Visual Studio Code empfohlen.

Go

Go 1.1.8 oder höher.

Ausführlichere Informationen finden Sie im Abschnitt Voraussetzungen für Ihre Sprache:

- [the section called “In TypeScript”](#)
- [the section called “In JavaScript”](#)
- [the section called “In Python”](#)
- [the section called “In Java”](#)
- [the section called “In C#”](#)
- [the section called “In Go”](#)

Sprachveralterung von Drittanbietern

Jede Sprachversion wird nur unterstützt, bis sie EOL (End of Life) ist, und kann sich ohne vorherige Ankündigung ändern.

Schritt 1: Erstellen eines AWS-Konto

Wenn Sie noch nicht mit vertraut sind AWS, müssen Sie sich für ein registrieren AWS-Konto und einen Administratorbenutzer erstellen. Weitere Informationen finden Sie unter [Einrichtung von IAM](#) im IAM-Benutzerhandbuch.

Wenn Sie mit interagieren AWS, geben Sie Ihre AWS Sicherheitsanmeldeinformationen an, um zu überprüfen, wer Sie sind und ob Sie die Berechtigung zum Zugriff auf die Ressourcen haben, die Sie anfordern. AWS verwendet die Sicherheitsanmeldeinformationen, um Ihre Anforderungen zu authentifizieren und zu autorisieren. Weitere Informationen finden Sie unter [-AWS Sicherheitsanmeldeinformationen](#) im IAM-Benutzerhandbuch.

Schritt 2: Programmgesteuerten Zugriff konfigurieren

Bei der Entwicklung mit der AWS CDK in Ihrer lokalen Umgebung verlassen Sie sich auf die AWS CDK CLI , um mit Ihren - AWS Ressourcen zu interagieren AWS-Services und diese zu verwalten. Um die verwenden zu können AWS CDK CLI, müssen Sie den programmgesteuerten

Zugriff konfigurieren. Weitere Informationen zu den verschiedenen Möglichkeiten, wie Sie den programmgesteuerten Zugriff konfigurieren können, finden Sie unter [Authentifizierung und Zugriff](#) im AWS Referenzhandbuch zu -SDKs und Tools.

Für neue Benutzer, denen von ihrem Arbeitgeber keine Authentifizierungsmethode zugewiesen wurde, empfehlen wir die Verwendung von AWS IAM Identity Center. Diese Methode umfasst die Installation der AWS Command Line Interface (AWS CLI) und deren Verwendung für die Konfiguration und Anmeldung beim - AWS Zugriffsportal. Informationen zum Konfigurieren des programmgesteuerten Zugriffs mit IAM Identity Center finden Sie unter [IAM-Identity-Center-Authentifizierung](#) im AWS Referenzhandbuch zu -SDKs und Tools. Nach Abschluss sollte Ihre Umgebung die folgenden Elemente enthalten:

- Die AWS CLI, mit der Sie eine AWS -Zugriffsportalsitzung starten, bevor Sie Ihre Anwendung ausführen.
- Eine [AWSconfigfreigegebene Datei](#) mit einem [default] Profil mit einer Reihe von Konfigurationswerten, auf die über die verwiesen werden kann AWS CDK. Den Speicherort dieser Datei finden Sie unter [Speicherort der freigegebenen Dateien](#) im Referenzhandbuch für AWS SDKs und Tools.
- Die freigegebene config Datei legt die [region](#) Einstellung fest. Dadurch wird der Standardwert festgelegt, den AWS-Region die für - AWS Anforderungen AWS CDK verwendet.
- Die AWS CDK verwendet die [SSO-Token-Anbieterkonfiguration](#) des Profils, um Anmeldeinformationen zu erhalten, bevor Anfragen an gesendet werden AWS. Der `sso_role_name` Wert, bei dem es sich um eine IAM-Rolle handelt, die mit einem IAM-Identity-Center-Berechtigungssatz verbunden ist, sollte den Zugriff auf die in Ihrer Anwendung AWS-Services verwendeten ermöglichen.

Die folgende config Beispieldatei zeigt ein Standardprofil, das mit der Konfiguration des SSO-Token-Anbieters eingerichtet wurde. Die `sso_session` Einstellung des Profils bezieht sich auf den benannten [sso-session Abschnitt](#). Der `sso-session` Abschnitt enthält Einstellungen zum Initiieren einer AWS -Zugriffsportalsitzung.

```
[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
```

Starten einer - AWS Zugriffsportalsitzung

Bevor Sie auf zugreifen AWS-Services können, benötigen Sie eine aktive - AWS Zugriffsportalsitzung für die , AWS CDK um die IAM-Identity-Center-Authentifizierung zum Auflösen von Anmeldeinformationen verwenden zu können. Abhängig von Ihren konfigurierten Sitzungslängen läuft Ihr Zugriff schließlich ab und bei AWS CDK tritt ein Authentifizierungsfehler auf. Führen Sie den folgenden Befehl in der aus AWS CLI , um sich beim - AWS Zugriffsportal anzumelden.

```
aws sso login
```

Wenn Ihre Konfiguration des SSO-Token-Anbieters ein benanntes Profil anstelle des Standardprofils verwendet, lautet der Befehl `aws sso login --profile NAME`. Geben Sie dieses Profil auch an, wenn Sie `cdk` Befehle mit der `---profileOption` oder der `-AWS_PROFILE` Umgebungsvariablen ausgeben.

Führen Sie den folgenden AWS CLI Befehl aus, um zu testen, ob Sie bereits über eine aktive Sitzung verfügen.

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

In der Antwort auf diesen Befehl sollten das in der freigegebenen `config`-Datei konfigurierte IAM-Identity-Center-Konto und der Berechtigungssatz angegeben werden.

Note

Wenn Sie bereits über eine aktive - AWS Zugriffsportalsitzung verfügen und ausführen `aws sso login`, müssen Sie keine Anmeldeinformationen angeben. Während des Anmeldevorgangs werden Sie möglicherweise aufgefordert, den AWS CLI Zugriff auf Ihre Daten zu erlauben. Da die auf dem SDK für Python AWS CLI aufbaut, können Berechtigungsnachrichten Variationen des `botocore` Namens enthalten.

Schritt 3: Installieren der AWS CDKCLI

Installieren Sie den AWS CDK CLI global mit dem folgenden Node Package Manager-Befehl.

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

Note

Wenn Sie einen Berechtigungsfehler erhalten und Administratorzugriff auf Ihr System haben, versuchen Sie es mit `sudo npm install -g aws-cdk`.

Führen Sie den folgenden Befehl aus, um eine erfolgreiche Installation zu überprüfen. Der AWS CDK CLI sollte die Versionsnummer ausgeben:

```
cdk --version
```

Wenn Sie eine Fehlermeldung erhalten, versuchen Sie, zu AWS CDK CLI deinstallieren, indem Sie Folgendes ausführen:

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

Wiederholen Sie dann die Schritte, um die neu zu installieren AWS CDK CLI.

Wenn Sie immer noch eine Fehlermeldung erhalten, löschen Sie den `node_modules` Ordner aus dem aktuellen Projekt und auch aus dem globalen `node_modules` Ordner. Um diesen Ordner zu finden, führen Sie `ausnpm config get prefix`.

Der AWS CDK CLI erhält Sicherheitsanmeldeinformationen von Quellen, die Sie in den vorherigen Schritten konfiguriert haben.

Note

CDK Toolkit v2 funktioniert mit vorhandenen CDK-v1-Projekten. Es kann jedoch keine neuen CDK-v1-Projekte initialisieren. Überprüfen Sie, [the section called "Neue Voraussetzungen"](#) ob Sie dies tun müssen.

Schritt 4: Bootstrappen Ihrer Umgebung

Jede AWS [Umgebung](#), in der Sie Ressourcen bereitstellen möchten, muss [bootstrappen](#).

Führen Sie zum Bootstrappen Folgendes aus:

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

Tip

Wenn Sie Ihre AWS Kontonummer nicht zur Hand haben, erhalten Sie sie über die AWS Management Console. Oder, wenn Sie die AWS CLI installiert haben, zeigt der folgende Befehl Ihre Standardkontoinformationen an, einschließlich der Kontonummer.

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

Wenn Sie benannte Profile in Ihrer lokalen AWS Konfiguration erstellt haben, können Sie die `--profile` Option verwenden, um die Kontoinformationen für ein bestimmtes Profil anzuzeigen. Das folgende Beispiel zeigt, wie Kontoinformationen für das `prod`-Profil angezeigt werden.

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

Um die Standardregion anzuzeigen, verwenden Sie `aws configure get`.

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

Optionale AWS CDK Tools

Das [AWS Toolkit for Visual Studio Code](#) ist ein Open-Source-Plug-In für Visual Studio Code, mit dem Sie Anwendungen auf erstellen, debuggen und bereitstellen können AWS. Das Toolkit bietet eine integrierte Erfahrung für die Entwicklung von AWS CDK Anwendungen. Sie enthält die AWS CDK-Explorer-Funktion, um Ihre AWS CDK Projekte aufzulisten und die verschiedenen Komponenten der CDK-Anwendung zu durchsuchen. [Installieren Sie das -Plug-In](#) und erfahren Sie mehr über die [Verwendung von AWS CDK Explorer](#).

Nächste Schritte

Nachdem Sie die installiert haben AWS CDK CLI, verwenden Sie sie, um [Ihre erste AWS CDK App zu erstellen](#).

Weitere Informationen zur Verwendung der AWS CDK in Ihrer bevorzugten Programmiersprache finden Sie unter [Arbeiten mit AWS CDK in unterstützten Programmiersprachen](#).

ist AWS CDK ein Open-Source-Projekt. Informationen zum Beitragen finden [Sie unter Beitragen zum AWS Cloud Development Kit \(AWS CDK\)](#).

Weitere Informationen

Weitere Informationen über finden AWS CDK Sie hier:

- [CDK-Workshop](#) – Detaillierter praxisbezogener Workshop.
- [API-Referenz](#) – Erkunden Sie Konstrukte, die für die verfügbar sind AWS-Services , die Sie verwenden werden.
- [Construct Hub](#) – Finden Sie Konstrukte aus der CDK-Community.
- [-AWS CDK Beispiele](#) – Erkunden Sie Codebeispiele von - AWS CDK Projekten.

Deine erste AWS CDK App

Beginnen Sie mit der Verwendung von, AWS Cloud Development Kit (AWS CDK) indem Sie Ihre erste CDK-App erstellen.

Bevor Sie mit diesem Tutorial beginnen, empfehlen wir Ihnen, die folgenden Schritte durchzuführen:

- Eine Einführung in die finden [Was ist der AWS CDK?](#) Sie unter AWS CDK.
- Weitere Informationen [Lernen Sie die AWS CDK Kernkonzepte kennen](#) zu den Kernkonzepten von finden Sie unter AWS CDK.
- Die Voraussetzungen und AWS CDK Einrichtungsschritte finden Sie unter [Erste Schritte mit der AWS CDK](#).

Themen

- [Über dieses Tutorial](#)

- [Schritt 1: Erstellen Sie die App](#)
- [Schritt 2: Erstellen Sie die App](#)
- [Schritt 3: Listet die Stapel in der App auf](#)
- [Schritt 4: Einen Amazon S3 S3-Bucket hinzufügen](#)
- [Schritt 5: Synthetisieren Sie eine Vorlage AWS CloudFormation](#)
- [Schritt 6: Stellen Sie Ihren Stack bereit](#)
- [Schritt 7: Ändern Sie Ihre App](#)
- [Schritt 8: Zerstören der Ressourcen der App](#)
- [Nächste Schritte](#)

Über dieses Tutorial

In diesem Tutorial werden Sie eine einfache AWS CDK App erstellen und bereitstellen. Diese App enthält einen Stapel mit einer einzigen Bucket-Ressource von Amazon Simple Storage Service (Amazon S3). In diesem Tutorial lernen Sie Folgendes:

- Die Struktur eines AWS CDK Projekts.
- So erstellen Sie eine AWS CDK App.
- So verwenden Sie die AWS Construct-Bibliothek, um Apps, Stacks und AWS Ressourcen zu definieren.
- So verwenden Sie das CDK, CLI um Ihre CDK-App zu synthetisieren, zu vergleichen, bereitzustellen und zu löschen.
- So modifizieren und implementieren Sie Ihre CDK-App erneut, um Ihre bereitgestellten Ressourcen zu aktualisieren.

Der AWS CDK Standard-Entwicklungsablauf besteht aus den folgenden Schritten:

1. Erstellen Sie Ihre AWS CDK App — Hier verwenden Sie eine vom CDK CLI bereitgestellte Vorlage.
2. Definieren Sie Ihre Stacks und Ressourcen — Verwenden Sie Konstrukte, um Ihre Stacks und AWS Ressourcen in Ihrer App zu definieren.
3. Erstellen Sie Ihre App — Dieser Schritt ist optional. Das CDK führt diesen Schritt bei Bedarf CLI automatisch durch. Es wird empfohlen, diesen Schritt durchzuführen, um Syntax- und Typfehler zu identifizieren.

4. Synthetisieren Sie Ihre Stapel — In diesem Schritt wird eine AWS CloudFormation Vorlage für jeden Stapel in Ihrer App erstellt. Dieser Schritt ist nützlich, um logische Fehler in Ihren definierten AWS Ressourcen zu identifizieren.
5. Stellen Sie Ihre App bereit — Stellen Sie sie in Ihrer AWS Umgebung bereit AWS CloudFormation , um Ihre Ressourcen bereitzustellen. Während der Bereitstellung werden Sie alle Berechtigungsprobleme mit Ihrer App identifizieren.

In einem typischen Arbeitsablauf gehen Sie zurück und wiederholen die vorherigen Schritte, um Ihre App zu ändern oder zu debuggen.

Wir empfehlen Ihnen, die Versionskontrolle für Ihre AWS CDK Projekte zu verwenden.

Schritt 1: Erstellen Sie die App

Eine CDK-App sollte sich in einem eigenen Verzeichnis mit eigenen lokalen Modulabhängigkeiten befinden. Erstellen Sie auf Ihrem Entwicklungscomputer ein neues Verzeichnis. Im Folgenden finden Sie ein Beispiel, das ein neues `hello-cdk` Verzeichnis erstellt:

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

Important

Achten Sie darauf, Ihr Projektverzeichnis genau so zu benennen `hello-cdk`, wie hier gezeigt. Die AWS CDK Projektvorlage verwendet den Verzeichnisnamen, um Dinge im generierten Code zu benennen. Wenn Sie einen anderen Namen verwenden, funktioniert der Code in diesem Tutorial nicht.

Initialisieren Sie als Nächstes die App in Ihrem neuen Verzeichnis mit dem `cdk init` Befehl. Geben Sie mit der `--language` Option die app Vorlage und Ihre bevorzugte Programmiersprache an. Im Folgenden wird ein Beispiel gezeigt:

TypeScript

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


JavaScript

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

Python

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

Nachdem die App erstellt wurde, geben Sie auch die folgenden beiden Befehle ein. Diese aktivieren die virtuelle Python-Umgebung der App und installieren die AWS CDK Kernabhängigkeiten.

```
$ source .venv/bin/activate # On Windows, run `.\venv\Scripts\activate` instead  
$ python -m pip install -r requirements.txt
```

Java

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

Wenn Sie eine IDE verwenden, können Sie das Projekt jetzt öffnen oder importieren. Wählen Sie in Eclipse beispielsweise „Datei“ > „Importieren“ > „Maven“ > „Bestehende Maven-Projekte“. Stellen Sie sicher, dass die Projekteinstellungen so eingestellt sind, dass sie Java 8 (1.8) verwenden.

C#

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

Wenn Sie Visual Studio verwenden, öffnen Sie die Lösungsdatei im `src` Verzeichnis.

Go

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

Geben Sie nach der Erstellung der App auch den folgenden Befehl ein, um die AWS Construct Library-Module zu installieren, die die App benötigt.

```
$ go get
```

Der `cdk init` Befehl erstellt eine Reihe von Dateien und Ordnern innerhalb des `hello-cdk` Verzeichnisses, um Ihnen bei der Organisation des Quellcodes für Ihre AWS CDK App zu helfen. Zusammengenommen wird dies als Ihr AWS CDK Projekt bezeichnet. Nehmen Sie sich einen Moment Zeit, um das CDK-Projekt zu erkunden.

Wenn Sie es Git installiert haben, wird `cdk init` jedes Projekt, mit dem Sie erstellen, auch als Git Repository initialisiert.

Schritt 2: Erstellen Sie die App

In den meisten Programmierumgebungen erstellen oder kompilieren Sie Code, nachdem Sie Änderungen vorgenommen haben. Dies ist bei der nicht erforderlich, AWS CDK da das CDK diesen Schritt CLI automatisch durchführt. Sie können jedoch immer noch manuell bauen, wenn Sie Syntax- und Typfehler abfangen möchten. Im Folgenden wird ein Beispiel gezeigt:

TypeScript

```
$ npm run build
```

JavaScript

Es ist kein Build-Schritt erforderlich.

Python

Es ist kein Build-Schritt erforderlich.

Java

```
$ mvn compile -q
```

Oder drücken Sie `Control-B` in Eclipse (andere Java-IDEs können variieren)

C#

```
$ dotnet build src
```

Oder drücken Sie `F6` in Visual Studio

Go

```
$ go build
```

Schritt 3: Listet die Stapel in der App auf

Stellen Sie sicher, dass Ihre App korrekt erstellt wurde, indem Sie die Stacks in Ihrer App auflisten. Führen Sie Folgendes aus:

```
$ cdk ls
```

Die Ausgabe sollte angezeigt `HelloCdkStack` werden. Wenn Sie diese Ausgabe nicht sehen, stellen Sie sicher, dass Sie sich im richtigen Arbeitsverzeichnis Ihres Projekts befinden, und versuchen Sie es erneut. Wenn Sie Ihren Stack immer noch nicht sehen, wiederholen Sie den Vorgang [the section called “Schritt 1: Erstellen Sie die App”](#) und versuchen Sie es erneut.

Schritt 4: Einen Amazon S3 S3-Bucket hinzufügen

Zu diesem Zeitpunkt enthält Ihre CDK-App einen einzigen Stack. Als Nächstes definieren Sie eine Amazon Simple Storage Service (Amazon S3) -Bucket-Ressource in Ihrem Stack. Dazu importieren und verwenden Sie das [Bucket](#) L2-Konstrukt aus der AWS Construct-Bibliothek.

Ändern Sie Ihre CDK-App, indem Sie das `Bucket` Konstrukt importieren und Ihre Amazon S3 S3-Bucket-Ressource definieren. Im Folgenden wird ein Beispiel gezeigt:

TypeScript

In `lib/hello-cdk-stack.ts`:

```
import * as cdk from 'aws-cdk-lib';
import { aws_s3 as s3 } from 'aws-cdk-lib';

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

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

JavaScript

In `lib/hello-cdk-stack.js`:

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

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

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

module.exports = { HelloCdkStack }
```

Python

In `hello_cdk/hello_cdk_stack.py`:

```
from aws_cdk import (
    Stack,
    aws_s3 as s3,
)
from constructs import Construct

class HelloCdkStack(Stack):

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

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

Java

In `src/main/java/com/myorg/HelloCdkStack.java`:

```
package com.myorg;

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

public class HelloCdkStack extends Stack {
    public HelloCdkStack(final App scope, final String id) {
```

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

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

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

C#

In `src/HelloCdk/HelloCdkStack.cs`:

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

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

Go

In `hello-cdk.go`:

```
package main

import (
    "github.com/aws/aws-cdk-go/awscdk/v2"
    "github.com/aws/aws-cdk-go/awscdk/v2/awss3"
    "github.com/aws/constructs-go/constructs/v10"
```

```
"github.com/aws/jsii-runtime-go"
)

type HelloCdkStackProps struct {
    awscdk.StackProps
}

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

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

    return stack
}

func main() {
    defer jsii.Close()

    app := awscdk.NewApp(nil)

    NewHelloCdkStack(app, "HelloCdkStack", &HelloCdkStackProps{
        awscdk.StackProps{
            Env: env(),
        },
    })

    app.Synth(nil)
}

func env() *awscdk.Environment {
    return nil
}
```

Schauen wir uns das Bucket Konstrukt genauer an. Wie alle Konstrukte benötigt die Bucket Klasse drei Parameter:

- `scope` — Definiert die Stack Klasse als übergeordnetes Objekt des Bucket Konstrukts. Alle Konstrukte, die AWS Ressourcen definieren, werden innerhalb des Gültigkeitsbereichs eines Stacks erstellt. Sie können Konstrukte innerhalb von Konstrukten definieren und so eine Hierarchie (Baum) erstellen. Hier, und in den meisten Fällen, ist der Gültigkeitsbereich `this` (`self` in Python).
- `ID` — Die logische ID von Bucket innerhalb Ihrer AWS CDK App. Diese ID sowie ein Hash, der auf der Position des Buckets innerhalb des Stacks basiert, identifizieren den Bucket während der Bereitstellung eindeutig. Der verweist AWS CDK auch auf diese ID, wenn Sie das Konstrukt in Ihrer App aktualisieren und erneut bereitstellen, um die bereitgestellte Ressource zu aktualisieren. Hier ist `MyFirstBucket` Ihre logische ID. Buckets können auch einen Namen haben, der mit der `bucketName` Eigenschaft angegeben wird. Dies unterscheidet sich von der logischen ID.
- `props` — Ein Bündel von Werten, die die Eigenschaften des Buckets definieren. Hier haben Sie die `versioned` Eigenschaft als `definierttrue`, was die Versionierung für die Dateien im Bucket ermöglicht.

Requisiten werden in den von der unterstützten Sprachen unterschiedlich dargestellt. AWS CDK

- In TypeScript und `props` steht für ein einzelnes Argument JavaScript, und Sie übergeben ein Objekt, das die gewünschten Eigenschaften enthält.
- In Python werden Requisiten als Schlüsselwortargumente übergeben.
- In Java wird ein Builder bereitgestellt, um die Requisiten zu übergeben. Es gibt zwei: einen dafür und einen `secondBucketProps`, Bucket damit Sie das Konstrukt und das zugehörige Requisitenobjekt in einem Schritt erstellen können. Dieser Code verwendet Letzteres.
- In C# instanziiieren Sie ein `BucketProps` Objekt mithilfe eines Objektinitialisierers und übergeben es als dritten Parameter.

Wenn die Requisiten eines Konstrukts optional sind, können Sie den Parameter komplett weglassen. `props`

Alle Konstrukte verwenden dieselben drei Argumente, sodass Sie beim Kennenlernen neuer Argumente leicht den Überblick behalten können. Und wie Sie vielleicht erwarten, können Sie jedes Konstrukt in Unterklassen unterteilen, um es an Ihre Bedürfnisse anzupassen oder wenn Sie die Standardwerte ändern möchten.

Schritt 5: Synthetisieren Sie eine Vorlage AWS CloudFormation

Synthetisieren Sie eine AWS CloudFormation Vorlage für die App wie folgt:

\$ cdk synth

Wenn Ihre App mehr als einen Stapel enthält, müssen Sie angeben, welche Stapel synthetisiert werden sollen. Da Ihre App einen einzelnen Stapel enthält, erkennt das CDK CLI automatisch den zu synthetisierenden Stapel.

Wenn Sie ihn nicht ausführen `cdk synth`, führt das CDK diesen Schritt bei der Bereitstellung CLI automatisch durch. Wir empfehlen jedoch, diesen Schritt vor jeder Bereitstellung auszuführen.

i Tip

Wenn Sie eine Fehlermeldung wie erhalten `--app is required ...`, überprüfen Sie das Verzeichnis, in dem Sie CLI CDK-Befehle ausführen. Sie sollten sich in Ihrem Haupt-App-Verzeichnis befinden.

Der `cdk synth` Befehl führt Ihre App aus. Dadurch wird eine AWS CloudFormation Vorlage für jeden Stapel in Ihrer App erstellt. Das CDK CLI zeigt eine YAML-formatierte Version Ihrer Vorlage in der Befehlszeile an und speichert eine JSON-formatierte Version Ihrer Vorlage im Verzeichnis `cdk.out`. Im Folgenden finden Sie einen Ausschnitt aus der Befehlszeilenausgabe, der zeigt, dass der Bucket in der Vorlage definiert ist: AWS CloudFormation

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

i Note

Jede generierte Vorlage enthält standardmäßig eine `AWS::CDK::Metadata` Ressource. Das AWS CDK Team verwendet diese Metadaten, um Einblicke in die AWS CDK Nutzung zu gewinnen und Möglichkeiten zu finden, diese zu verbessern. Einzelheiten, einschließlich der Deaktivierung der Versionsberichterstattung, finden Sie unter [Versionsberichterstattung](#).

Die generierte Vorlage kann über die AWS CloudFormation Konsole oder ein beliebiges AWS CloudFormation Bereitstellungstool bereitgestellt werden. Sie können das CDK auch für CLI die Bereitstellung verwenden. Im nächsten Schritt verwenden Sie das CDK CLI für die Bereitstellung.

Schritt 6: Stellen Sie Ihren Stack bereit

Um Ihren CDK-Stack AWS CloudFormation mithilfe des CDK bereitzustellen CLI, führen Sie Folgendes aus:

```
$ cdk deploy
```

Important

Sie müssen vor der Bereitstellung ein einmaliges Bootstrapping Ihrer AWS Umgebung durchführen. Anweisungen finden Sie unter [Bootstrap your environment](#).

Ähnlich wie müssen `cdk synth` Sie den AWS CDK Stack nicht angeben, da die App einen einzigen Stapel enthält.

Wenn Ihr Code Auswirkungen auf die Sicherheit hat, gibt das CDK CLI eine Zusammenfassung aus. Sie müssen sie bestätigen, um mit der Bereitstellung fortzufahren. Die App in diesem Tutorial hat diese Auswirkungen nicht.

Nach der Ausführung `cdk deploy` CLI zeigt das CDK Fortschrittsinformationen an, während Ihr Stack bereitgestellt wird. Wenn Sie fertig sind, können Sie zur [AWS CloudFormation Konsole](#) gehen, um Ihren `HelloCdkStack` Stack anzusehen. Sie können auch zur Amazon S3 S3-Konsole gehen, um Ihre `MyFirstBucket` Ressource anzusehen.

Herzlichen Glückwunsch! Sie haben Ihren ersten Stack mit dem bereitgestellt AWS CDK. Als Nächstes ändern Sie Ihre App und stellen sie erneut bereit, um Ihre Ressource zu aktualisieren.

Schritt 7: Ändern Sie Ihre App

In diesem Schritt ändern Sie Ihren Amazon S3 S3-Bucket, indem Sie ihn so konfigurieren, dass er automatisch gelöscht wird, wenn Ihr Stack gelöscht wird. Diese Änderung beinhaltet die Änderung der `RemovalPolicy` Eigenschaft des Buckets. Sie werden die `autoDeleteObjects` Eigenschaft auch so konfigurieren, dass das CDK CLI Objekte aus dem Bucket löscht, bevor er zerstört wird.

Standardmäßig werden Amazon S3 S3-Buckets, die Objekte enthalten, AWS CloudFormation nicht gelöscht.

Verwenden Sie das folgende Beispiel, um Ihre Ressource zu ändern:

TypeScript

Aktualisieren von `lib/hello-cdk-stack.ts`.

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

JavaScript

Aktualisieren von `lib/hello-cdk-stack.js`.

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

Python

Aktualisieren von `hello_cdk/hello_cdk_stack.py`.

```
from aws_cdk import (
    # ...
    RemovalPolicy,
)
# ...

class HelloCdkStack(Stack):

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

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

```
auto_delete_objects=True)
```

Java

Aktualisieren von `src/main/java/com/myorg/HelloCdkStack.java`.

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

C#

Aktualisieren von `src/HelloCdk/HelloCdkStack.cs`.

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

Go

Aktualisieren von `hello-cdk.go`.

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

Derzeit wurden durch Ihre Codeänderungen keine direkten Aktualisierungen Ihrer bereitgestellten Amazon S3 S3-Bucket-Ressource vorgenommen. Ihr Code definiert den gewünschten Status Ihrer Ressource. Um Ihre bereitgestellte Ressource zu ändern, verwenden Sie das CDK, CLI um den gewünschten Status in einer neuen AWS CloudFormation Vorlage zusammenzufassen. Anschließend stellen Sie Ihre neue AWS CloudFormation Vorlage als Änderungssatz bereit. Mit Änderungssätzen werden nur die Änderungen vorgenommen, die erforderlich sind, um den gewünschten neuen Status zu erreichen.

Verwenden Sie den `cdk diff` Befehl, um diese Änderungen zu sehen. Führen Sie Folgendes aus:

```
$ cdk diff
```

Das CDK CLI fragt Ihr AWS-Konto nach der neuesten AWS CloudFormation Vorlage für den `HelloCdkStack` Stack ab. Anschließend vergleicht es die neueste Vorlage mit der Vorlage, die sie gerade aus Ihrer App synthetisiert hat. Die Ausgabe sollte wie folgt aussehen.

```
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
```

Dieser Unterschied besteht aus vier Abschnitten:

- Änderungen der IAM-Erklärung und Änderungen der IAM-Richtlinien — Diese Berechtigungsänderungen sind vorhanden, weil Sie die `AutoDeleteObjects` Eigenschaft für Ihren Amazon S3 S3-Bucket festgelegt haben. Die Funktion zum automatischen Löschen verwendet eine benutzerdefinierte Ressource, um die Objekte im Bucket zu löschen, bevor

der Bucket selbst gelöscht wird. Die IAM-Objekte gewähren dem Code der benutzerdefinierten Ressource Zugriff auf den Bucket.

- **Parameter** — Der AWS CDK verwendet diese Einträge, um das AWS Lambda Funktions-Asset für die benutzerdefinierte Ressource zu finden.
- **Ressourcen** — Die neuen und geänderten Ressourcen in diesem Stapel. Wir können sehen, dass die zuvor erwähnten IAM-Objekte, die benutzerdefinierte Ressource und die zugehörige Lambda-Funktion hinzugefügt werden. Wir können auch sehen, dass der Bucket `DeletionPolicy` und die `UpdateReplacePolicy` Attribute aktualisiert werden. Diese ermöglichen es, den Bucket zusammen mit dem Stack zu löschen und durch einen neuen zu ersetzen.

Möglicherweise stellen Sie fest, dass wir `RemovalPolicy` in unserer AWS CDK App angegeben haben, aber in der resultierenden AWS CloudFormation Vorlage eine `DeletionPolicy` Eigenschaft erhalten haben. Dies liegt daran, dass der einen anderen Namen für die Eigenschaft AWS CDK verwendet. AWS CDK Standardmäßig wird der Bucket beibehalten, wenn der Stack gelöscht wird, während er AWS CloudFormation standardmäßig gelöscht wird. Weitere Informationen finden Sie unter [the section called “Richtlinien zur Entfernung”](#).

Um Ihre neue AWS CloudFormation Vorlage zu sehen, können Sie `aws cloudformation synth` ausführen. Wenn Sie einige Änderungen an Ihrer CDK-App vornehmen, enthält Ihre neue AWS CloudFormation Vorlage jetzt viele zusätzliche Codezeilen im Vergleich zur ursprünglichen AWS CloudFormation Vorlage.

Stellen Sie als Nächstes Ihre App bereit, indem Sie Folgendes ausführen:

```
$ cdk deploy
```

Darin AWS CDK werden Sie über die Änderungen der Sicherheitsrichtlinien informiert, die wir bereits im Vergleich gesehen haben. Geben Sie `y` ein, um die Änderungen zu genehmigen und den aktualisierten Stack bereitzustellen. Das CDK CLI stellt Ihren Stack bereit, um die gewünschten Änderungen vorzunehmen. Im Folgenden finden Sie ein Beispiel für eine Ausgabe:

```
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
```

Schritt 8: Zerstören der Ressourcen der App

Nachdem Sie dieses Tutorial abgeschlossen haben, können Sie den bereitgestellten AWS CloudFormation Stack und alle damit verbundenen Ressourcen löschen. Dies ist eine bewährte Methode, um unnötige Kosten zu minimieren und Ihre Umgebung sauber zu halten. Führen Sie Folgendes aus:

```
$ cdk destroy
```

Geben Sie ein, um die Änderungen zu genehmigen und Ihren Stack zu löschen.

Note

Wenn Sie die Buckets nicht ändern würden `RemovalPolicy`, würde das Löschen des Stacks erfolgreich abgeschlossen, aber der Bucket wäre verwaist (nicht mehr mit dem Stack verknüpft).

Nächste Schritte

Herzlichen Glückwunsch! Sie haben dieses Tutorial abgeschlossen und mit AWS CDK dem erfolgreich Ressourcen in der erstellt, geändert und gelöscht. AWS Cloud Sie sind jetzt bereit, mit der Verwendung von zu beginnen AWS CDK.

Weitere Informationen zur Verwendung von AWS CDK in Ihrer bevorzugten Programmiersprache finden Sie unter [Arbeiten mit AWS CDK in unterstützten Programmiersprachen](#).

Weitere Ressourcen finden Sie unter den folgenden Links:

- Probieren Sie den [CDK-Workshop](#) aus, um sich eingehender mit einem komplexeren Projekt vertraut zu machen.
- Tauchen Sie tiefer in Konzepte wie [the section called “Umgebungen”](#), [the section called “Objekte”](#), [the section called “Berechtigungen”](#), [the section called “Kontext”](#) [the section called “Parameter”](#), und [the section called “Anpassen von Konstrukten”](#) ein.
- Sehen Sie sich die [API-Referenz](#) an, um mit der Erkundung der CDK-Konstrukte zu beginnen, die für Ihre bevorzugten AWS Dienste verfügbar sind.

- Besuchen Sie [Construct Hub](#), um Konstrukte zu entdecken, die von und anderen erstellt AWS wurden.
- Entdecken Sie [Beispiele für](#) die AWS CDK Verwendung von.

Das AWS CDK ist ein Open-Source-Projekt. Informationen zum Beitrag finden Sie unter [Beitrag zum AWS Cloud Development Kit \(AWS CDK\)](#).

Arbeiten mit AWS CDK in unterstützten Programmiersprachen

Verwenden Sie die AWS Cloud Development Kit (AWS CDK) , um Ihre AWS Cloud Infrastruktur mit einer [unterstützten Programmiersprache zu](#) definieren.

Themen

- [Importieren der AWS Konstruktbibliothek](#)
- [Verwalten von Abhängigkeiten](#)
- [Vergleich von AWS CDK in TypeScript mit anderen Sprachen](#)
- [Arbeiten mit der AWS CDK in TypeScript](#)
- [Arbeiten mit der AWS CDK in JavaScript](#)
- [Arbeiten mit dem AWS CDK in Python](#)
- [Arbeiten mit in AWS CDK Java](#)
- [Arbeiten mit in AWS CDK C#](#)
- [Arbeiten mit in AWS CDK Go](#)

Importieren der AWS Konstruktbibliothek

Die AWS CDK enthält die AWS Construct Library, eine Sammlung von Konstrukten, die nach AWS Diensten organisiert sind. Die stabilen Konstrukte der Bibliothek werden in einem einzigen Modul angeboten, das durch seinen TypeScript Paketnamen aufgerufen wird: `aws-cdk-lib`. Der tatsächliche Paketname variiert je nach Sprache.

TypeScript

Installieren

```
npm-Installation aws-cdk-lib
```

Import

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

JavaScript

Installieren	<code>npm-Installation aws-cdk-lib</code>
Import	<code>const cdk = require('aws-cdk-lib');</code>

Python

Installieren	<code>python -m pip-Installation aws-cdk-lib</code>
Import	Importieren von <code>aws_cdk</code> als <code>cdk</code>

Java

Hinzufügen zu <code>pom.xml</code>	<code>Group software.amazon.awscdk ;</code> <code>artifact aws-cdk-lib</code>
Import	Importieren von <code>software.amazon.awscdk.App;</code> (for example)

C#

Installieren	<code>dotnet Paket Amazon.CDK.Lib</code> hinzufügen
Import	mit <code>Amazon.CDK;</code>

Die `construct` Basisklasse und der unterstützende Code befinden sich im `constructs` Modul. Experimentelle Konstrukte, bei denen die API noch verfeinert wird, werden als separate Module verteilt.

Die AWS CDK API-Referenz zu

Die [AWS CDK API-Referenz](#) enthält eine detaillierte Dokumentation der Konstrukte (und anderer Komponenten) in der Bibliothek. Für jede unterstützte Programmiersprache wird eine Version der API-Referenz bereitgestellt.

Das Referenzmaterial jedes Moduls ist in die folgenden Abschnitte unterteilt.

- **Übersicht:** Einführungsmaterial, das Sie kennen müssen, um mit dem Service in der AWS CDK, einschließlich Konzepten und Beispielen, arbeiten zu können.
- **Konstrukte:** Bibliotheksklassen, die eine oder mehrere konkrete AWS Ressourcen darstellen. Dies sind die „kuratierten“ (L2) Ressourcen oder Muster (L3-Ressourcen), die eine High-Level-Schnittstelle mit verzweigten Standardwerten bereitstellen.
- **Klassen :** Nicht-Konstrukt-Klassen, die Funktionen bereitstellen, die von Konstrukten im Modul verwendet werden.
- **Strukturen :** Datenstrukturen (Attributpakete), die die Struktur zusammengesetzter Werte wie Eigenschaften (das Argument von `props` Konstrukten) und Optionen definieren.
- **Schnittstellen :** Schnittstellen, deren Namen alle mit „I“ beginnen, definieren die absolute Mindestfunktionalität für das entsprechende Konstrukt oder eine andere Klasse. Das CDK verwendet Konstruktschnittstellen, um AWS Ressourcen darzustellen, die außerhalb Ihrer AWS CDK App definiert sind und von Methoden wie `referenziert werden Bucket.fromBucketArn()`.
- **Enums :** Sammlungen benannter Werte zur Angabe bestimmter Konstruktparameter. Die Verwendung eines Aufzählungswerts ermöglicht es dem CDK, diese Werte während der Synthetisierung auf Gültigkeit zu überprüfen.
- **CloudFormation Ressourcen :** Diese L1-Konstrukte, deren Namen mit „Cfn“ beginnen, stellen genau die in der CloudFormation Spezifikation definierten Ressourcen dar. Sie werden automatisch mit jeder CDK-Version aus dieser Spezifikation generiert. Jedes L2- oder L3-Konstrukt kapselt eine oder mehrere CloudFormation Ressourcen.
- **CloudFormation Eigenschaftstypen :** Die Sammlung benannter Werte, die die Eigenschaften für jede CloudFormation Ressource definieren.

Schnittstellen im Vergleich zu Konstruktclassen

Die AWS CDK verwendet Schnittstellen auf eine bestimmte Weise, die möglicherweise nicht offensichtlich ist, selbst wenn Sie mit Schnittstellen als Programmierkonzept vertraut sind.

unterstützt die AWS CDK Verwendung von Ressourcen, die außerhalb von CDK-Anwendungen definiert wurden, mit Methoden wie `Bucket.fromBucketArn()`. Externe Ressourcen können nicht geändert werden und verfügen möglicherweise nicht über alle Funktionen, die mit Ressourcen verfügbar sind, die in Ihrer CDK-App definiert sind, z. B. die `Bucket`-Klasse. Schnittstellen stellen dann die im CDK verfügbare Mindestfunktionalität für einen bestimmten AWS Ressourcentyp dar, einschließlich externer Ressourcen.

Wenn Sie Ressourcen in Ihrer CDK-App instanziiieren, sollten Sie immer konkrete Klassen wie `Bucket` verwenden. Wenn Sie den Typ eines Arguments angeben, das Sie in einem Ihrer eigenen Konstrukte akzeptieren, verwenden Sie den Schnittstellentyp, z. B. `IBucket` wenn Sie bereit sind, sich mit externen Ressourcen zu befassen (d. h. Sie müssen sie nicht ändern). Wenn Sie ein CDK-definiertes Konstrukt benötigen, geben Sie den allgemeinsten Typ an, den Sie verwenden können.

Einige Schnittstellen sind Mindestversionen von Eigenschaften oder Optionspaketen, die bestimmten Klassen zugeordnet sind, und nicht Konstrukten. Solche Schnittstellen können nützlich sein, wenn Sie ein Subclassing durchführen, um Argumente zu akzeptieren, die Sie an Ihre übergeordnete Klasse übergeben. Wenn Sie eine oder mehrere zusätzliche Eigenschaften benötigen, möchten Sie diese Schnittstelle oder einen bestimmten Typ implementieren oder daraus ableiten.

Note

Einige von unterstützte Programmiersprachen verfügen AWS CDK über keine Schnittstellenfunktion. In diesen Sprachen sind Schnittstellen einfach übliche Klassen. Sie können sie anhand ihrer Namen identifizieren, die dem Muster eines anfänglichen „I“ gefolgt vom Namen eines anderen Konstrukts folgen (z. B. `IBucket`). Es gelten dieselben Regeln.

Verwalten von Abhängigkeiten

Abhängigkeiten für Ihre AWS CDK App oder Bibliothek werden mithilfe von Paketverwaltungstools verwaltet. Diese Tools werden häufig mit den Programmiersprachen verwendet.

In der Regel AWS CDK unterstützt das standardmäßige oder offizielle Paketverwaltungs-Tool der Sprache, falls es eines gibt. Andernfalls AWS CDK unterstützt die beliebteste oder weit verbreitetste der Sprache. Möglicherweise können Sie auch andere Tools verwenden, insbesondere wenn sie mit den unterstützten Tools funktionieren. Die offizielle Unterstützung für andere Tools ist jedoch begrenzt.

Die AWS CDK unterstützt die folgenden Paketmanager:

Sprache	Unterstütztes Paketverwaltungstool
TypeScript/JavaScript	NPM (Node Package Manager) oder Yarn
Python	PIP (Package Installer für Python)
Java	Maven
C#	NuGet
Go	Go-Module

Wenn Sie ein neues Projekt mit dem AWS CDK CLI `cdk init` Befehl erstellen, werden Abhängigkeiten für die CDK-Kernbibliotheken und stabile Konstrukte automatisch angegeben.

Weitere Informationen zum Verwalten von Abhängigkeiten für unterstützte Programmiersprachen finden Sie im Folgenden:

- [Verwalten von Abhängigkeiten in TypeScript.](#)
- [Verwalten von Abhängigkeiten in JavaScript.](#)
- [Verwaltung von Abhängigkeiten in Python.](#)
- [Verwalten von Abhängigkeiten in Java.](#)
- [Verwalten von Abhängigkeiten in C#.](#)
- [Verwalten von Abhängigkeiten in Go.](#)

Vergleich von AWS CDK in TypeScript mit anderen Sprachen

TypeScript war die erste Sprache, die für die Entwicklung von AWS CDK Anwendungen unterstützt wurde. Daher wird eine beträchtliche Menge an Beispiel-CDK-Code in geschrieben TypeScript. Wenn Sie in einer anderen Sprache entwickeln, kann es nützlich sein, zu vergleichen, wie AWS CDK Code in im TypeScript Vergleich zu Ihrer Sprache Ihrer Wahl implementiert wird. Dies kann Ihnen helfen, die Beispiele in der gesamten Dokumentation zu verwenden.

Importieren eines Moduls

TypeScript/JavaScript

TypeScript unterstützt den Import entweder eines gesamten Namespace oder einzelner Objekte aus einem Namespace. Jeder Namespace enthält Konstrukte und andere Klassen zur Verwendung mit einem bestimmten AWS Service.

```
// 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

Wie unterstützt TypeScript Python auch Importe von namespace-Modulen und selektive Importe. Namespaces in Python sehen wie `aws_cdk.xxx` aus, wobei `xxx` einen - AWS Servicenamen darstellt, z. B. `s3` für Amazon S3. (Amazon S3 wird in diesen Beispielen verwendet).

```
# 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

Java-Importe funktionieren anders als TypeScript. Jede Importanweisung importiert entweder einen einzelnen Klassennamen aus einem bestimmten Paket oder alle in diesem Paket definierten Klassen (mit *). Auf Klassen kann entweder mit dem Klassennamen selbst zugegriffen werden, wenn er importiert wurde, oder mit dem qualifizierten Klassennamen einschließlich seines Pakets.

Bibliotheken werden wie `software.amazon.awscdk.services.xxx` für die AWS Construct Library benannt (die Hauptbibliothek ist `software.amazon.awscdk`). Die Maven-Gruppen-ID für AWS CDK Pakete lautet `software.amazon.awscdk`.

```
// Make certain core classes available
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.App;

// Make all Amazon S3 construct library classes available
import software.amazon.awscdk.services.s3.*;

// Make only Bucket and EventType classes available
import software.amazon.awscdk.services.s3.Bucket;
import software.amazon.awscdk.services.s3.EventType;

// An imported class may now be accessed using the simple class name (assuming that
// name
// does not conflict with another class)
Bucket bucket = Bucket.Builder.create(...).build();
```



```
// We can always use the qualified name of a class (including its package) even
// without an
// import directive
software.amazon.awscdk.services.s3.Bucket bucket =
    software.amazon.awscdk.services.s3.Bucket.Builder.create(...)
        .build();

// Java 10 or later can use var keyword to avoid typing the type twice
var bucket =
    software.amazon.awscdk.services.s3.Bucket.Builder.create(...)
        .build();
```

C#

In C# importieren Sie Typen mit der `-using`Richtlinie. Es gibt zwei Stile. Eine ermöglicht Ihnen den Zugriff auf alle Typen im angegebenen Namespace unter Verwendung ihrer einfachen Namen. Mit dem anderen können Sie mithilfe eines Alias auf den Namespace selbst verweisen.

Pakete werden wie `Amazon.CDK.AWS.xxx` für Pakete der AWS Construct Library benannt. (Das Kernmodul ist `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

Jedes Modul der AWS Construct Library wird als Go-Paket bereitgestellt.

```
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(...)
```

Instanziieren eines Konstrukts

AWS CDK -Konstruktclassen haben in allen unterstützten Sprachen denselben Namen. Die meisten Sprachen verwenden das `new` Schlüsselwort, um eine Klasse zu instanzieren (Python und Go tun dies nicht). Außerdem `this` bezieht sich das Schlüsselwort in den meisten Sprachen auf die aktuelle Instance. (Python verwendet `self` nach Konvention.) Sie sollten einen Verweis auf die aktuelle Instance als `scope` Parameter an jedes Konstrukt übergeben, das Sie erstellen.

Das dritte Argument für ein AWS CDK Konstrukt ist `props`, ein Objekt, das Attribute enthält, die zum Erstellen des Konstrukts erforderlich sind. Dieses Argument kann optional sein, aber wenn es erforderlich ist, behandeln die unterstützten Sprachen es auf idiomatische Weise. Die Namen der Attribute werden auch an die Standardnamensmuster der Sprache angepasst.

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 verwendet beim Instanzieren einer Klasse kein `new` Schlüsselwort. Das Eigenschaftensargument wird mithilfe von Schlüsselwortargumenten dargestellt, und die Argumente werden mit `benanntsnake_case`.

Wenn es sich bei einem Eigenschaftswert selbst um ein Paket von Attributen handelt, wird er durch eine Klasse mit dem Namen nach der Eigenschaft dargestellt, die Schlüsselwortargumente für die Untereigenschaften akzeptiert.

In Python wird die aktuelle Instance als erstes Argument, das `self` nach Konvention benannt ist, an Methoden übergeben.

```
# Instantiate default Bucket
bucket = s3.Bucket(self, "MyBucket")

# Instantiate Bucket with bucket_name and versioned properties
bucket = s3.Bucket(self, "MyBucket", bucket_name="my-bucket", versioned=true)

# Instantiate Bucket with website_redirect, which has its own sub-properties
bucket = s3.Bucket(self, "MyBucket", website_redirect=s3.WebsiteRedirect(
    host_name="aws.amazon.com"))
```

Java

In Java wird das Props-Argument durch eine Klasse mit dem Namen `XxxxProps` (z. B. `BucketProps` für die Props des `Bucket`-Konstrukts). Sie erstellen das Props-Argument mit einem Builder-Muster.

Jede `XxxxProps` Klasse hat einen Builder. Es gibt auch einen praktischen Builder für jedes Konstrukt, das die Eigenschaften und das Konstrukt in einem Schritt erstellt, wie im folgenden Beispiel gezeigt.

Props werden mit gleich benannt wie in TypeScriptcamelCase.

```
// Instantiate default Bucket
Bucket bucket = Bucket(self, "MyBucket");

// Instantiate Bucket with bucketName and versioned properties
Bucket bucket = Bucket.Builder.create(self, "MyBucket")
    .bucketName("my-bucket").versioned(true)
    .build();

# Instantiate Bucket with websiteRedirect, which has its own sub-properties
Bucket bucket = Bucket.Builder.create(self, "MyBucket")
    .websiteRedirect(new websiteRedirect.Builder()
        .hostName("aws.amazon.com").build())
    .build();
```

C#

In C# werden Eigenschaften mithilfe eines Objektinitialisierers für eine Klasse namens `XxxxProps` (z. B. `BucketProps` für die Eigenschaften des `Bucket`-Konstrukts) angegeben.

Props werden ähnlich benannt wie TypeScript, mit Ausnahme von PascalCase.

Es ist praktisch, das `var`-Schlüsselwort beim Instanzieren eines Konstrukts zu verwenden, sodass Sie den Klassennamen nicht zweimal eingeben müssen. Ihr lokaler Leitfaden zum Codestil kann jedoch variieren.

```
// 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

Um ein Konstrukt in Go zu erstellen, rufen Sie die Funktion auf, `NewXxxxxx` wobei der Name des Konstrukts `Xxxxxxx` ist. Die Eigenschaften der Konstrukte sind als Struktur definiert.

In Go sind alle Konstruktparameter Zeiger, einschließlich Werte wie Zahlen, boolesche Werte und Zeichenfolgen. Verwenden Sie die Convenience-Funktionen wie `jsii.String`, um diese Zeiger zu erstellen.

```
// 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"),
    }})
```

Zugreifen auf Mitglieder

Es ist üblich, sich auf Attribute oder Eigenschaften von Konstrukten und anderen AWS CDK Klassen zu beziehen und diese Werte beispielsweise als Eingaben zum Erstellen anderer Konstrukte zu verwenden. Die zuvor beschriebenen Namensunterschiede für Methoden gelten auch hier. Darüber hinaus ist es in Java nicht möglich, direkt auf Mitglieder zuzugreifen. Stattdessen wird eine getter-Methode bereitgestellt.

TypeScript/JavaScript

Namen sind `camelCase`.

```
bucket.bucketArn
```

Python

Namen sind snake_case.

```
bucket.bucket_arn
```

Java

Für jede Eigenschaft wird eine Getter-Methode bereitgestellt. Diese Namen sind camelCase.

```
bucket.getBucketArn()
```

C#

Namen sind PascalCase.

```
bucket.BucketArn
```

Go

Namen sind PascalCase.

```
bucket.BucketArn
```

Aufzählungskonstanten

Enum-Konstanten sind auf eine Klasse beschränkt und haben Großbuchstaben mit Unterstrichen in allen Sprachen (manchmal auch als bezeichnet SCREAMING_SNAKE_CASE). Da Klassennamen in allen unterstützten Sprachen außer Go auch dieselbe Groß- und Kleinschreibung verwenden, sind qualifizierte Enum-Namen in diesen Sprachen ebenfalls identisch.

```
s3.BucketEncryption.KMS_MANAGED
```

In Go sind Enum-Konstanten Attribute des Modul-Namespaces und werden wie folgt geschrieben.

```
awss3.BucketEncryption_KMS_MANAGED
```

Objektschnittstellen

Die AWS CDK verwendet TypeScript Objektschnittstellen, um anzugeben, dass eine Klasse einen erwarteten Satz von Methoden und Eigenschaften implementiert. Sie können eine Objektschnittstelle erkennen, da ihr Name mit `beginntI`. Eine konkrete Klasse gibt die Schnittstellen an, die sie mithilfe des `implements` Schlüsselworts implementiert.

TypeScript/JavaScript

Note

JavaScript verfügt über kein Schnittstellenfeature. Sie können das `implements` Schlüsselwort und die darauf folgenden Klassennamen ignorieren.

```
import { IAspect, IConstruct } from 'aws-cdk-lib';

class MyAspect implements IAspect {
  public visit(node: IConstruct) {
    console.log('Visited', node.node.path);
  }
}
```

Python

Python verfügt über keine Schnittstellenfunktion. Für können AWS CDK Sie jedoch die Schnittstellenimplementierung angeben, indem Sie Ihre Klasse mit `dekorieren@jsii.implements(interface)`.

```
from aws_cdk import IAspect, IConstruct
import jsii

@jsii.implements(IAspect)
class MyAspect():
    def visit(self, node: IConstruct) -> None:
        print("Visited", node.node.path)
```

Java

```
import software.amazon.awscdk.IAspect;
```

```
import software.amazon.awscdk.IConstruct;

public class MyAspect implements IAspect {
    public void visit(IConstruct node) {
        System.out.format("Visited %s", node.getNode().getPath());
    }
}
```

C#

```
using Amazon.CDK;

public class MyAspect : IAspect
{
    public void Visit(IConstruct node)
    {
        System.Console.WriteLine($"Visited ${node.Node.Path}");
    }
}
```

Go

Go-Strukturen müssen nicht explizit deklarieren, welche Schnittstellen sie implementieren. Der Go-Compiler bestimmt die Implementierung basierend auf den Methoden und Eigenschaften, die in der -Struktur verfügbar sind. Im folgenden Code `MyAspect` implementiert beispielsweise die `IAspect` Schnittstelle, da sie eine `Visit` Methode bereitstellt, die ein `Construct` verwendet.

```
type MyAspect struct {
}

func (a MyAspect) Visit(node constructs.IConstruct) {
    fmt.Println("Visited", *node.Node().Path())
}
```

Arbeiten mit der AWS CDK in TypeScript

TypeScript ist eine vollständig unterstützte Client-Sprache für das AWS Cloud Development Kit (AWS CDK) und wird als stabil angesehen. Bei der Arbeit mit dem AWS CDK werden vertraute Tools TypeScript verwendet, darunter Microsoft- TypeScript Compiler (`tsc`), [Node.js](#) und Node Package Manager (`npm`). Sie können [Yarn](#) auch verwenden, wenn Sie möchten, obwohl die Beispiele in

diesem Handbuch NPM verwenden. Die Module, aus denen die AWS Construct Library besteht, werden über das NPM-Repository [npmjs.org](https://www.npmjs.org) verteilt.

Sie können jeden Editor oder jede IDE verwenden. Viele AWS CDK Entwickler verwenden [Visual Studio Code](#) (oder sein Open-Source-Äquivalent [VSCodium](#)), das hervorragende Unterstützung für bietet TypeScript.

Themen

- [Erste Schritte mit TypeScript](#)
- [Erstellen eines Projekts](#)
- [Verwenden von lokal tsc und cdk](#)
- [Verwalten von Modulen AWS der Construct Library](#)
- [Verwalten von Abhängigkeiten in TypeScript](#)
- [AWS CDK idiomias in TypeScript](#)
- [Erstellen, Synthetisieren und Bereitstellen](#)

Erste Schritte mit TypeScript

Um mit der zu arbeiten AWS CDK, müssen Sie über ein - AWS Konto und Anmeldeinformationen verfügen und Node.js und das AWS CDK Toolkit installiert haben. Siehe [Erste Schritte mit der AWS CDK](#).

Sie benötigen auch TypeScript sich selbst (Version 3.8 oder höher). Wenn Sie es noch nicht haben, können Sie es mit installierenpm.

```
npm install -g typescript
```

Note

Wenn Sie einen Berechtigungsfehler erhalten und Administratorzugriff auf Ihr System haben, versuchen Sie es mit `sudo npm install -g typescript`.

Halten Sie sich mit einem regulären TypeScript auf dem Laufendennpm update -g typescript.

Note

Sprachveralterung von Drittanbietern: Die Sprachversion wird nur unterstützt, bis ihr EOL (End of Life) vom Anbieter oder der Community gemeinsam genutzt wird, und kann sich ohne vorherige Ankündigung ändern.

Erstellen eines Projekts

Sie erstellen ein neues AWS CDK Projekt, indem Sie `cdk init` in einem leeren Verzeichnis aufrufen. Verwenden Sie die `--language` Option und geben Sie `typescript`:

```
mkdir my-project
cd my-project
cdk init app --language typescript
```

Durch das Erstellen eines Projekts werden auch das [aws-cdk-lib](#) Modul und seine Abhängigkeiten installiert.

`cdk init` verwendet den Namen des Projektordners, um verschiedene Elemente des Projekts zu benennen, einschließlich Klassen, Unterordner und Dateien. Bindestriche im Ordnernamen werden in Unterstriche umgewandelt. Der Name sollte jedoch ansonsten der Form einer TypeScript Kennung folgen; er sollte beispielsweise nicht mit einer Zahl beginnen oder Leerzeichen enthalten.

Verwenden von lokal **tsc** und **cdk**

In diesem Handbuch wird größtenteils davon ausgegangen, dass Sie installieren TypeScript und das CDK Toolkit global (`npm install -g typescript aws-cdk`) ist. Die bereitgestellten Befehlsbeispiele (z. B. `cdk synth`) folgen dieser Annahme. Dieser Ansatz macht es einfach, beide Komponenten auf dem neuesten Stand zu halten, und da beide einen strengen Ansatz für die Abwärtskompatibilität verfolgen, besteht in der Regel ein geringes Risiko, immer die neuesten Versionen zu verwenden.

Einige Teams ziehen es vor, alle Abhängigkeiten innerhalb jedes Projekts anzugeben, einschließlich Tools wie dem TypeScript Compiler und dem CDK Toolkit. Mit dieser Vorgehensweise können Sie diese Komponenten an bestimmte Versionen anheften und sicherstellen, dass alle Entwickler in Ihrem Team (und Ihrer CI/CD-Umgebung) genau diese Versionen verwenden. Dadurch wird eine mögliche Änderungsquelle beseitigt und Builds und Bereitstellungen werden einheitlicher und wiederholbarer.

Das CDK enthält Abhängigkeiten sowohl für als auch für TypeScript das CDK Toolkit im der TypeScript Projektvorlage. `package.json` Wenn Sie diesen Ansatz verwenden möchten, müssen Sie keine Änderungen an Ihrem Projekt vornehmen. Sie müssen lediglich leicht unterschiedliche Befehle zum Erstellen Ihrer App und zum Ausgeben von `cdk` Befehlen verwenden.

Operation	Globale Tools verwenden	Lokale Tools verwenden
Projekt initialisieren	<code>cdk init --language typescript</code>	<code>npx aws-cdk init --language typescript</code>
Entwicklung	<code>tsc</code>	<code>npm run build</code>
CDK-Toolkit-Befehl ausführen	<code>cdk ...</code>	<code>npm run cdk ...</code> or <code>npx aws-cdk ...</code>

`npx aws-cdk` führt die lokal im aktuellen Projekt installierte Version des CDK Toolkits aus, falls vorhanden, die auf die globale Installation zurückgreift, falls vorhanden. Wenn keine globale Installation vorhanden ist, lädt eine temporäre Kopie des CDK Toolkits `npx` herunter und führt diese aus. Sie können eine beliebige Version des CDK Toolkits mit der `@` Syntax angeben: `npx aws-cdk@2.0 --version` Drückt `2.0.0`.

Tip

Richten Sie einen Alias ein, damit Sie den `cdk` Befehl mit einer lokalen CDK-Toolkit-Installation verwenden können.

macOS/Linux

```
alias cdk="npx aws-cdk"
```

Windows

```
doskey cdk=npx aws-cdk $*
```

Verwalten von Modulen AWS der Construct Library

Verwenden Sie den Node Package Manager (npm), um AWS Construct Library-Module für die Verwendung durch Ihre Apps sowie andere benötigte Pakete zu installieren und zu aktualisieren. (Sie können `yarn` anstelle von `verwendennpm`, wenn Sie möchten.) installiert npm auch die Abhängigkeiten für diese Module automatisch.

Die meisten AWS CDK Konstrukte befinden sich im CDK-Hauptpaket namens `aws-cdk-lib`, was eine Standardabhängigkeit bei neuen Projekten ist `aws-cdk-lib`, die von `cdk init` erstellt wurden. „Experimentell“ Module der AWS Konstruktbibliothek, bei denen sich übergeordnete Konstrukte noch in der Entwicklung befinden, heißen `@aws-cdk/SERVICE-NAME-alpha`. Der Servicename hat das Präfix `aws-`. Wenn Sie sich über den Namen eines Moduls nicht sicher sind, [suchen Sie nach diesem auf NPM](#).

Note

Die [CDK-API-Referenz](#) zeigt auch die Paketnamen an.

Mit dem folgenden Befehl wird beispielsweise das experimentelle Modul für installiert AWS CodeStar.

```
npm install @aws-cdk/aws-codestar-alpha
```

Die Unterstützung der Konstruktbibliothek einiger Services befindet sich in mehr als einem Namespace. Neben `aws-route53es` beispielsweise drei zusätzliche Amazon Route 53-Namespace, `aws-route53-targets`, `aws-route53-patterns`, und `aws-route53resolver`.

Die Abhängigkeiten Ihres Projekts werden in `package.json`. Sie können diese Datei bearbeiten, um einige oder alle Ihrer Abhängigkeiten auf eine bestimmte Version zu sperren oder deren Aktualisierung auf neuere Versionen unter bestimmten Kriterien zu ermöglichen. So aktualisieren Sie die NPM-Abhängigkeiten Ihres Projekts auf die neueste zulässige Version gemäß den Regeln, die Sie in `package.json` angegeben haben:

```
npm update
```

Importieren Sie Module in Ihren Code unter demselben Namen TypeScript, den Sie verwenden, um sie mit NPM zu installieren. Wir empfehlen die folgenden Methoden, wenn Sie AWS CDK Klassen und Module der AWS Construct Library in Ihre Anwendungen importieren. Die Einhaltung dieser

Richtlinien trägt dazu bei, dass Ihr Code mit anderen AWS CDK Anwendungen konsistent und leichter verständlich ist.

- Verwenden Sie `import` Richtlinien im ES6-style, nicht `require()`.
- Importieren Sie im Allgemeinen einzelne Klassen aus `aws-cdk-lib`.

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

- Wenn Sie viele Klassen aus benötigen `aws-cdk-lib`, können Sie einen Namespace-Alias verwenden, `cdk` anstatt die einzelnen Klassen zu importieren. Vermeiden Sie beides.

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

- Im Allgemeinen importieren Sie AWS Service-Konstrukte mit kurzen Namespace-Aliassen.

```
import { aws_s3 as s3 } from 'aws-cdk-lib';
```

Verwalten von Abhängigkeiten in TypeScript

In TypeScript CDK-Projekten werden Abhängigkeiten in der `-package.json` Datei im Hauptverzeichnis des Projekts angegeben. Die AWS CDK Kernmodule befinden sich in einem einzigen NPM Paket namens `aws-cdk-lib`.

Wenn Sie ein Paket mit installieren `npm install`, zeichnet NPM das Paket in `package.json` für Sie auf.

Wenn Sie möchten, können Sie Yarn anstelle von NPM verwenden. Das CDK unterstützt jedoch nicht den Yarn- `plug-and-play` Modus, der der Standardmodus in Yarn 2 ist. Fügen Sie der `-.yarnrc.yml` Datei Ihres Projekts Folgendes hinzu, um diese Funktion zu deaktivieren.

```
nodeLinker: node-modules
```

CDK-Anwendungen

Im Folgenden finden Sie eine `package.json` Beispieldatei, die mit dem `cdk init --language typescript` Befehl generiert wurde:

```
{
```

```
"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"
}
}
```

Für bereitstellbare CDK-Apps `aws-cdk-lib` muss im `dependencies` Abschnitt von angegeben werden `package.json`. Sie können einen caret (^)-Versionsnummernbezeichner verwenden, um anzugeben, dass Sie spätere Versionen als die angegebene akzeptieren werden, solange sie sich innerhalb derselben Hauptversion befinden.

Geben Sie für experimentelle Konstrukte exakte Versionen für die Alpha-Konstrukt-Bibliotheksmodule an, deren APIs sich ändern können. Verwenden Sie nicht ^ oder ~, da spätere Versionen dieser Module API-Änderungen mit sich bringen können, die Ihre App beeinträchtigen können.

Geben Sie im `devDependencies` Abschnitt von Versionen von Bibliotheken und Tools an, die zum Testen Ihrer App erforderlich sind (z. B. `package.json`. das `jest` Test-Framework). Verwenden Sie optional ^, um anzugeben, dass spätere kompatible Versionen akzeptabel sind.

Bibliotheken für Konstrukte von Drittanbietern

Wenn Sie eine Konstruktbibliothek entwickeln, geben Sie ihre Abhängigkeiten mithilfe einer Kombination der `devDependencies` Abschnitte `peerDependencies` und `an`, wie in der folgenden `package.json` Beispieldatei gezeigt.

```
{
  "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"
  }
}
```

Verwenden Sie `peerDependencies` in ein caret (^), um die niedrigste Version von anzugeben `aws-cdk-lib`, mit der Ihre Bibliothek arbeitet. Dadurch wird die Kompatibilität Ihrer Bibliothek mit einer Reihe von CDK-Versionen maximiert. Geben Sie genaue Versionen für Alpha-Konstrukt-Bibliotheksmodule an, deren APIs sich ändern können. Die Verwendung von `peerDependencies` stellt sicher, dass sich nur eine Kopie aller CDK-Bibliotheken im `node_modules` Baum befindet.

Geben Sie in die `Tools` und `devDependencies` Bibliotheken an, die Sie zum Testen benötigen, optional mit ^, um anzuzeigen, dass spätere kompatible Versionen akzeptabel sind. Geben Sie genau (ohne ^ oder ~) die niedrigsten Versionen von `aws-cdk-lib` an und andere CDK-Pakete, mit denen Sie Ihre Bibliothek ankündigen, sind kompatibel. Diese Vorgehensweise stellt sicher, dass Ihre Tests mit diesen Versionen ausgeführt werden. Wenn Sie also versehentlich ein Feature verwenden, das nur in neueren Versionen gefunden wurde, können Ihre Tests es abfangen.

Warning

`peerDependencies` werden automatisch nur von NPM 7 und höher installiert. Wenn Sie NPM 6 oder früher verwenden oder wenn Sie Yarn verwenden, müssen Sie die

Abhängigkeiten Ihrer Abhängigkeiten in `aufnehmendevDependencies`. Andernfalls werden sie nicht installiert, und Sie erhalten eine Warnung über nicht aufgelöste Peer-Abhängigkeiten.

Installieren und Aktualisieren von Abhängigkeiten

Führen Sie den folgenden Befehl aus, um die Abhängigkeiten Ihres Projekts zu installieren.

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
```

Um die installierten Module zu aktualisieren, können die vorhergehenden `yarn upgrade` Befehle `npm install` und verwendet werden. Beide Befehle aktualisieren die Pakete in `node_modules` auf die neuesten Versionen, die die Regeln in `package.json` erfüllen. Sie aktualisieren sich jedoch nicht `package.json` selbst, was Sie möglicherweise tun möchten, um eine neue Mindestversion festzulegen. Wenn Sie Ihr Paket auf `GitHub` hosten, können Sie [Dependabot-Versionsaktualisierungen](#) konfigurieren, um automatisch zu aktualisieren `package.json`. Als alternative Vorgehensweise verwenden Sie [npm-check-updates](#).

Important

Wenn Sie Abhängigkeiten installieren oder aktualisieren, wählen NPM und Yarn standardmäßig die neueste Version jedes Pakets aus, das die in angegebenen Anforderungen erfüllt `package.json`. Es besteht immer das Risiko, dass diese Versionen

beschädigt werden können (entweder versehentlich oder absichtlich). Testen Sie gründlich, nachdem Sie die Abhängigkeiten Ihres Projekts aktualisiert haben.

AWS CDK idiomias in TypeScript

Eigenschaften

Alle AWS Construct Library-Klassen werden mit drei Argumenten instanziiert: dem Bereich, in dem das Konstrukt definiert wird (sein übergeordnetes Element im Konstruktbaum), einer ID und Eigenschaften. Argumenteigenschaften sind ein Paket von Schlüssel-Wert-Paaren, die das Konstrukt zur Konfiguration der erstellten AWS Ressourcen verwendet. Andere Klassen und Methoden verwenden auch das „Bundle of Attribute“-Muster für Argumente.

In `props` wird TypeScript die Form von über eine Schnittstelle definiert, die Ihnen die erforderlichen und optionalen Argumente und ihre Typen mitteilt. Eine solche Schnittstelle wird für jede Art von Argument definiert, die normalerweise für ein einzelnes `props` Konstrukt oder eine Methode spezifisch ist. Beispielsweise gibt das `Bucket`-Konstrukt (in der `aws-cdk-lib/aws-s3` module) ein `props` Argument an, das der `BucketProps` Schnittstelle entspricht.

Wenn eine Eigenschaft selbst ein Objekt ist, z. B. die Eigenschaft `websiteRedirect` von `BucketProps`, hat dieses Objekt seine eigene Schnittstelle, die seine Form entsprechen muss, in diesem Fall `RedirectTarget`.

Wenn Sie eine AWS Klasse der Konstruktbibliothek unterteilen (oder eine Methode überschreiben, die ein eigenschaftsähnliches Argument verwendet), können Sie von der vorhandenen Schnittstelle erben, um eine neue zu erstellen, die alle neuen Eigenschaften angibt, die Ihr Code erfordert. Wenn Sie die übergeordnete Klasse oder Basismethode aufrufen, können Sie im Allgemeinen das gesamte `props`-Argument übergeben, das Sie erhalten haben, da alle Attribute, die im `-Objekt` bereitgestellt, aber nicht in der `-Schnittstelle` angegeben sind, ignoriert werden.

Eine zukünftige Version von AWS CDK könnte zufällig eine neue Eigenschaft mit einem Namen hinzufügen, den Sie für Ihre eigene Eigenschaft verwendet haben. Das Übergeben des Werts, den Sie für die Vererbungskette erhalten, kann dann zu unerwartetem Verhalten führen. Es ist sicherer, eine gültige Kopie der Eigenschaften, die Sie erhalten haben, mit entfernter oder auf eingestellter Eigenschaft zu übergeben `undefined`. Beispielsweise:

```
super(scope, name, {...props, encryptionKeys: undefined});
```

Alternativ können Sie Ihre Eigenschaften benennen, damit klar ist, dass sie zu Ihrem Konstrukt gehören. Auf diese Weise ist es unwahrscheinlich, dass sie in zukünftigen AWS CDK Versionen mit Eigenschaften kollidieren werden. Wenn viele davon vorhanden sind, verwenden Sie ein einziges entsprechend benanntes Objekt, um sie zu speichern.

Fehlende Werte

Fehlende Werte in einem Objekt (z. B. Eigenschaften) haben den Wert `undefined` in TypeScript. In Version 3.7 der Sprache wurden Operatoren eingeführt, die die Arbeit mit diesen Werten vereinfachen, sodass es einfacher ist, Standardwerte und „Kurzspann“-Verkettung anzugeben, wenn ein undefinierter Wert erreicht wird. Weitere Informationen zu diesen Funktionen finden Sie in den [TypeScript Versionshinweisen zu 3.7](#), insbesondere den ersten beiden Funktionen, Optionale Verkettung und Nullische Koalescing.

Erstellen, Synthetisieren und Bereitstellen

Im Allgemeinen sollten Sie sich beim Erstellen und Ausführen Ihrer Anwendung im Stammverzeichnis des Projekts befinden.

Node.js kann nicht TypeScript direkt ausgeführt werden. Stattdessen wird Ihre Anwendung JavaScript mit dem TypeScript Compiler in `konvertierttsc`. Der resultierende JavaScript Code wird dann ausgeführt.

Der tut dies AWS CDK automatisch, wenn er Ihre App ausführen muss. Es kann jedoch nützlich sein, manuell zu kompilieren, um nach Fehlern zu suchen und Tests auszuführen. Um Ihre TypeScript App manuell zu kompilieren, geben Sie `ausnpm run build`. Sie können auch `npm run watch` den Watch-Modus aufrufen, in dem der TypeScript Compiler Ihre App automatisch neu erstellt, wenn Sie Änderungen an einer Quelldatei speichern.

Die in Ihrer AWS CDK App definierten [Stacks](#) können mit den folgenden Befehlen einzeln oder zusammen synthetisiert und bereitgestellt werden. Im Allgemeinen sollten Sie sich im Hauptverzeichnis Ihres Projekts befinden, wenn Sie sie ausgeben.

- `cdk synth`: Stellt eine AWS CloudFormation Vorlage aus einem oder mehreren Stacks in Ihrer AWS CDK App zusammen.
- `cdk deploy`: Stellt die durch einen oder mehrere Stacks in Ihrer AWS CDK App definierten Ressourcen in bereit AWS.

Sie können die Namen mehrerer Stacks angeben, die synthetisiert oder in einem einzigen Befehl bereitgestellt werden sollen. Wenn Ihre App nur einen Stack definiert, müssen Sie ihn nicht angeben.

```
cdk synth                # app defines single stack
cdk deploy Happy Grumpy # app defines two or more stacks; two are deployed
```

Sie können auch die Platzhalter `*` (beliebige Anzahl von Zeichen) und `?` (beliebiges einzelnes Zeichen) verwenden, um Stacks nach Mustern zu identifizieren. Wenn Sie Platzhalter verwenden, schließen Sie das Muster in Anführungszeichen ein. Andernfalls versucht die Shell möglicherweise, sie auf die Namen der Dateien im aktuellen Verzeichnis zu erweitern, bevor sie an das AWS CDK Toolkit übergeben werden.

```
cdk synth "Stack?"      # Stack1, StackA, etc.
cdk deploy "*Stack"    # PipeStack, LambdaStack, etc.
```

Tip

Sie müssen Stacks nicht explizit synthetisieren, bevor Sie sie bereitstellen. `cdk deploy` führt diesen Schritt aus, damit Sie sicherstellen können, dass Ihr neuester Code bereitgestellt wird.

Eine vollständige Dokumentation des `cdk` Befehls finden Sie unter [the section called “AWS CDK Toolkit”](#).

Arbeiten mit der AWS CDK in JavaScript

JavaScript ist eine vollständig unterstützte Client-Sprache für das AWS CDK und gilt als stabil. Beim Arbeiten mit dem AWS Cloud Development Kit (AWS CDK) werden vertraute Tools JavaScript verwendet, einschließlich [Node.js](#) und Node Package Manager (npm). Sie können [Yarn](#) auch verwenden, wenn Sie möchten, obwohl die Beispiele in diesem Handbuch NPM verwenden. Die Module, aus denen die AWS Construct Library besteht, werden über das NPM-Repository [npmjs.org](#) verteilt.

Sie können jeden Editor oder jede IDE verwenden. Viele AWS CDK Entwickler verwenden [Visual Studio Code](#) (oder sein Open-Source-Äquivalent [VSCodium](#)), das gute Unterstützung für bietet JavaScript.

Themen

- [Erste Schritte mit JavaScript](#)
- [Erstellen eines Projekts](#)
- [Verwenden von Local cdk](#)
- [Verwalten von Modulen AWS der Construct Library](#)
- [Verwalten von Abhängigkeiten in JavaScript](#)
- [AWS CDK idioms in JavaScript](#)
- [Synthetisieren und Bereitstellen](#)
- [Verwenden von TypeScript Beispielen mit JavaScript](#)
- [Migrieren zu TypeScript](#)

Erste Schritte mit JavaScript

Um mit der zu arbeiten AWS CDK, müssen Sie über ein - AWS Konto und Anmeldeinformationen verfügen und Node.js und das AWS CDK Toolkit installiert haben. Siehe [Erste Schritte mit der AWS CDK](#).

JavaScript AWS CDK -Anwendungen erfordern keine weiteren Voraussetzungen.

Note

Sprachveralterung von Drittanbietern: Die Sprachversion wird nur unterstützt, bis ihr EOL (End of Life) vom Anbieter oder der Community gemeinsam genutzt wird und kann sich ohne vorherige Ankündigung ändern.

Erstellen eines Projekts

Sie erstellen ein neues AWS CDK Projekt, indem Sie `cdk init` in einem leeren Verzeichnis aufrufen. Verwenden Sie die `--language` Option und geben Sie `javascript`:

```
mkdir my-project
cd my-project
cdk init app --language javascript
```

Durch das Erstellen eines Projekts werden auch das [aws-cdk-lib](#) Modul und seine Abhängigkeiten installiert.

`cdk init` verwendet den Namen des Projektordners, um verschiedene Elemente des Projekts zu benennen, einschließlich Klassen, Unterordner und Dateien. Bindestriche im Ordnernamen werden in Unterstriche umgewandelt. Der Name sollte jedoch ansonsten der Form einer JavaScript Kennung folgen; er sollte beispielsweise nicht mit einer Zahl beginnen oder Leerzeichen enthalten.

Verwenden von Local **cdk**

In diesem Handbuch wird größtenteils davon ausgegangen, dass Sie das CDK Toolkit global installieren (`npm install -g aws-cdk`), und die bereitgestellten Befehlsbeispiele (z. B. `cdk synth`) folgen dieser Annahme. Dieser Ansatz macht es einfach, das CDK Toolkit auf dem neuesten Stand zu halten, und da das CDK einen strengen Ansatz für die Abwärtskompatibilität verfolgt, besteht in der Regel ein geringes Risiko, immer die neueste Version zu verwenden.

Einige Teams ziehen es vor, alle Abhängigkeiten innerhalb jedes Projekts anzugeben, einschließlich Tools wie dem CDK Toolkit. Mit dieser Methode können Sie solche Komponenten an bestimmte Versionen anheften und sicherstellen, dass alle Entwickler in Ihrem Team (und Ihrer CI/CD-Umgebung) genau diese Versionen verwenden. Dadurch wird eine mögliche Änderungsquelle beseitigt und Builds und Bereitstellungen werden einheitlicher und wiederholbarer.

Das CDK enthält eine Abhängigkeit für das CDK Toolkit im der JavaScript Projektvorlagepackage.json. Wenn Sie diesen Ansatz verwenden möchten, müssen Sie keine Änderungen an Ihrem Projekt vornehmen. Sie müssen lediglich leicht unterschiedliche Befehle verwenden, um Ihre App zu erstellen und `cdk` Befehle auszugeben.

Operation	Verwenden des globalen CDK Toolkits	Lokales CDK Toolkit verwenden
Projekt initialisieren	<code>cdk init --Sprache javascript</code>	<code>npx aws-cdk init --language javascript</code>
Ausführen des CDK Toolkit-Befehls	<code>cdk ...</code>	<code>npm-Ausführung cdk... or npx aws-cdk...</code>

`npx aws-cdk` führt die lokal im aktuellen Projekt installierte Version des CDK Toolkits aus, falls vorhanden, die auf die globale Installation zurückgreift, falls vorhanden. Wenn keine globale Installation vorhanden ist, lädt eine temporäre Kopie des CDK Toolkits `npx` herunter und führt diese aus. Sie können eine beliebige Version des CDK Toolkits mit der `@` Syntax angeben: `npx aws-cdk@1.120 --version` Drückt `1.120.0`.

i Tip

Richten Sie einen Alias ein, damit Sie den `cdk` Befehl mit einer lokalen CDK-Toolkit-Installation verwenden können.

macOS/Linux

```
alias cdk="npx aws-cdk"
```

Windows

```
doskey cdk=npx aws-cdk $*
```

Verwalten von Modulen AWS der Construct Library

Verwenden Sie den Node Package Manager (npm), um AWS Construct Library-Module für die Verwendung durch Ihre Apps sowie andere benötigte Pakete zu installieren und zu aktualisieren. (Sie können `yarn` anstelle von `verwendennpm`, wenn Sie möchten.) installiert `npm` auch die Abhängigkeiten für diese Module automatisch.

Die meisten AWS CDK Konstrukte befinden sich im CDK-Hauptpaket namens `aws-cdk-lib`, was eine Standardabhängigkeit bei neuen Projekten ist, die von `cdk init` erstellt wurden. „Experimentell“ AWS Module der Konstruktbibliothek, bei denen sich übergeordnete Konstrukte noch in der Entwicklung befinden, heißen `aws-cdk-lib/SERVICE-NAME-alpha`. Der Servicename hat das Präfix `aws-`. Wenn Sie sich über den Namen eines Moduls nicht sicher sind, [suchen Sie nach diesem auf NPM](#).

i Note

Die [CDK-API-Referenz](#) zeigt auch die Paketnamen an.

Mit dem folgenden Befehl wird beispielsweise das experimentelle Modul für installiert AWS CodeStar.

```
npm install @aws-cdk/aws-codestar-alpha
```

Die Unterstützung der Konstruktbibliothek einiger Services befindet sich in mehr als einem Namespace. Neben gibt `aws-route53es` beispielsweise drei zusätzliche Amazon Route 53-Namespace, `aws-route53-targets`, `aws-route53-patterns`, und `aws-route53resolver`.

Die Abhängigkeiten Ihres Projekts werden in `verwaltpackage.json`. Sie können diese Datei bearbeiten, um einige oder alle Ihrer Abhängigkeiten auf eine bestimmte Version zu sperren oder deren Aktualisierung auf neuere Versionen unter bestimmten Kriterien zu ermöglichen. So aktualisieren Sie die NPM-Abhängigkeiten Ihres Projekts auf die neueste zulässige Version gemäß den Regeln, die Sie in angegeben haben `package.json`:

```
npm update
```

In importieren Sie Module in Ihren Code unter demselben Namen JavaScript, den Sie für deren Installation mit NPM verwenden. Wir empfehlen die folgenden Methoden beim Importieren von AWS CDK Klassen und Modulen AWS der Konstruktbibliothek in Ihre Anwendungen. Die Einhaltung dieser Richtlinien trägt dazu bei, dass Ihr Code mit anderen AWS CDK Anwendungen konsistent und leichter verständlich ist.

- Verwenden Sie `require()`, keine `import` Richtlinien im ES6-style. Ältere Versionen von Node.js unterstützen keine ES6-Importe, sodass die Verwendung der älteren Syntax weiter kompatibel ist. (Wenn Sie ES6-Importe wirklich verwenden möchten, verwenden Sie [esm](#), um sicherzustellen, dass Ihr Projekt mit allen unterstützten Versionen von Node.js kompatibel ist.)
- Importieren Sie im Allgemeinen einzelne Klassen aus `aws-cdk-lib`.

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

- Wenn Sie viele Klassen aus benötigen `aws-cdk-lib`, können Sie einen Namespace-Alias verwenden, `cdk` anstatt die einzelnen Klassen zu importieren. Vermeiden Sie beides.

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

- Importieren Sie im Allgemeinen AWS Construct Libraries mit kurzen Namespace-Aliassen.

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

Verwalten von Abhängigkeiten in JavaScript

In JavaScript CDK-Projekten werden Abhängigkeiten in der `package.json` Datei im Hauptverzeichnis des Projekts angegeben. Die AWS CDK Kernmodule befinden sich in einem einzigen NPM Paket namens `aws-cdk-lib`.

Wenn Sie ein Paket mit `installieren npm install`, zeichnet NPM das Paket in `package.json` für Sie auf.

Wenn Sie möchten, können Sie Yarn anstelle von NPM verwenden. Das CDK unterstützt jedoch nicht den Yarn- `plug-and-play` Modus, der der Standardmodus in Yarn 2 ist. Fügen Sie der `-.yarnrc.yml` Datei Ihres Projekts Folgendes hinzu, um diese Funktion zu deaktivieren.

```
nodeLinker: node-modules
```

CDK-Anwendungen

Im Folgenden finden Sie eine `package.json` Beispieldatei, die mit dem `cdk init --language typescript` Befehl generiert wurde. Die für generierte Datei JavaScript ist ähnlich, nur ohne die TypeScript-bezogenen Einträge.

```
{
  "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"  
}  
}
```

Für bereitstellbare CDK-Apps `aws-cdk-lib` muss im `dependencies` Abschnitt von angegeben werden `package.json`. Sie können einen caret (^) Versionsnummernbezeichner verwenden, um anzugeben, dass Sie neuere Versionen als die angegebene akzeptieren werden, solange sie sich innerhalb derselben Hauptversion befinden.

Geben Sie für experimentelle Konstrukte exakte Versionen für die Alpha-Konstrukt-Bibliotheksmodule an, deren APIs sich ändern können. Verwenden Sie nicht ^ oder ~, da spätere Versionen dieser Module API-Änderungen mit sich bringen können, die Ihre App beeinträchtigen können.

Geben Sie im `devDependencies` Abschnitt von Versionen von Bibliotheken und Tools an, die zum Testen Ihrer App erforderlich sind (z. B. `package.json`. das `jest` Test-Framework). Verwenden Sie optional ^, um anzugeben, dass spätere kompatible Versionen akzeptabel sind.

Bibliotheken für Konstrukte von Drittanbietern

Wenn Sie eine Konstruktbibliothek entwickeln, geben Sie ihre Abhängigkeiten mithilfe einer Kombination der `devDependencies` Abschnitte `peerDependencies` und `an`, wie in der folgenden `package.json` Beispieldatei gezeigt.

```
{  
  "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"  
  }  
}
```

```
}
```

Verwenden Sie `peerDependencies` in ein caret (^), um die niedrigste Version von anzugeben `aws-cdk-lib`, mit der Ihre Bibliothek arbeitet. Dadurch wird die Kompatibilität Ihrer Bibliothek mit einer Reihe von CDK-Versionen maximiert. Geben Sie genaue Versionen für Alpha-Konstrukt-Bibliotheksmodule an, deren APIs sich ändern können. Die Verwendung von `peerDependencies` stellt sicher, dass sich nur eine Kopie aller CDK-Bibliotheken im `node_modules` Baum befindet.

Geben Sie in die Tools und Bibliotheken `devDependencies`, die Sie zum Testen benötigen, optional mit ^, um anzuzeigen, dass spätere kompatible Versionen akzeptabel sind. Geben Sie genau (ohne ^ oder ~) die niedrigsten Versionen von `aws-cdk-lib` an und andere CDK-Pakete, mit denen Sie Ihre Bibliothek ankündigen, sind kompatibel. Diese Vorgehensweise stellt sicher, dass Ihre Tests mit diesen Versionen ausgeführt werden. Wenn Sie also versehentlich ein Feature verwenden, das nur in neueren Versionen gefunden wurde, können Ihre Tests es abfangen.

Warning

`peerDependencies` werden automatisch nur von NPM 7 und höher installiert. Wenn Sie NPM 6 oder früher verwenden oder wenn Sie Yarn verwenden, müssen Sie die Abhängigkeiten Ihrer Abhängigkeiten in einschließend `devDependencies`. Andernfalls werden sie nicht installiert, und Sie erhalten eine Warnung über nicht aufgelöste Peer-Abhängigkeiten.

Installieren und Aktualisieren von Abhängigkeiten

Führen Sie den folgenden Befehl aus, um die Abhängigkeiten Ihres Projekts zu installieren.

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
```

Um die installierten Module zu aktualisieren, können die vorhergehenden `yarn upgrade` Befehle `npm install` und verwendet werden. Beide Befehle aktualisieren die Pakete in `node_modules` auf die neuesten Versionen, die die Regeln in `package.json` erfüllen. Sie aktualisieren sich jedoch nicht `package.json` selbst, was Sie möglicherweise tun möchten, um eine neue Mindestversion festzulegen. Wenn Sie Ihr Paket auf `hosten GitHub`, können Sie [Dependabot-Versionsaktualisierungen](#) konfigurieren, um automatisch zu aktualisieren `package.json`. Als alternative Vorgehensweise verwenden Sie [npm-check-updates](#).

Important

Wenn Sie Abhängigkeiten installieren oder aktualisieren, wählen NPM und Yarn standardmäßig die neueste Version jedes Pakets aus, das die in angegebenen Anforderungen erfüllt `package.json`. Es besteht immer das Risiko, dass diese Versionen beschädigt werden können (entweder versehentlich oder absichtlich). Testen Sie gründlich, nachdem Sie die Abhängigkeiten Ihres Projekts aktualisiert haben.

AWS CDK idioms in JavaScript

Eigenschaften

Alle AWS Klassen der Konstruktbibliothek werden mit drei Argumenten instanziiert: dem Umfang, in dem das Konstrukt definiert wird (der übergeordnete Wert in der Konstruktstruktur), einer ID und `props`, einem Paket von Schlüssel-Wert-Paaren, die das Konstrukt zur Konfiguration der erstellten AWS Ressourcen verwendet. Andere Klassen und Methoden verwenden auch das Muster „Paket von Attributen“ für Argumente.

Die Verwendung einer IDE oder eines Editors mit guter JavaScript automatischer Vervollständigung trägt dazu bei, dass Eigenschaftsnamen nicht falsch geschrieben werden. Wenn ein Konstrukt eine `-encryptionKeys` Eigenschaft erwartet und Sie sie buchstabieren `encryptionkeys`, haben Sie beim Instanzieren des Konstrukts den gewünschten Wert nicht übergeben. Dies kann bei Bedarf zu einem Fehler zur Generierungszeit führen oder dazu führen, dass die Eigenschaft

stillschweigend ignoriert wird, wenn sie optional ist. Im letzteren Fall erhalten Sie möglicherweise ein Standardverhalten, das Sie überschreiben möchten. Seien Sie hier besonders vorsichtig.

Wenn Sie eine AWS Construct Library-Klasse unterteilen (oder eine Methode überschreiben, die ein eigenschaftsähnliches Argument verwendet), sollten Sie zusätzliche Eigenschaften für Ihre eigene Verwendung akzeptieren. Diese Werte werden von der übergeordneten Klasse oder der überschriebenen Methode ignoriert, da in diesem Code nie auf sie zugegriffen wird, sodass Sie im Allgemeinen alle Eigenschaften übergeben können, die Sie erhalten haben.

Eine zukünftige Version von AWS CDK könnte zufällig eine neue Eigenschaft mit einem Namen hinzufügen, den Sie für Ihre eigene Eigenschaft verwendet haben. Das Übergeben des Werts, den Sie für die Vererbungskette erhalten, kann dann zu unerwartetem Verhalten führen. Es ist sicherer, eine gültige Kopie der Eigenschaften, die Sie erhalten haben, mit entfernter oder auf eingestellter Eigenschaft zu übergeben `undefined`. Beispielsweise:

```
super(scope, name, {...props, encryptionKeys: undefined});
```

Alternativ können Sie Ihre Eigenschaften benennen, damit klar ist, dass sie zu Ihrem Konstrukt gehören. Auf diese Weise ist es unwahrscheinlich, dass sie in zukünftigen AWS CDK Versionen mit Eigenschaften kollidieren werden. Wenn viele davon vorhanden sind, verwenden Sie ein einziges entsprechend benanntes Objekt, um sie zu speichern.

Fehlende Werte

Fehlende Werte in einem Objekt (z. B. `props`) haben den Wert `undefined` in JavaScript. Für den Umgang mit diesen gelten die üblichen Techniken. Ein häufiger Ausdruck für den Zugriff auf eine Eigenschaft eines Werts, der möglicherweise nicht definiert ist, ist beispielsweise wie folgt:

```
// 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;
```

Wenn jedoch einen anderen „false“-Wert als haben `a` könnte `undefined`, ist es besser, den Test expliziter zu gestalten. Hier nutzen wir die Tatsache, dass `null` und gleich `undefined` sind, für beide auf einmal zu testen:

```
let c = a == null ? a : a.b;
```

i Tip

Node.js 14.0 und höher unterstützen neue Operatoren, die die Handhabung undefinierter Werte vereinfachen können. Weitere Informationen finden Sie in den [optionalen Vorschlägen für Verkettung](#) und [nullische Zusammenführung](#).

Synthetisieren und Bereitstellen

Die in Ihrer AWS CDK App definierten [Stacks](#) können mit den folgenden Befehlen einzeln oder zusammen synthetisiert und bereitgestellt werden. Im Allgemeinen sollten Sie sich im Hauptverzeichnis Ihres Projekts befinden, wenn Sie sie ausgeben.

- `cdk synth`: Stellt eine AWS CloudFormation Vorlage aus einem oder mehreren Stacks in Ihrer AWS CDK App zusammen.
- `cdk deploy`: Stellt die durch einen oder mehrere Stacks in Ihrer AWS CDK App definierten Ressourcen in bereit AWS.

Sie können die Namen mehrerer Stacks angeben, die in einem einzigen Befehl synthetisiert oder bereitgestellt werden sollen. Wenn Ihre App nur einen Stack definiert, müssen Sie ihn nicht angeben.

```
cdk synth                # app defines single stack
cdk deploy Happy Grumpy # app defines two or more stacks; two are deployed
```

Sie können auch die Platzhalter `*` (beliebige Anzahl von Zeichen) und `?` (beliebiges einzelnes Zeichen) verwenden, um Stacks nach Mustern zu identifizieren. Wenn Sie Platzhalter verwenden, schließen Sie das Muster in Anführungszeichen ein. Andernfalls versucht die Shell möglicherweise, sie auf die Namen der Dateien im aktuellen Verzeichnis zu erweitern, bevor sie an das AWS CDK Toolkit übergeben werden.

```
cdk synth "Stack?"      # Stack1, StackA, etc.
cdk deploy "*Stack"    # PipeStack, LambdaStack, etc.
```

i Tip

Sie müssen Stacks nicht explizit synthetisieren, bevor Sie sie bereitstellen. `cdk deploy` führt diesen Schritt aus, damit Sie sicherstellen können, dass Ihr neuester Code bereitgestellt wird.

Eine vollständige Dokumentation des `cdk` Befehls finden Sie unter [the section called “AWS CDK Toolkit”](#).

Verwenden von TypeScript Beispielen mit JavaScript

[TypeScript](#) ist die Sprache AWS CDK, die wir für die Entwicklung der verwenden, und war die erste Sprache, die für die Entwicklung von Anwendungen unterstützt wurde. Daher sind viele verfügbare AWS CDK Codebeispiele in geschrieben TypeScript. Diese Codebeispiele können eine gute Ressource für JavaScript Entwickler sein. Sie müssen lediglich die TypeScript-spezifischen Teile des Codes entfernen.

TypeScript Ausschnitte verwenden häufig das neuere ECMAScript `import` und `export` Schlüsselwörter, um Objekte aus anderen Modulen zu importieren und die Objekte zu deklarieren, die außerhalb des aktuellen Moduls verfügbar gemacht werden sollen. Node.js hat gerade damit begonnen, diese Schlüsselwörter in seinen neuesten Versionen zu unterstützen. Abhängig von der Version von Node.js, die Sie verwenden (oder unterstützen möchten), können Sie Importe und Exporte umschreiben, um die ältere Syntax zu verwenden.

Importe können durch Aufrufe der `require()` Funktion ersetzt werden.

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');
```

Exporte können dem `module.exports` Objekt zugewiesen werden.

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

Eine Alternative zur Verwendung von Importen und Exporten im alten Stil ist die Verwendung des [-esm](#) Moduls.

Sobald Sie die Importe und Exporte sortiert haben, können Sie in den tatsächlichen Code eintauchen. Möglicherweise stoßen Sie auf diese häufig verwendeten TypeScript Funktionen:

- Anmerkungen eingeben
- Schnittstellendefinitionen
- Typkonvertierungen/-casts
- Zugriffsmodifizierer

Typanmerkungen können für Variablen, Klassenmitglieder, Funktionsparameter und Funktionsrückgabetypen bereitgestellt werden. Für Variablen, Parameter und Mitglieder werden Typen angegeben, indem der Kennung ein Doppelpunkt und der Typ folgen. Funktionsrückgabewerte folgen der Funktionssignatur und bestehen aus einem Doppelpunkt und dem Typ.

Um Code vom Typ mit Anmerkungen in zu konvertieren JavaScript, entfernen Sie den Doppelpunkt und den Typ . Klassenmitglieder müssen einen bestimmten Wert in haben JavaScript; setzen Sie sie auf `undefined`, wenn sie nur eine Typanmerkung in haben TypeScript.

TypeScript

```
var encrypted: boolean = true;

class myStack extends cdk.Stack {
  bucket: s3.Bucket;
  // ...
}

function makeEnv(account: string, region: string) : object {
  // ...
}
```

JavaScript

```
var encrypted = true;

class myStack extends cdk.Stack {
  bucket = undefined;
  // ...
}

function makeEnv(account, region) {
  // ...
}
```

In werden Schnittstellen verwendet TypeScript, um Pakete mit erforderlichen und optionalen Eigenschaften und ihren Typen einen Namen zu geben. Sie können dann den Schnittstellennamen als Typanmerkung verwenden. TypeScript stellt sicher, dass das Objekt, das Sie als verwenden, z. B. ein Argument für eine Funktion die erforderlichen Eigenschaften der richtigen Typen hat.

```
interface myFuncProps {
  code: lambda.Code,
  handler?: string
}
```

JavaScript verfügt über kein Schnittstellenfeature. Sobald Sie die Typanmerkungen entfernt haben, löschen Sie die Schnittstellendeklarationen vollständig.

Wenn eine Funktion oder Methode einen Allzwecktyp zurückgibt (z. B. `object`), Sie diesen Wert jedoch als einen spezifischeren JavaScript untergeordneten Typ behandeln möchten, um auf

Eigenschaften oder Methoden zuzugreifen, die nicht Teil der Schnittstelle des allgemeineren Typs sind, TypeScript lässt Sie den Wert mit `as` umwandeln, gefolgt von einem Typ oder Schnittstellennamen. `as` unterstützt dies nicht (oder benötigt), entfernen Sie also einfach `as` und die folgende Kennung. Eine weniger häufige Umwandlungssyntax besteht darin, einen Typnamen in Klammern zu verwenden `<LikeThis>`. Diese Umwandlungen müssen ebenfalls entfernt werden.

Schließlich TypeScript unterstützt die Zugriffsmodifikatoren `public`, `protected` und `private` für Mitglieder von Klassen. Alle Klassenmitglieder in JavaScript sind öffentlich. Entfernen Sie diese Modifikatoren einfach überall, wo Sie sie sehen.

Wenn Sie wissen, wie Sie diese TypeScript Funktionen identifizieren und entfernen können, ist dies ein großer Weg, um kurze TypeScript Ausschnitte an anzupassen JavaScript. Es kann jedoch unpraktisch sein, längere TypeScript Beispiele auf diese Weise zu konvertieren, da es wahrscheinlicher ist, dass sie andere TypeScript Funktionen verwenden. Für diese Situationen empfehlen wir [Sucrase](#). Sucrase beschwert sich beispielsweise nicht, wenn Code eine undefinierte Variable verwendet, wie es der Fall `tsc` wäre. Wenn es syntaktisch gültig ist, kann Sucrase es mit wenigen Ausnahmen in übersetzen JavaScript. Dies macht es besonders nützlich, um Ausschnitte zu konvertieren, die möglicherweise nicht allein ausgeführt werden können.

Migrieren zu TypeScript

Viele JavaScript Entwickler wechseln zu [TypeScript](#) wenn ihre Projekte größer und komplexer werden. TypeScript ist eine Obermenge von JavaScript – der gesamte JavaScript Code ist gültiger TypeScript Code, sodass keine Änderungen an Ihrem Code erforderlich sind – und es ist auch eine unterstützte AWS CDK Sprache. Typenmerkungen und andere TypeScript Funktionen sind optional und können Ihrer AWS CDK App hinzugefügt werden, sobald Sie Wert darin finden. TypeScript bietet Ihnen auch frühzeitigen Zugriff auf neue JavaScript Funktionen wie optionale Verkettung und nullische Zusammenführung, bevor sie abgeschlossen sind – und ohne dass Sie Node.js aktualisieren müssen.

Die „formbasierten“ Schnittstellen von `ts`, die Pakete mit erforderlichen und optionalen Eigenschaften (und ihren Typen) innerhalb eines Objekts definieren, zulassen, dass häufige Fehler beim Schreiben des Codes erkannt werden, und erleichtern es Ihrer IDE, robuste Empfehlungen zur automatischen Vervollständigung und anderen Echtzeitcodierung bereitzustellen.

Die Codierung in TypeScript beinhaltet einen zusätzlichen Schritt: Kompilieren Ihrer App mit dem TypeScript Compiler `tsc`. Bei typischen AWS CDK Apps dauert die Kompilierung maximal einige Sekunden.

Am einfachsten können Sie eine vorhandene JavaScript AWS CDK App zu migrieren, TypeScript indem Sie ein neues TypeScript Projekt mit erstellen `cdk init app --language typescript` und dann Ihre Quelldateien (und alle anderen erforderlichen Dateien, z. B. Komponenten wie AWS Lambda Funktionsquellcode) in das neue Projekt kopieren. Benennen Sie Ihre JavaScript Dateien um, damit sie in enden `.ts` und mit der Entwicklung in beginnen TypeScript.

Arbeiten mit dem AWS CDK in Python

Python ist eine vollständig unterstützte Client-Sprache für die AWS Cloud Development Kit (AWS CDK) und gilt als stabil. AWS CDK Bei der Arbeit mit Python werden vertraute Tools verwendet, darunter die Standard-Python-Implementierung (CPython), virtuelle Umgebungen mit `virtualenv` und das Python-Paketinstallationsprogramm `pip`. Die Module, aus denen die AWS Construct-Bibliothek besteht, werden über pypi.org vertrieben. Die Python-Version von AWS CDK Even verwendet Bezeichner im Python-Stil (z. B. Methodennamen). `snake_case`

Sie können jeden Editor oder jede IDE verwenden. [Viele AWS CDK Entwickler verwenden Visual Studio Code \(oder sein Open-Source-Äquivalent VsCodium\), das Python über eine offizielle Erweiterung gut unterstützt.](#) Der in Python enthaltene IDLE-Editor reicht aus, um loszulegen. Die Python-Module für die AWS CDK haben Typhinweise, die für ein Linting-Tool oder eine IDE, die Typvalidierung unterstützt, nützlich sind.

Themen

- [Erste Schritte mit Python](#)
- [Erstellen eines Projekts](#)
- [Verwaltung von AWS Construct Library-Modulen](#)
- [Verwaltung von Abhängigkeiten in Python](#)
- [AWS CDK Redewendungen in Python](#)
- [Synthetisieren und bereitstellen](#)

Erste Schritte mit Python

Um mit dem arbeiten zu können AWS CDK, müssen Sie über ein AWS Konto und Anmeldeinformationen verfügen und Node.js und das AWS CDK Toolkit installiert haben. Siehe [Erste Schritte mit der AWS CDK](#).

AWS CDK Python-Anwendungen benötigen Python 3.6 oder höher. Falls Sie es noch nicht installiert haben, [laden Sie eine kompatible Version](#) für Ihr Betriebssystem von python.org herunter. Wenn Sie Linux verwenden, wurde Ihr System möglicherweise mit einer kompatiblen Version geliefert, oder Sie können es mit dem Paketmanager Ihrer Distribution installieren (yumapt, usw.). Mac-Benutzer könnten an [Homebrew](#) interessiert sein, einem Paketmanager im Linux-Stil für macOS.

Note

Veraltete Sprachversion von Drittanbietern: Die Sprachversion wird nur bis zu ihrem vom Anbieter oder der Community mitgeteilten EOL (End Of Life) unterstützt und kann sich mit vorheriger Ankündigung ändern.

Das Python-Paketinstallationsprogramm und der Virtual Environment Manager sind ebenfalls erforderlich. `pip` `virtualenv` Windows-Installationen kompatibler Python-Versionen enthalten diese Tools. Unter Linux `pip` und `virtualenv` kann als separate Pakete in Ihrem Paketmanager bereitgestellt werden. Alternativ können Sie sie mit den folgenden Befehlen installieren:

```
python -m ensurepip --upgrade
python -m pip install --upgrade pip
python -m pip install --upgrade virtualenv
```

Wenn Sie auf einen Berechtigungsfehler stoßen, führen Sie die obigen Befehle mit der `--user` Markierung aus, sodass die Module in Ihrem Benutzerverzeichnis installiert werden, oder verwenden Sie `sudo` sie, um die Berechtigungen für die systemweite Installation der Module zu erhalten.

Note

Es ist üblich, dass Linux-Distributionen den Namen der ausführbaren Datei `python3` für Python 3.x verwenden und auf eine Python 2.x-Installation `python` verweisen. Einige Distributionen haben ein optionales Paket, das Sie installieren können, damit sich der `python` Befehl auf Python 3 bezieht. Andernfalls können Sie den Befehl, mit dem Ihre Anwendung ausgeführt wird, anpassen, indem Sie ihn `cdk.json` im Hauptverzeichnis des Projekts bearbeiten.

Note

Unter Windows möchten Sie vielleicht Python (und pip) mit der `py` ausführbaren Datei, dem [Python-Launcher für Windows](#), aufrufen. Mit dem Launcher können Sie unter anderem einfach angeben, welche installierte Version von Python Sie verwenden möchten. Wenn die Eingabe `python` in der Befehlszeile zu einer Meldung über die Installation von Python aus dem Windows Store führt, auch nach der Installation einer Windows-Version von Python, öffnen Sie den Windows-Einstellungsbereich „App-Ausführungs-Aliase verwalten“ und deaktivieren Sie die beiden App-Installer-Einträge für Python.

Erstellen eines Projekts

Sie erstellen ein neues AWS CDK Projekt, indem Sie es `cdk init` in einem leeren Verzeichnis aufrufen. Verwenden Sie die `--language` Option und geben Sie `anypython`:

```
mkdir my-project
cd my-project
cdk init app --language python
```

`cdk init` verwendet den Namen des Projektordners, um verschiedene Elemente des Projekts zu benennen, einschließlich Klassen, Unterordnern und Dateien. Bindestriche im Ordernamen werden in Unterstriche umgewandelt. Ansonsten sollte der Name jedoch der Form eines Python-Bezeichners folgen; er sollte beispielsweise nicht mit einer Zahl beginnen oder Leerzeichen enthalten.

Um mit dem neuen Projekt zu arbeiten, aktivieren Sie seine virtuelle Umgebung. Dadurch können die Abhängigkeiten des Projekts lokal im Projektordner installiert werden und müssen nicht global installiert werden.

```
source .venv/bin/activate
```

Note

Sie kennen diesen Befehl vielleicht als Mac-/Linux-Befehl zum Aktivieren einer virtuellen Umgebung. Die Python-Vorlagen enthalten eine Batch-Datei, `source.bat`, die die Verwendung desselben Befehls unter Windows ermöglicht. Der herkömmliche Windows-Befehl funktioniert auch. `.\venv\Scripts\activate`

Wenn Sie Ihr AWS CDK Projekt mit CDK Toolkit v1.70.0 oder früher initialisiert haben, befindet sich Ihre virtuelle Umgebung im Verzeichnis statt. `.env` `.venv`

Important

Aktivieren Sie die virtuelle Umgebung des Projekts, wann immer Sie damit beginnen, daran zu arbeiten. Andernfalls haben Sie keinen Zugriff auf die dort installierten Module, und die von Ihnen installierten Module werden in das globale Python-Modulverzeichnis verschoben (oder führen zu einem Berechtigungsfehler).

Nachdem Sie Ihre virtuelle Umgebung zum ersten Mal aktiviert haben, installieren Sie die Standardabhängigkeiten der App:

```
python -m pip install -r requirements.txt
```

Verwaltung von AWS Construct Library-Modulen

Verwenden Sie das Python-Paketinstallationsprogramm `pip`, um AWS Construct Library-Module für Ihre Apps sowie andere Pakete, die Sie benötigen, zu installieren und zu aktualisieren. `pip` installiert auch die Abhängigkeiten für diese Module automatisch. Wenn Ihr System den Befehl nicht `pip` als eigenständigen Befehl erkennt, rufen Sie ihn wie folgt `pip` als Python-Modul auf:

```
python -m pip PIP-COMMAND
```

Die meisten AWS CDK Konstrukte sind in `aws-cdk-lib` Experimentelle Module befinden sich in separaten Modulen mit dem Namen wie `aws-cdk.SERVICE-NAME.alpha`. Der Dienstname enthält ein AWS-Präfix. Wenn Sie sich des Namens eines Moduls nicht sicher sind, [suchen Sie bei PyPI danach](#). Mit dem folgenden Befehl wird beispielsweise die Bibliothek installiert. AWS CodeStar

```
python -m pip install aws-cdk.aws-codestar-alpha
```

Die Konstrukte einiger Dienste befinden sich in mehr als einem Namespace. Außerdem gibt es `aws-cdk.aws-route53` beispielsweise drei zusätzliche Amazon Route 53-Namespace mit den Namen `aws-cdk.aws-route53-targets`, `aws-cdk.aws-route53-patterns` und `aws-cdk.aws-route53resolver`

Note

Die [Python-Ausgabe der CDK API Reference](#) zeigt auch die Paketnamen.

Die Namen, die für den Import von AWS Construct Library-Modulen in Ihren Python-Code verwendet werden, sehen wie folgt aus.

```
import aws_cdk.aws_s3 as s3
import aws_cdk.aws_lambda as lambda_
```

Wir empfehlen die folgenden Methoden, wenn Sie AWS CDK Klassen und AWS Construct Library-Module in Ihre Anwendungen importieren. Das Befolgen dieser Richtlinien trägt dazu bei, dass Ihr Code mit anderen AWS CDK Anwendungen konsistent und verständlicher ist.

- Im Allgemeinen sollten Sie einzelne Klassen aus der obersten Ebene `aws_cdk` importieren.

```
from aws_cdk import App, Construct
```

- Wenn Sie viele Klassen aus dem benötigen `aws_cdk`, können Sie einen Namespace-Alias verwenden, `cdk` anstatt einzelne Klassen zu importieren. Vermeiden Sie es, beides zu tun.

```
import aws_cdk as cdk
```

- Im Allgemeinen sollten Sie AWS Construct-Bibliotheken mit kurzen Namespace-Aliasen importieren.

```
import aws_cdk.aws_s3 as s3
```

Aktualisieren Sie nach der Installation eines Moduls die `requirements.txt` Datei Ihres Projekts, in der die Abhängigkeiten Ihres Projekts aufgeführt sind. Es ist am besten, dies manuell zu tun, anstatt es zu verwenden `pip freeze`. `pip freeze` erfasst die aktuellen Versionen aller Module, die in Ihrer virtuellen Python-Umgebung installiert sind. Dies kann nützlich sein, wenn Sie ein Projekt bündeln, das an anderer Stelle ausgeführt werden soll.

Normalerweise `requirements.txt` sollten Sie jedoch nur Abhängigkeiten der obersten Ebene auflisten (Module, von denen Ihre App direkt abhängt) und nicht die Abhängigkeiten dieser Bibliotheken. Diese Strategie erleichtert die Aktualisierung Ihrer Abhängigkeiten.

Sie können Änderungen vornehmen `requirements.txt`, um Upgrades zuzulassen. Ersetzen Sie einfach die `==` vorherige Versionsnummer durch, `~=` um Upgrades auf eine höhere kompatible Version zu ermöglichen, oder entfernen Sie die Versionsanforderung vollständig, um die neueste verfügbare Version des Moduls anzugeben.

Wenn die `requirements.txt` Datei entsprechend bearbeitet wurde, um Upgrades zu ermöglichen, geben Sie diesen Befehl ein, um die in Ihrem Projekt installierten Module jederzeit zu aktualisieren:

```
pip install --upgrade -r requirements.txt
```

Verwaltung von Abhängigkeiten in Python

In Python spezifizieren Sie Abhängigkeiten, indem Sie sie `requirements.txt` für Anwendungen oder `setup.py` für Konstruktbibliotheken eingeben. Abhängigkeiten werden dann mit dem PIP-Tool verwaltet. PIP wird auf eine der folgenden Arten aufgerufen:

```
pip command options  
python -m pip command options
```

Der `python -m pip` Aufruf funktioniert auf den meisten Systemen; `pip` erfordert, dass sich die ausführbare Datei von PIP im Systempfad befindet. Wenn `pip` das nicht funktioniert, versuchen Sie, es durch `python -m pip` zu ersetzen.

Der `cdk init --language python` Befehl erstellt eine virtuelle Umgebung für Ihr neues Projekt. Dadurch kann jedes Projekt seine eigenen Versionen von Abhängigkeiten und auch eine `requirements.txt` Basisdatei haben. Sie müssen diese virtuelle Umgebung `source .venv/bin/activate` jedes Mal aktivieren, indem Sie sie ausführen, wenn Sie mit der Arbeit an dem Projekt beginnen. Führen Sie unter Windows `.\venv\Scripts\activate` stattdessen den Befehl aus

CDK-Anwendungen

Im Folgenden sehen Sie ein Beispiel für eine `requirements.txt`-Datei. Da PIP nicht über eine Funktion zum Sperren von Abhängigkeiten verfügt, empfehlen wir, den Operator `==` zu verwenden, um exakte Versionen für alle Abhängigkeiten anzugeben, wie hier gezeigt.

```
aws-cdk-lib==2.14.0  
aws-cdk.aws-appsync-alpha==2.10.0a0
```

Durch die Installation eines Moduls mit `pip install` wird es nicht automatisch hinzugefügt. `requirements.txt` Das müssen Sie selbst tun. Wenn Sie ein Upgrade auf eine neuere Version einer Abhängigkeit durchführen möchten, bearbeiten Sie deren Versionsnummer unter `requirements.txt`.

Um die Abhängigkeiten Ihres Projekts nach der Erstellung oder Bearbeitung zu installieren oder zu aktualisieren `requirements.txt`, führen Sie den folgenden Befehl aus:

```
python -m pip install -r requirements.txt
```

Tip

Der `pip freeze` Befehl gibt die Versionen aller installierten Abhängigkeiten in einem Format aus, das in eine Textdatei geschrieben werden kann. Dies kann als Anforderungsdatei mit verwendet werden `pip install -r`. Diese Datei ist praktisch, um alle Abhängigkeiten (einschließlich transitiver Abhängigkeiten) genau den Versionen zuzuordnen, mit denen Sie getestet haben. Um Probleme bei der späteren Aktualisierung von Paketen zu vermeiden, verwenden Sie hierfür eine separate Datei, z. B. `freeze.txt` (not `requirements.txt`). Generieren Sie sie dann neu, wenn Sie die Abhängigkeiten Ihres Projekts aktualisieren.

Konstruktbibliotheken von Drittanbietern

In Bibliotheken werden Abhängigkeiten in `setup.py` angegeben, sodass transitive Abhängigkeiten automatisch heruntergeladen werden, wenn das Paket von einer Anwendung verwendet wird. Andernfalls muss jede Anwendung, die Ihr Paket verwenden möchte, Ihre Abhängigkeiten in ihre `requirements.txt` Dateien kopieren. Ein Beispiel `setup.py` wird hier gezeigt.

```
from setuptools import setup

setup(
    name='my-package',
    version='0.0.1',
    install_requires=[
        'aws-cdk-lib==2.14.0',
    ],
    ...
)
```


Um an dem Paket für die Entwicklung zu arbeiten, erstellen oder aktivieren Sie eine virtuelle Umgebung und führen Sie dann den folgenden Befehl aus.

```
python -m pip install -e .
```

Obwohl PIP automatisch transitive Abhängigkeiten installiert, kann von jedem Paket nur eine Kopie installiert werden. Die Version, die im Abhängigkeitsbaum am höchsten angegeben ist, wird ausgewählt; Anwendungen haben immer das letzte Wort darüber, welche Version von Paketen installiert wird.

AWS CDK Redewendungen in Python

Sprachkonflikte

In Python `lambda` ist dies ein Sprachschlüsselwort, sodass Sie es nicht als Namen für das AWS Lambda Construct-Bibliotheksmodul oder für Lambda-Funktionen verwenden können. Die Python-Konvention für solche Konflikte besteht darin, im Variablennamen einen abschließenden Unterstrich wie in `lambda_` zu verwenden.

Konventionell wird das zweite Argument von AWS CDK Konstrukten benannt. `id` Wenn Sie Ihre eigenen Stapel und Konstrukte schreiben, `id` „überschattet“ der Aufruf eines Parameters die eingebaute Python-Funktion `id()`, die den eindeutigen Bezeichner eines Objekts zurückgibt. Diese Funktion wird nicht sehr oft verwendet, aber wenn Sie sie zufällig in Ihrem Konstrukt benötigen sollten, benennen Sie das Argument beispielsweise um `construct_id`.

Argumente und Eigenschaften

Alle AWS Construct-Library-Klassen werden mit drei Argumenten instanziiert: dem Bereich, in dem das Konstrukt definiert wird (sein übergeordnetes Objekt im Konstruktbaum), einer ID und Requisiten, einem Bündel von Schlüssel/Wert-Paaren, die das Konstrukt zur Konfiguration der von ihm erstellten Ressourcen verwendet. Andere Klassen und Methoden verwenden ebenfalls das Muster „Bündel von Attributen“ für Argumente.

`scope` und `id` sollten immer als Positionsargumente und nicht als Schlüsselwortargumente übergeben werden, da sich ihre Namen ändern, wenn das Konstrukt eine Eigenschaft namens `scope` oder `id` akzeptiert.

In Python werden Requisiten als Schlüsselwortargumente ausgedrückt. Wenn ein Argument verschachtelte Datenstrukturen enthält, werden diese mithilfe einer Klasse ausgedrückt, die bei der

Instanziierung ihre eigenen Schlüsselwortargumente verwendet. Das gleiche Muster wird auf andere Methodenaufrufen angewendet, die ein strukturiertes Argument verwenden.

In der `add_lifecycle_rule` Methode eines Amazon S3 S3-Buckets ist die `transitions` Eigenschaft beispielsweise eine Liste von Transition Instances.

```
bucket.add_lifecycle_rule(  
    transitions=[  
        Transition(  
            storage_class=StorageClass.GLACIER,  
            transition_after=Duration.days(10)  
        )  
    ]  
)
```

Wenn Sie eine Klasse erweitern oder eine Methode überschreiben, möchten Sie möglicherweise zusätzliche Argumente für Ihre eigenen Zwecke akzeptieren, die von der übergeordneten Klasse nicht verstanden werden. In diesem Fall sollten Sie die Argumente akzeptieren, die Ihnen egal sind, indem Sie das `**kwargs` Idiom verwenden, und nur Stichwortargumente verwenden, um die Argumente zu akzeptieren, an denen Sie interessiert sind. Wenn Sie den Konstruktor des übergeordneten Elements oder die überschriebene Methode aufrufen, übergeben Sie nur die Argumente, die das übergeordnete Objekt erwartet (oft nur). `**kwargs` Die Übergabe von Argumenten, die die übergeordnete Klasse oder Methode nicht erwartet, führt zu einem Fehler.

```
class MyConstruct(Construct):  
    def __init__(self, id, *, MyProperty=42, **kwargs):  
        super().__init__(self, id, **kwargs)  
        # ...
```

In einer future Version von AWS CDK könnte zufällig eine neue Immobilie mit einem Namen hinzugefügt werden, den Sie für Ihre eigene Immobilie verwendet haben. Dies wird zwar keine technischen Probleme für Benutzer Ihres Konstrukts oder Ihrer Methode verursachen (da Ihre Eigenschaft nicht „in der Kette“ weitergereicht wird, verwendet die übergeordnete Klasse oder überschriebene Methode einfach einen Standardwert), kann aber zu Verwirrung führen. Sie können dieses potenzielle Problem vermeiden, indem Sie Ihre Eigenschaften so benennen, dass sie eindeutig zu Ihrem Konstrukt gehören. Wenn es viele neue Eigenschaften gibt, bündeln Sie sie in einer Klasse mit dem entsprechenden Namen und übergeben Sie sie als einzelnes Schlüsselwortargument.

Fehlende Werte

Wird AWS CDK verwendet `None`, um fehlende oder undefinierte Werte darzustellen. Wenn Sie mit `arbeiten**kwargs`, verwenden Sie die `get()` Methode des Wörterbuchs, um einen Standardwert anzugeben, falls eine Eigenschaft nicht angegeben wird. Vermeiden Sie die Verwendung `kwargs[...]`, da dies `KeyError` auf fehlende Werte zurückzuführen ist.

```
encrypted = kwargs.get("encrypted")           # None if no property "encrypted" exists
encrypted = kwargs.get("encrypted", False)    # specify default of False if property is
missing
```

Einige AWS CDK Methoden (z. B. `tryGetContext()` zum Abrufen eines Laufzeitkontextwerts) geben möglicherweise einen Wert zurück `None`, den Sie explizit überprüfen müssen.

Verwenden von Schnittstellen

Python hat keine Schnittstellenfunktion wie einige andere Sprachen, obwohl es [abstrakte Basisklassen](#) hat, die ähnlich sind. (Wenn Sie mit Schnittstellen nicht vertraut sind, hat Wikipedia [eine gute Einführung](#).) TypeScript, die Sprache, in der das implementiert AWS CDK ist, stellt Schnittstellen bereit, und Konstrukte und andere AWS CDK Objekte erfordern oft ein Objekt, das einer bestimmten Schnittstelle entspricht, anstatt es von einer bestimmten Klasse zu erben. [Das AWS CDK bietet also eine eigene Schnittstellenfunktion als Teil der JSII-Schicht](#).

Um anzugeben, dass eine Klasse eine bestimmte Schnittstelle implementiert, können Sie den `@jsii.implements` Decorator verwenden:

```
from aws_cdk import IAspect, IConstruct
import jsii

@jsii.implements(IAspect)
class MyAspect():
    def visit(self, node: IConstruct) -> None:
        print("Visited", node.node.path)
```

Geben Sie Fallstricke ein

Python verwendet dynamische Typisierung, bei der sich alle Variablen auf einen Wert beliebigen Typs beziehen können. Parameter und Rückgabewerte können mit Typen annotiert werden, aber das sind „Hinweise“ und werden nicht erzwungen. Das bedeutet, dass es in Python einfach ist, einem

AWS CDK Konstrukt den falschen Wertetyp zu übergeben. Anstatt beim Build einen Typfehler zu erhalten, wie Sie es bei einer statisch typisierten Sprache tun würden, erhalten Sie möglicherweise einen Laufzeitfehler, wenn die JSII-Schicht (die zwischen Python und dem TypeScript Kern übersetzt) den AWS CDK unerwarteten Typ nicht verarbeiten kann.

Unserer Erfahrung nach fallen die Typfehler, die Python-Programmierer machen, in diese Kategorien.

- Übergeben eines einzelnen Werts, wenn ein Konstrukt einen Container erwartet (Python-Liste oder Wörterbuch) oder umgekehrt.
- Übergabe eines Werts eines Typs, der mit einem Layer-1-Konstrukt (CfnXxxxxx) verknüpft ist, an ein L2- oder L3-Konstrukt oder umgekehrt.

Die AWS CDK Python-Module enthalten Typanmerkungen, sodass Sie Tools verwenden können, die sie unterstützen, um bei Typen zu helfen. Wenn Sie beispielsweise keine IDE verwenden, die diese unterstützt, möchten Sie möglicherweise den [MyPy](#) Typvalidator als Schritt in Ihrem Build-Prozess aufrufen. [PyCharm](#) Es gibt auch Laufzeit-Typprüfer, mit denen Fehlermeldungen für typbezogene Fehler verbessert werden können.

Synthetisieren und bereitstellen

Die in Ihrer AWS CDK App definierten [Stacks](#) können mithilfe der folgenden Befehle einzeln oder zusammen synthetisiert und bereitgestellt werden. Im Allgemeinen sollten Sie sich im Hauptverzeichnis Ihres Projekts befinden, wenn Sie sie ausgeben.

- `cdk synth`: Synthetisiert eine AWS CloudFormation Vorlage aus einem oder mehreren Stacks in Ihrer AWS CDK App.
- `cdk deploy`: Stellt die durch einen oder mehrere Stacks in Ihrer App definierten Ressourcen bereit. `AWS CDK AWS`

Sie können die Namen mehrerer Stacks, die synthetisiert oder bereitgestellt werden sollen, in einem einzigen Befehl angeben. Wenn Ihre App nur einen Stack definiert, müssen Sie ihn nicht angeben.

```
cdk synth # app defines single stack
cdk deploy Happy Grumpy # app defines two or more stacks; two are deployed
```

Sie können auch die Platzhalter `*` (eine beliebige Anzahl von Zeichen) und `?` verwenden (jedes einzelne Zeichen), um Stapel anhand eines Musters zu identifizieren. Wenn Sie Platzhalter

verwenden, setzen Sie das Muster in Anführungszeichen. Andernfalls versucht die Shell möglicherweise, es auf die Namen der Dateien im aktuellen Verzeichnis zu erweitern, bevor sie an das Toolkit übergeben werden. AWS CDK

```
cdk synth "Stack?"    # Stack1, StackA, etc.  
cdk deploy "*Stack"  # PipeStack, LambdaStack, etc.
```

Tip

Sie müssen Stacks nicht explizit synthetisieren, bevor Sie sie bereitstellen; `cdk deploy` führt diesen Schritt für Sie aus, um sicherzustellen, dass Ihr neuester Code bereitgestellt wird.

Eine vollständige Dokumentation des `cdk` Befehls finden Sie unter [the section called “AWS CDK Toolkit”](#)

Arbeiten mit in AWS CDK Java

Java ist eine vollständig unterstützte Client-Sprache für die AWS CDK und gilt als stabil. Sie können AWS CDK Anwendungen in Java mit vertrauten Tools entwickeln, einschließlich des JDK (Oracle oder eine OpenJDK-Distribution wie Amazon Corretto) und Apache Maven.

Die AWS CDK unterstützt Java 8 und höher. Wir empfehlen jedoch, die neueste Version zu verwenden, da spätere Versionen der Sprache Verbesserungen enthalten, die besonders praktisch für die Entwicklung von AWS CDK Anwendungen sind. Java 9 führt beispielsweise die `-Map.of()` Methode ein (eine bequeme Möglichkeit, Hashmaps zu deklarieren, die als Objeklitterale in geschrieben werden würden TypeScript). Java 10 führt eine lokale Typinferenz mit dem `var` Schlüsselwort ein.

Note

Die meisten Codebeispiele in diesem Entwicklerhandbuch funktionieren mit Java 8. Einige Beispiele verwenden `Map.of()`. Diese Beispiele enthalten Kommentare, die darauf hinweisen, dass sie Java 9 erfordern.

Sie können einen beliebigen Texteditor oder eine Java-IDE verwenden, die Maven-Projekte lesen kann, um an Ihren AWS CDK Apps zu arbeiten. In diesem Handbuch stellen wir [Eclipse](#)-Hinweise

bereit, aber IntelliJ IDEA NetBeans und andere IDEs können Maven-Projekte importieren und für die Entwicklung von AWS CDK Anwendungen in Java verwendet werden.

Es ist möglich, AWS CDK Anwendungen in anderen JVM-gehosteten Sprachen als Java zu schreiben (z. B. Kotlin, Groovy, Clojure oder Scala), aber die Erfahrung ist möglicherweise nicht besonders idiomatisch und wir können keine Unterstützung für diese Sprachen anbieten.

Themen

- [Erste Schritte mit Java](#)
- [Erstellen eines Projekts](#)
- [Verwalten von Modulen AWS der Construct Library](#)
- [Verwalten von Abhängigkeiten in Java](#)
- [AWS CDK idioms in Java](#)
- [Erstellen, Synthetisieren und Bereitstellen](#)

Erste Schritte mit Java

Um mit der zu arbeiten AWS CDK, müssen Sie über ein - AWS Konto und Anmeldeinformationen verfügen und Node.js und das AWS CDK Toolkit installiert haben. Siehe [Erste Schritte mit der AWS CDK](#).

Java- AWS CDK Anwendungen erfordern Java 8 (v1.8) oder höher. Wir empfehlen [Amazon Corretto](#) , aber Sie können jede OpenJDK-Verteilung oder [Oracle-JDK](#) verwenden. Sie benötigen auch [Apache Maven](#) 3.5 oder höher. Sie können auch Tools wie Gradle verwenden, aber die vom AWS CDK Toolkit generierten Anwendungs-Skeletons sind Maven-Projekte.

Note

Sprachveralterung von Drittanbietern: Die Sprachversion wird nur unterstützt, bis ihr EOL (End of Life) vom Anbieter oder der Community gemeinsam genutzt wird und kann sich ohne vorherige Ankündigung ändern.

Erstellen eines Projekts

Sie erstellen ein neues AWS CDK Projekt, indem Sie `cdk init` in einem leeren Verzeichnis aufrufen. Verwenden Sie die `--language` Option und geben Sie `anjava`:

```
mkdir my-project
cd my-project
cdk init app --language java
```

`cdk init` verwendet den Namen des Projektordners, um verschiedene Elemente des Projekts zu benennen, einschließlich Klassen, Unterordner und Dateien. Bindestriche im Ordnernamen werden in Unterstriche umgewandelt. Der Name sollte jedoch ansonsten der Form einer Java-Kennung folgen; er sollte beispielsweise nicht mit einer Zahl beginnen oder Leerzeichen enthalten.

Das resultierende Projekt enthält einen Verweis auf das `software.amazon.awscdk` Maven-Paket. Es und seine Abhängigkeiten werden von Maven automatisch installiert.

Wenn Sie eine IDE verwenden, können Sie das Projekt jetzt öffnen oder importieren. Wählen Sie in Eclipse beispielsweise Datei > Import > Maven > Vorhandene Maven-Projekte aus. Stellen Sie sicher, dass die Projekteinstellungen für die Verwendung von Java 8 (1.8) festgelegt sind.

Verwalten von Modulen AWS der Construct Library

Verwenden Sie Maven, um AWS Construct Library-Pakete zu installieren, die sich in der Gruppe `software.amazon.awscdk` befinden. Die meisten Konstrukte befinden sich im Artefakt `aws-cdk-lib`, das standardmäßig neuen Java-Projekten hinzugefügt wird. Module für Services, deren CDK-Unterstützung auf höherer Ebene noch entwickelt wird, befinden sich in separaten „experimentellen“ Paketen, benannt mit einer Kurzversion (kein AWS oder Amazon-Präfix) des Namens ihres Services. [Suchen Sie im Maven Central Repository](#) nach den Namen aller Bibliotheken AWS CDK und AWS Konstruktmodulbibliotheken.

Note

Die [Java-Edition der CDK-API-Referenz](#) zeigt auch die Paketnamen an.

Einige Services „AWS Construct Library“-Unterstützung befinden sich in mehr als einem Namespace. Amazon Route 53 hat beispielsweise seine Funktionalität in `software.amazon.awscdk.route53`, `route53-patternsroute53resolver`, und `unterteilroute53-targets`.

Das AWS CDK Hauptpaket wird im Java-Code als `importiertsoftware.amazon.awscdk`. Module für die verschiedenen Services in der AWS Construct Library befinden sich unter `software.amazon.awscdk.services` und werden ähnlich wie ihr Maven-

Paketname benannt. Der Namespace des Amazon S3-Moduls lautet beispielsweise `software.amazon.awscdk.services.s3`.

Wir empfehlen, für jede AWS Construct Library-Klasse, die Sie in jeder Ihrer Java-Quelldateien verwenden, eine separate Java-`import`-Anweisung zu schreiben und Platzhalterimporte zu vermeiden. Sie können den vollqualifizierten Namen eines Typs (einschließlich seines Namespace) ohne `-import`-Anweisung immer verwenden.

Wenn Ihre Anwendung von einem experimentellen Paket abhängt, bearbeiten Sie die Ihres Projekts `pom.xml` und fügen Sie dem `<dependencies>` Container ein neues `<dependency>` Element hinzu. Das folgende `<dependency>` Element gibt beispielsweise das CodeStar experimentelle Konstruktbibliotheksmodul an:

```
<dependency>
  <groupId>software.amazon.awscdk</groupId>
  <artifactId>codestar-alpha</artifactId>
  <version>2.0.0-alpha.10</version>
</dependency>
```

Tip

Wenn Sie eine Java-IDE verwenden, verfügt sie wahrscheinlich über Funktionen zur Verwaltung von Maven-Abhängigkeiten. Wir empfehlen jedoch, die Bearbeitung `pom.xml` direkt durchzuführen, es sei denn, Sie sind sich absolut sicher, dass die Funktionalität der IDE mit der manuellen Ausführung übereinstimmt.

Verwalten von Abhängigkeiten in Java

In Java werden Abhängigkeiten in angegeben `pom.xml` und mit Maven installiert. Der `<dependencies>` Container enthält ein `<dependency>`-Element für jedes Paket. Im Folgenden finden Sie einen Abschnitt von `pom.xml` für eine typische CDK-Java-App.

```
<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

Viele Java-IDEs verfügen über integrierte Mavenpom.xml-Unterstützung und visuelle Editoren, die Sie möglicherweise für die Verwaltung von Abhängigkeiten benötigen.

Maven unterstützt keine Abhängigkeitssperre. Obwohl es möglich ist, Versionsbereiche in anzugebenpom.xml, empfehlen wir Ihnen, immer exakte Versionen zu verwenden, um Ihre Builds wiederholbar zu halten.

Maven installiert automatisch transitive Abhängigkeiten, aber es kann nur eine installierte Kopie jedes Pakets geben. Die Version, die im POM-Baum am höchsten angegeben ist, wird ausgewählt. Anwendungen haben immer das letzte Wort in der installierten Version von Paketen.

Maven installiert oder aktualisiert Ihre Abhängigkeiten automatisch, wenn Sie Ihr Projekt erstellen (mvn compile) oder verpacken (mvn package). Das CDK Toolkit tut dies automatisch bei jeder Ausführung, sodass Maven im Allgemeinen nicht manuell aufgerufen werden muss.

AWS CDK idioms in Java

Eigenschaften

Alle AWS Klassen der Konstruktbibliothek werden mit drei Argumenten instanziiert: dem Bereich, in dem das Konstrukt definiert wird (der übergeordnete Wert in der Konstruktstruktur), einer ID und props, einem Paket von Schlüssel-Wert-Paaren, die das Konstrukt zur Konfiguration der erstellten Ressourcen verwendet. Andere Klassen und Methoden verwenden auch das „Bundle of Attribute“-Muster für Argumente.

In Java werden Props mit dem [Builder-Muster](#) ausgedrückt. Jeder Konstrukttyp hat einen entsprechenden Props-Typ. Beispielsweise verwendet das BucketKonstrukt (das einen Amazon S3-Bucket darstellt) als Props eine Instance von BucketProps.

Die BucketProps Klasse (wie jede AWS Construct Library-Props-Klasse) hat eine innere Klasse namens Builder. Der BucketProps.Builder Typ bietet Methoden zum Festlegen der

verschiedenen Eigenschaften einer `BucketProps` Instance. Jede Methode gibt die `Builder` Instance zurück, sodass die Methodenaufrufe verkettet werden können, um mehrere Eigenschaften festzulegen. Am Ende der Kette rufen Sie auf, `build()` um das `BucketProps` Objekt tatsächlich zu erzeugen.

```
Bucket bucket = new Bucket(this, "MyBucket", new BucketProps.Builder()
    .versioned(true)
    .encryption(BucketEncryption.KMS_MANAGED)
    .build());
```

Konstrukte und andere Klassen, die ein eigenschaftsähnliches Objekt als endgültiges Argument verwenden, bieten eine Verknüpfung. Die Klasse hat einen eigenen, `Builder` der sie und ihr Eigenschaftsobjekt in einem Schritt instanziiert. Auf diese Weise müssen Sie nicht explizit `BucketProps` sowohl als auch eine instanziiieren (z. B.) `Bucket` und Sie benötigen keinen Import für den `props`-Typ.

```
Bucket bucket = Bucket.Builder.create(this, "MyBucket")
    .versioned(true)
    .encryption(BucketEncryption.KMS_MANAGED)
    .build();
```

Wenn Sie Ihr eigenes Konstrukt aus einem vorhandenen Konstrukt ableiten, möchten Sie möglicherweise zusätzliche Eigenschaften akzeptieren. Wir empfehlen Ihnen, diese `Builder`-Muster zu befolgen. Dies ist jedoch nicht so einfach wie das Unterteilen einer Konstrukt-Klasse. Sie müssen die verschiebenden Teile der beiden neuen `Builder` Klassen selbst angeben. Möglicherweise möchten Sie einfach, dass Ihr Konstrukt ein oder mehrere zusätzliche Argumente akzeptiert. Sie sollten zusätzliche Konstruktoren angeben, wenn ein Argument optional ist.

Allgemeine Strukturen

In einigen APIs AWS CDK verwendet die JavaScript Arrays oder nicht typisierte Objekte als Eingabe für eine Methode. (Siehe z. B. die [BuildSpec.fromObject\(\)](#) Methode `AWS CodeBuild` von `.`) In Java werden diese Objekte als `dargestellt java.util.Map<String, Object>`. In Fällen, in denen die Werte alle Zeichenfolgen sind, können Sie verwenden `Map<String, String>`.

Java bietet keine Möglichkeit, Literale für solche Container zu schreiben, wie es bei einigen anderen Sprachen der Fall ist. In Java 9 und höher können Sie verwenden, [java.util.Map.of\(\)](#) um Karten mit bis zu zehn Einträgen bequem inline mit einem dieser Aufrufe zu definieren.

```
java.util.Map.of(
    "base-directory", "dist",
    "files", "LambdaStack.template.json"
)
```

Um Karten mit mehr als zehn Einträgen zu erstellen, verwenden Sie [java.util.Map.ofEntries\(\)](#).

Wenn Sie Java 8 verwenden, können Sie Ihre eigenen Methoden bereitstellen, die diesen ähneln.

JavaScript -Arrays werden als `List<Object>` oder `List<String>` in Java dargestellt. Die Methode `java.util.Arrays.asList` eignet sich für die Definition kurzer `List`.

```
List<String> cmds = Arrays.asList("cd lambda", "npm install", "npm install typescript")
```

Fehlende Werte

In Java werden fehlende Werte in AWS CDK Objekten wie Eigenschaften durch `dargestelltnull`. Sie müssen jeden Wert, der sein könnte, explizit testen, `null` um sicherzustellen, dass er einen Wert enthält, bevor Sie etwas damit tun. Java verfügt nicht über „syntaktische Telefonie“, um bei der Handhabung von Nullwerten zu helfen, wie es bei anderen Sprachen der Fall ist. Möglicherweise finden Sie [defaultIfNull](#) und `ObjectUtil` von Apache in einigen Situationen [firstNonNull](#) nützlich. Alternativ können Sie Ihre eigenen statischen Hilfsmethoden schreiben, um die Handhabung potenziell Nullwerte zu erleichtern und Ihren Code lesbarer zu machen.

Erstellen, Synthetisieren und Bereitstellen

Die kompiliert Ihre App AWS CDK automatisch, bevor sie ausgeführt wird. Es kann jedoch nützlich sein, Ihre App manuell zu erstellen, um nach Fehlern zu suchen und Tests auszuführen. Sie können dies in Ihrer IDE tun (z. B. Control-B in Eclipse drücken) oder indem Sie `mvn compile` an einer Eingabeaufforderung ausgeben, während Sie sich im Stammverzeichnis Ihres Projekts befinden.

Führen Sie alle Tests aus, die Sie geschrieben haben, indem Sie `mvn test` an einer Eingabeaufforderung ausführen.

Die in Ihrer AWS CDK App definierten [Stacks](#) können mithilfe der folgenden Befehle einzeln oder zusammen synthetisiert und bereitgestellt werden. Im Allgemeinen sollten Sie sich im Hauptverzeichnis Ihres Projekts befinden, wenn Sie sie ausgeben.

- `cdk synth`: Stellt eine AWS CloudFormation Vorlage aus einem oder mehreren Stacks in Ihrer AWS CDK App zusammen.
- `cdk deploy`: Stellt die von einem oder mehreren Stacks in Ihrer AWS CDK App definierten Ressourcen in bereit AWS.

Sie können die Namen mehrerer Stacks angeben, die in einem einzigen Befehl synthetisiert oder bereitgestellt werden sollen. Wenn Ihre App nur einen Stack definiert, müssen Sie ihn nicht angeben.

```
cdk synth                # app defines single stack
cdk deploy Happy Grumpy # app defines two or more stacks; two are deployed
```

Sie können auch die Platzhalter `*` (beliebige Anzahl von Zeichen) und `?` (beliebiges einzelnes Zeichen) verwenden, um Stacks nach Mustern zu identifizieren. Wenn Sie Platzhalter verwenden, schließen Sie das Muster in Anführungszeichen ein. Andernfalls versucht die Shell möglicherweise, sie auf die Namen der Dateien im aktuellen Verzeichnis zu erweitern, bevor sie an das AWS CDK Toolkit übergeben werden.

```
cdk synth "Stack?"      # Stack1, StackA, etc.
cdk deploy "*Stack"    # PipeStack, LambdaStack, etc.
```

Tip

Sie müssen Stacks nicht explizit synthetisieren, bevor Sie sie bereitstellen. `cdk deploy` führt diesen Schritt aus, damit Sie sicherstellen können, dass Ihr neuester Code bereitgestellt wird.

Eine vollständige Dokumentation des `cdk` Befehls finden Sie unter [the section called “AWS CDK Toolkit”](#).

Arbeiten mit in AWS CDK C#

.NET ist eine vollständig unterstützte Client-Sprache für das AWS CDK und gilt als stabil. C# ist die Hauptsprache von .NET, für die wir Beispiele und Support bereitstellen. Sie können Anwendungen AWS CDK in anderen .NET-Sprachen schreiben, z. B. Visual Basic oder F#, aber AWS bietet eingeschränkte Unterstützung für die Verwendung dieser Sprachen mit dem CDK.

Sie können AWS CDK Anwendungen in C# mit vertrauten Tools wie Visual Studio, Visual Studio Code, dem `-dotnet` Befehl und dem NuGet Paketmanager entwickeln. Die Module, aus denen die AWS Construct Library besteht, werden über nuget.org verteilt.

Wir empfehlen, [Visual Studio 2019](#) (jede Edition) unter Windows zu verwenden, um AWS CDK Apps in C# zu entwickeln.

Themen

- [Erste Schritte mit C#](#)
- [Erstellen eines Projekts](#)
- [Verwalten von Modulen AWS der Construct Library](#)
- [Verwalten von Abhängigkeiten in C#](#)
- [AWS CDK idioms in C#](#)
- [Erstellen, Synthetisieren und Bereitstellen](#)

Erste Schritte mit C#

Um mit der zu arbeiten AWS CDK, müssen Sie über ein - AWS Konto und Anmeldeinformationen verfügen und Node.js und das AWS CDK Toolkit installiert haben. Siehe [Erste Schritte mit der AWS CDK](#).

C# AWS CDK -Anwendungen erfordern .NET Core v3.1 oder höher, die [hier](#) verfügbar sind.

Die .NET-Toolchain enthält `dotnet`, ein Befehlszeilen-Tool zum Erstellen und Ausführen von .NET-Anwendungen und zum Verwalten von NuGet Paketen. Selbst wenn Sie hauptsächlich in Visual Studio arbeiten, kann dieser Befehl für Batchoperationen und für die Installation von AWS Construct Library-Paketen nützlich sein.

Erstellen eines Projekts

Sie erstellen ein neues AWS CDK Projekt, indem Sie `cdk init` in einem leeren Verzeichnis aufrufen. Verwenden Sie die `--language` Option und geben Sie `ancsharp`:

```
mkdir my-project
cd my-project
cdk init app --language csharp
```

`cdk init` verwendet den Namen des Projektordners, um verschiedene Elemente des Projekts zu benennen, einschließlich Klassen, Unterordner und Dateien. Bindestriche im Ordnernamen werden in Unterstriche umgewandelt. Der Name sollte jedoch ansonsten der Form einer C#-Kennung folgen; er sollte beispielsweise nicht mit einer Zahl beginnen oder Leerzeichen enthalten.

Das resultierende Projekt enthält einen Verweis auf das `-Amazon.CDK.Lib` NuGet Paket. Es und seine Abhängigkeiten werden automatisch von installiert NuGet.

Verwalten von Modulen AWS der Construct Library

Das .NET-Ökosystem verwendet den NuGet Paketmanager. Das Haupt-CDK-Paket, das die Kernklassen und alle stabilen Servicekonstrukte enthält, ist `Amazon.CDK.Lib`. Experimentelle Module, bei denen sich neue Funktionen in der aktiven Entwicklung befinden, werden als bezeichnet `Amazon.CDK.AWS.SERVICE-NAME.Alpha`, wobei der Servicename ein Kurzname ohne ein `-AWS` oder `Amazon`-Präfix ist. Der NuGet Paketname für das AWS IoT Modul lautet beispielsweise `Amazon.CDK.AWS.IoT.Alpha`. Wenn Sie ein gewünschtes Paket nicht finden können, [suchen Sie Nuget.org](#).

Note

Die [.NET-Edition der CDK-API-Referenz](#) zeigt auch die Paketnamen an.

Einige Services „AWS Construct Library“-Unterstützung befinden sich in mehr als einem Modul. Beispielsweise AWS IoT hat ein zweites Modul mit dem Namen `Amazon.CDK.AWS.IoT.Actions.Alpha`.

Das Hauptmodul AWS CDK von , das Sie in den meisten AWS CDK Apps benötigen, wird im C#-Code als `importiertAmazon.CDK`. Module für die verschiedenen Services in der AWS Construct Library befinden sich unter `Amazon.CDK.AWS`. Der Namespace des Amazon S3-Moduls lautet beispielsweise `Amazon.CDK.AWS.S3`.

Wir empfehlen, C#-`using`Richtlinien für die CDK-Kernkonstrukte und für jeden AWS Service zu schreiben, den Sie in jeder Ihrer C#-Quelldateien verwenden. Möglicherweise finden Sie es praktisch, einen Alias für einen Namespace oder Typ zu verwenden, um Namenskonflikte zu lösen. Sie können den vollqualifizierten Namen eines Typs (einschließlich seines Namespace) ohne eine `-using`Anweisung verwenden.

Verwalten von Abhängigkeiten in C#

In C# AWS CDK apps verwalten Sie Abhängigkeiten mit NuGet. NuGet verfügt über vier Standardschnittstellen, größtenteils gleichwertige Schnittstellen. Verwenden Sie diejenige, die Ihren Anforderungen und Ihrem Arbeitsstil entspricht. Sie können auch kompatible Tools wie [Paket](#) oder [MyGet](#) sogar die `.csproj` Datei direkt bearbeiten.

NuGet Mit können Sie keine Versionsbereiche für Abhängigkeiten angeben. Jede Abhängigkeit ist an eine bestimmte Version angeheftet.

Nachdem Sie Ihre Abhängigkeiten aktualisiert haben, verwendet Visual Studio , NuGet um die angegebenen Versionen jedes Pakets beim nächsten Erstellen abzurufen. Wenn Sie Visual Studio nicht verwenden, verwenden Sie den `dotnet restore` Befehl , um Ihre Abhängigkeiten zu aktualisieren.

Direktes Bearbeiten der Projektdatei

Die `.csproj` Datei Ihres Projekts enthält einen `<ItemGroup>` Container, der Ihre Abhängigkeiten als `<PackageReferenceElemente` auflistet.

```
<ItemGroup>
  <PackageReference Include="Amazon.CDK.Lib" Version="2.14.0" />
  <PackageReference Include="Constructs" Version="%constructs-version%" />
</ItemGroup>
```

Die Visual Studio NuGet -Benutzeroberfläche

Auf die NuGet Tools von Visual Studio kann über `Tools > NuGet Paketmanager > NuGet Pakete für Lösung verwalten` zugegriffen werden. Verwenden Sie die Registerkarte `Durchsuchen`, um die Pakete der AWS Konstruktbibliothek zu finden, die Sie installieren möchten. Sie können die gewünschte Version auswählen, einschließlich Vorabversionen Ihrer Module, und sie jedem der offenen Projekte hinzufügen.

Note

Alle Module AWS der Konstruktbibliothek, die als „experimentell“ eingestuft werden (siehe [the section called “Versionsverwaltung”](#)), werden in als Vorabversion gekennzeichnet NuGet und haben ein `alpha` Namenssuffix.

The screenshot shows the NuGet Package Manager console in Visual Studio. The 'Updates' tab is selected, displaying a list of packages with updates available. The package 'Amazon.CDK.AWS.Redshift.Alpha' is selected, and its details are shown on the right. The details include a description, version (2.0.0-rc.24), author (Amazon Web Services), license (Apache-2.0), and dependencies.

Amazon.CDK.AWS.Redshift.Alpha by Amazon Web Services, 2,33K downloads, 2.0.0-rc.24
The CDK Construct Library for AWS::Redshift (Stability: Experimental)

Amazon.CDK.AWS.Redshift.Alpha Details:

- Description:** The CDK Construct Library for AWS::Redshift (Stability: Experimental)
- Version:** 2.0.0-rc.24
- Author(s):** Amazon Web Services
- License:** Apache-2.0
- Date published:** Wednesday, October 13, 2021 (10/13/2021)
- Report Abuse:** <https://www.nuget.org/packages/Amazon.CDK.AWS.Redshift.Alpha/2.0.0-rc.24/ReportAbuse>
- Tags:** aws, cdk, constructs, redshift
- Dependencies:**
 - .NETCoreApp,Version=v3.1
 - Amazon.CDK.Lib (>= 2.0.0-rc.24)
 - Amazon.JSII.Runtime (>= 1.39.0 && < 2.0.0)
 - Constructs (>= 10.0.0 && < 11.0.0)

Suchen Sie auf der Seite Updates, um neue Versionen Ihrer Pakete zu installieren.

Die NuGet Konsole

Die NuGet Konsole ist eine PowerShell-basierte Schnittstelle zu NuGet, die im Kontext eines Visual-Studio-Projekts funktioniert. Sie können es in Visual Studio öffnen, indem Sie Tools > NuGet Package Manager > Package Manager Console auswählen. Weitere Informationen zur Verwendung dieses

Tools finden Sie unter [Installieren und Verwalten von Paketen mit der Package Manager-Konsole in Visual Studio](#).

Der **dotnet** Befehl

Der `dotnet` Befehl ist das primäre Befehlszeilen-Tool für die Arbeit mit Visual Studio C#-Projekten. Sie können sie von jeder Windows-Eingabeaufforderung aus aufrufen. Unter seinen vielen Funktionen `dotnet` kann einem Visual Studio-Projekt Abhängigkeiten hinzufügen NuGet.

Angenommen, Sie befinden sich im selben Verzeichnis wie die Visual Studio-Projektdatei (`.csproj`), geben Sie einen Befehl wie den folgenden aus, um ein Paket zu installieren. Da die CDK-Hauptbibliothek beim Erstellen eines Projekts enthalten ist, müssen Sie nur explizit experimentelle Module installieren. Experimentelle Module erfordern die Angabe einer expliziten Versionsnummer.

```
dotnet add package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

Sie können den Befehl aus einem anderen Verzeichnis ausgeben. Fügen Sie dazu den Pfad zur Projektdatei oder zum Verzeichnis, das sie enthält, nach dem `add` Schlüsselwort ein. Im folgenden Beispiel wird davon ausgegangen, dass Sie sich im Hauptverzeichnis Ihres AWS CDK Projekts befinden.

```
dotnet add src/PROJECT-DIR package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

Um eine bestimmte Version eines Pakets zu installieren, fügen Sie das `--v` Flag und die gewünschte Version ein.

Um ein Paket zu aktualisieren, geben Sie denselben `dotnet add` Befehl aus, mit dem Sie es installiert haben. Für experimentelle Module müssen Sie erneut eine explizite Versionsnummer angeben.

Weitere Informationen zum Verwalten von Paketen mit dem `dotnet` Befehl finden Sie unter [Installieren und Verwalten von Paketen mit der dotnet-CLI](#).

Der **nuget** Befehl

Das `nuget` Befehlszeilen-Tool kann NuGet Pakete installieren und aktualisieren. Es erfordert jedoch, dass Ihr Visual Studio-Projekt anders eingerichtet wird als die Art und Weise, wie Projekte `cdk init` einrichtet. (Technische Details: `nuget` funktioniert mit `Packages.config` Projekten, während ein neueres `PackageReference` Projekt `cdk init` erstellt.)

Wir empfehlen nicht, das `nuget` Tool mit AWS CDK Projekten zu verwenden, die von `cdk init` erstellt wurden. Wenn Sie einen anderen Projekttyp verwenden und verwenden möchten, finden Sie weitere Informationen in der [NuGet CLI-Referenz](#).

AWS CDK idioms in C#

Eigenschaften

Alle AWS Klassen der Konstruktbibliothek werden mit drei Argumenten instanziiert: dem Bereich, in dem das Konstrukt definiert wird (der übergeordnete Wert in der Konstruktstruktur), einer ID und `props`, einem Paket von Schlüssel-Wert-Paaren, die das Konstrukt zur Konfiguration der erstellten Ressourcen verwendet. Andere Klassen und Methoden verwenden auch das „Bundle of Attribute“-Muster für Argumente.

In C# werden Props mit einem Props-Typ ausgedrückt. In idiomatischer C#-Weise können wir einen Objektinitialisierer verwenden, um die verschiedenen Eigenschaften festzulegen. Hier erstellen wir einen Amazon S3-Bucket mit dem `Bucket`-Konstrukt. Der entsprechende Props-Typ ist `BucketProps`.

```
var bucket = new Bucket(this, "MyBucket", new BucketProps {  
    Versioned = true  
});
```

Tip

Fügen Sie das Paket `Amazon.JSII.Analyzers` zu Ihrem Projekt hinzu, um die Überprüfung der erforderlichen Werte in Ihren Eigenschaftsdefinitionen in Visual Studio zu erhalten.

Wenn Sie eine Klasse erweitern oder eine Methode überschreiben, möchten Sie möglicherweise zusätzliche Eigenschaften für Ihre eigenen Zwecke akzeptieren, die von der übergeordneten Klasse nicht verstanden werden. Unterklassen Sie dazu den entsprechenden Eigenschaftstyp und fügen Sie die neuen Attribute hinzu.

```
// 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"
});
```

Wenn Sie den Initialisierer oder die überschriebene Methode der übergeordneten Klasse aufrufen, können Sie im Allgemeinen die Eigenschaften übergeben, die Sie erhalten haben. Der neue Typ ist mit seinem übergeordneten -Typ kompatibel, und zusätzliche Eigenschaften, die Sie hinzugefügt haben, werden ignoriert.

Eine zukünftige Version von AWS CDK könnte zufällig eine neue Eigenschaft mit einem Namen hinzufügen, den Sie für Ihre eigene Eigenschaft verwendet haben. Dies führt nicht zu technischen Problemen bei der Verwendung Ihres Konstrukts oder Ihrer Methode (da Ihre Eigenschaft nicht „über die Kette“ übergeben wird, verwendet die übergeordnete Klasse oder überschriebene Methode einfach einen Standardwert), kann jedoch zu Verwirrung für die Benutzer Ihres Konstrukts führen. Sie können dieses potenzielle Problem vermeiden, indem Sie Ihre Eigenschaften benennen, damit sie eindeutig zu Ihrem Konstrukt gehören. Wenn es viele neue Eigenschaften gibt, bündeln Sie sie in einer entsprechend benannten Klasse und übergeben Sie sie als einzelne Eigenschaft.

Allgemeine Strukturen

In einigen APIs AWS CDK verwendet die JavaScript Arrays oder nicht typisierte Objekte als Eingabe für eine Methode. (Siehe z. B. die [BuildSpec.fromObject\(\)](#) Methode AWS CodeBuild von .) In C# werden diese Objekte als `dargestelltSystem.Collections.Generic.Dictionary<String, Object>`. In Fällen, in denen die Werte alle Zeichenfolgen sind, können Sie `Dictionary<String, String>`. JavaScript aufrufen als `- object[]` oder `string[]`-Array-Typen in C# darstellen.

Tip

Sie können kurze Aliase definieren, um die Arbeit mit diesen spezifischen Wörterbuchtypen zu erleichtern.

```
using StringDict = System.Collections.Generic.Dictionary<string, string>;
using ObjectDict = System.Collections.Generic.Dictionary<string, object>;
```

Fehlende Werte

In C# werden fehlende Werte in AWS CDK Objekten wie Eigenschaften durch `dargestelltNull`. Der Operator für den bedingten Zugriff auf Mitglieder `?.` und der Operator für die Nullzusammenführung `??` eignen sich für die Arbeit mit diesen Werten.

```
// 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";
```

Erstellen, Synthetisieren und Bereitstellen

Die kompiliert Ihre App AWS CDK automatisch, bevor sie ausgeführt wird. Es kann jedoch nützlich sein, Ihre App manuell zu erstellen, um nach Fehlern zu suchen und Tests auszuführen. Drücken Sie dazu F6 in Visual Studio oder geben Sie `dotnet build src` über die Befehlszeile aus, wobei das Verzeichnis in Ihrem Projektverzeichnis `src` ist, das die Visual Studio-Lösungsdatei (`.sln`) enthält.

Die in Ihrer AWS CDK App definierten [Stacks](#) können mithilfe der folgenden Befehle einzeln oder zusammen synthetisiert und bereitgestellt werden. Im Allgemeinen sollten Sie sich im Hauptverzeichnis Ihres Projekts befinden, wenn Sie sie ausgeben.

- `cdk synth`: Stellt eine AWS CloudFormation Vorlage aus einem oder mehreren Stacks in Ihrer AWS CDK App zusammen.
- `cdk deploy`: Stellt die von einem oder mehreren Stacks in Ihrer AWS CDK App definierten Ressourcen in bereit AWS.

Sie können die Namen mehrerer Stacks angeben, die in einem einzigen Befehl synthetisiert oder bereitgestellt werden sollen. Wenn Ihre App nur einen Stack definiert, müssen Sie ihn nicht angeben.

```
cdk synth # app defines single stack
cdk deploy Happy Grumpy # app defines two or more stacks; two are deployed
```

Sie können auch die Platzhalter * (beliebige Anzahl von Zeichen) und ? (beliebiges einzelnes Zeichen) verwenden, um Stacks nach Mustern zu identifizieren. Wenn Sie Platzhalter verwenden, schließen Sie das Muster in Anführungszeichen ein. Andernfalls versucht die Shell möglicherweise, sie auf die Namen der Dateien im aktuellen Verzeichnis zu erweitern, bevor sie an das AWS CDK Toolkit übergeben werden.

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"  # PipeStack, LambdaStack, etc.
```

Tip

Sie müssen Stacks nicht explizit synthetisieren, bevor Sie sie bereitstellen. `cdk deploy` führt diesen Schritt aus, damit Sie sicherstellen können, dass Ihr neuester Code bereitgestellt wird.

Eine vollständige Dokumentation des `cdk` Befehls finden Sie unter [the section called “AWS CDK Toolkit”](#).

Arbeiten mit in AWS CDK Go

Go ist eine vollständig unterstützte Client-Sprache für das AWS Cloud Development Kit (AWS CDK) und gilt als stabil. Beim Arbeiten mit der AWS CDK in Go werden vertraute Tools verwendet. Die Go-Version des verwendet AWS CDK sogar Kennungen im Go-Stil.

Im Gegensatz zu den anderen Sprachen, die das CDK unterstützt, ist Go keine traditionelle objektorientierte Programmiersprache. Go verwendet Zusammensetzung, bei der andere Sprachen die Vererbung häufig nutzen. Wir haben versucht, idiomatische Go-Ansätze so weit wie möglich zu verwenden, aber es gibt Stellen, an denen sich das CDK unterscheiden kann.

Dieses Thema enthält Anleitungen zur Arbeit mit der AWS CDK in Go. Im [Blog-Beitrag zur Ankündigung](#) finden Sie eine exemplarische Vorgehensweise für ein einfaches Go-Projekt für die AWS CDK.

Themen

- [Erste Schritte mit Go](#)
- [Erstellen eines Projekts](#)
- [Verwalten von Modulen AWS der Construct Library](#)

- [Verwalten von Abhängigkeiten in Go](#)
- [AWS CDK idiomias in Go](#)
- [Erstellen, Synthetisieren und Bereitstellen](#)

Erste Schritte mit Go

Um mit der zu arbeiten AWS CDK, müssen Sie über ein - AWS Konto und Anmeldeinformationen verfügen und Node.js und das AWS CDK Toolkit installiert haben. Siehe [Erste Schritte mit der AWS CDK](#).

Die Go-Bindungen für die AWS CDK verwenden die standardmäßige [Go-Toolchain](#) , v1.18 oder höher. Sie können den Editor Ihrer Wahl verwenden.

Note

Sprachveralterung von Drittanbietern: Die Sprachversion wird nur unterstützt, bis ihr EOL (End of Life) vom Anbieter oder der Community gemeinsam genutzt wird und kann sich ohne vorherige Ankündigung ändern.

Erstellen eines Projekts

Sie erstellen ein neues AWS CDK Projekt, indem Sie `cdk init` in einem leeren Verzeichnis aufrufen. Verwenden Sie die `--language` Option und geben Sie `ango`:

```
mkdir my-project
cd my-project
cdk init app --language go
```

`cdk init` verwendet den Namen des Projektordners, um verschiedene Elemente des Projekts zu benennen, einschließlich Klassen, Unterordner und Dateien. Bindestriche im Ordnernamen werden in Unterstriche umgewandelt. Der Name sollte jedoch ansonsten der Form einer Go-ID folgen; er sollte beispielsweise nicht mit einer Zahl beginnen oder Leerzeichen enthalten.

Das resultierende Projekt enthält einen Verweis auf das AWS CDK Go-Kernmodul, github.com/aws/aws-cdk-go/awscdk/v2, in `go.mod`. Problem `go get` bei der Installation dieses und anderer erforderlicher Module.

Verwalten von Modulen AWS der Construct Library

In den meisten AWS CDK Dokumentationen und Beispielen wird das Wort „Modul“ häufig verwendet, um sich auf Module der AWS Construct Library zu beziehen, eines oder mehrere pro AWS Service, was sich von der idiomatischen Go-Nutzung des Begriffs unterscheidet. Die CDK-Konstruktbibliothek wird in einem Go-Modul mit den einzelnen Modulen der Konstruktbibliothek bereitgestellt, die die verschiedenen AWS -Services unterstützen, die als Go-Pakete in diesem Modul bereitgestellt werden.

Einige Services „AWS Construct Library“-Unterstützung befinden sich in mehr als einem Construct Library-Modul (Go-Paket). Amazon Route 53 verfügt beispielsweise über drei Construct Library-Module zusätzlich zum `awsroute53` Hauptpaket mit dem Namen `awsroute53patternsawsroute53resolver`, und `awsroute53targets`.

Das Kernpaket AWS CDK des , das Sie in den meisten AWS CDK Apps benötigen, wird im Go-Code als `import github.com/aws/aws-cdk-go/awscdk/v2`. Pakete für die verschiedenen Services in der AWS Construct Library finden Sie unter `github.com/aws/aws-cdk-go/awscdk/v2`. Der Namespace des Amazon S3-Moduls lautet beispielsweise `github.com/aws/aws-cdk-go/awscdk/v2/awss3`.

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

Nachdem Sie die Module der Construct Library (Go-Pakete) für die Services importiert haben, die Sie in Ihrer App verwenden möchten, greifen Sie in diesem Modul beispielsweise mit auf Konstrukte zu `awss3.Bucket`.

Verwalten von Abhängigkeiten in Go

In Go sind Abhängigkeitsversionen in `definiert go.mod`. Der Standardwert `go.mod` ähnelt dem hier gezeigten.

```
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
)
```

Paketnamen (Module, in Go Parlanze) werden durch URL mit der angehängten erforderlichen Versionsnummer angegeben. Das Modulsystem von Go unterstützt keine Versionsbereiche.

Geben Sie den `go get` Befehl aus, um alle erforderlichen Module zu installieren und zu aktualisierengo.mod. Eine Liste der verfügbaren Updates für Ihre Abhängigkeiten finden Sie unter `go list -m -u all`.

AWS CDK idiomias in Go

Feld- und Methodennamen

Feld- und Methodennamen verwenden die Groß-/Kleinschreibung (`likeThis`) in TypeScript, der Sprache des CDK als Ursprung. In Go folgen diese den Go-Konventionen, ebenso sind Pascal-verwaltet (`LikeThis`).

Bereinigen

Verwenden Sie in Ihrer `main` Methode `, defer jsii.Close()` um sicherzustellen, dass Ihre CDK-App nacheinander bereinigt wird.

Fehlende Werte und Zeigerkonvertierung

In Go werden fehlende Werte in AWS CDK Objekten wie Eigenschaftspaketen durch `dargestelltnil`. Go hat keine löschbaren Typen. Der einzige Typ, der enthalten kann, `nil` ist ein Zeiger. Damit Werte optional sind, sind alle CDK-Eigenschaften, Argumente und Rückgabewerte Zeiger, auch für primitive Typen. Dies gilt sowohl für erforderliche als auch für optionale Werte. Wenn ein erforderlicher Wert später optional wird, ist keine grundlegende Änderung des Typs erforderlich.

Verwenden Sie beim Übergeben von Literalwerten oder Ausdrücken die folgenden Hilfsfunktionen, um Zeiger auf die Werte zu erstellen.

- `jsii.String`
- `jsii.Number`
- `jsii.Bool`
- `jsii.Time`

Aus Gründen der Konsistenz empfehlen wir Ihnen, bei der Definition Ihrer eigenen Konstrukte Zeiger ähnlich zu verwenden, obwohl es praktischer erscheinen kann, beispielsweise die Ihres Konstrukts `id` als Zeichenfolge und nicht als Zeiger auf eine Zeichenfolge zu erhalten.

Wenn Sie mit optionalen AWS CDK Werten arbeiten, einschließlich primitiver Werte sowie komplexer Typen, sollten Sie die Zeiger explizit testen, um sicherzustellen, dass sie es nicht sind, `nil` bevor Sie mit ihnen etwas unternehmen. Go verfügt nicht über „syntaktische Telefonie“, um leere oder fehlende Werte zu behandeln, wie es bei anderen Sprachen der Fall ist. Es wird jedoch garantiert, dass die erforderlichen Werte in Eigenschaftspaketen und ähnlichen Strukturen vorhanden sind (die Konstruktion schlägt andernfalls fehl), sodass diese Werte nicht `nil` überprüft werden müssen.

Konstrukte und Eigenschaften

Konstrukte, die eine oder mehrere AWS Ressourcen und die zugehörigen Attribute darstellen, werden in Go als Schnittstellen dargestellt. Beispielsweise `awss3.Bucket` ist eine -Schnittstelle. Jedes Konstrukt verfügt über eine Factory-Funktion, z. B. `awss3.NewBucket`, um eine Struktur zurückzugeben, die die entsprechende Schnittstelle implementiert.

Alle Factory-Funktionen verwenden drei Argumente: die `scope` in der das Konstrukt definiert wird (sein übergeordnetes Element im Konstruktbaum), und `id`, `prop` sein Paket von Schlüssel-Wert-Paaren, die das Konstrukt zur Konfiguration der erstellten Ressourcen verwendet. Das Muster „Paket von Attributen“ wird auch an anderer Stelle in der verwendet AWS CDK.

In Go werden Eigenschaften durch einen bestimmten Strukturtyp für jedes Konstrukt dargestellt. Ein `awss3.Bucket` nimmt beispielsweise ein Eigenschaftsargument vom Typ `awss3.BucketProps`. Verwenden Sie ein Strukturliteral, um Props-Argumente zu schreiben.

```
var bucket = awss3.NewBucket(stack, jsii.String("MyBucket"), &awss3.BucketProps{
    Versioned: jsii.Bool(true),
})
```

Allgemeine Strukturen

An einigen Stellen AWS CDK verwendet die JavaScript Arrays oder nicht typisierte Objekte als Eingabe für eine Methode. (Siehe z. B. die [BuildSpec.fromObject\(\)](#) Methode AWS CodeBuild von `.`) In Go werden diese Objekte als Slices bzw. als leere Schnittstelle dargestellt.

Das CDK bietet variadische Hilfsfunktionen wie `jsii.Strings` zum Erstellen von Slices, die primitive Typen enthalten.

```
jsii.Strings("One", "Two", "Three")
```

Entwickeln von benutzerdefinierten Konstrukten

In Go ist es normalerweise einfacher, ein neues Konstrukt zu schreiben, als ein vorhandenes zu erweitern. Definieren Sie zunächst einen neuen Strukturtyp und betten Sie einen oder mehrere vorhandene Typen anonym ein, wenn eine erweiterungsähnliche Semantik gewünscht wird. Schreiben Sie Methoden für alle neuen Funktionen, die Sie hinzufügen, und die Felder, die für die Speicherung der benötigten Daten erforderlich sind. Definieren Sie eine Props-Schnittstelle, wenn Ihr Konstrukt eine benötigt. Schreiben Sie schließlich eine Factory-Funktion, `NewMyConstruct()` um eine Instance Ihres Konstrukts zurückzugeben.

Wenn Sie einfach einige Standardwerte für ein vorhandenes Konstrukt ändern oder ein einfaches Verhalten bei der Instanziierung hinzufügen, benötigen Sie nicht alle diese Tutorials. Schreiben Sie stattdessen eine Factory-Funktion, die die Factory-Funktion des Konstrukts aufruft, das Sie „erweitern“. In anderen CDK-Sprachen können Sie beispielsweise ein `TypedBucket` Konstrukt erstellen, das den Objekttyp in einem Amazon S3-Bucket erzwingt, indem der `s3.Bucket` Typ überschrieben und im Initialisierer Ihres neuen Typs eine Bucket-Richtlinie hinzugefügt wird, die nur das Hinzufügen bestimmter Dateinamenerweiterungen zum Bucket erlaubt. In Go ist es einfacher, einfach eine zu schreiben, `NewTypedBucket` die eine `s3.Bucket` (instanciert mit `s3.NewBucket`) zurückgibt, zu der Sie eine entsprechende Bucket-Richtlinie hinzugefügt haben. Es ist kein neuer Konstrukttyp erforderlich, da die Funktionalität bereits im Standard-Bucket-Konstrukt verfügbar ist. Das neue „Konstrukt“ bietet lediglich eine einfachere Möglichkeit, ihn zu konfigurieren.

Erstellen, Synthetisieren und Bereitstellen

Die kompiliert Ihre App AWS CDK automatisch, bevor sie ausgeführt wird. Es kann jedoch nützlich sein, Ihre App manuell zu erstellen, um nach Fehlern zu suchen und Tests durchzuführen. Sie können dies tun, indem Sie `go build` an einer Eingabeaufforderung ausgeben, während Sie sich im Stammverzeichnis Ihres Projekts befinden.

Führen Sie alle Tests aus, die Sie geschrieben haben, indem Sie `go test` an einer Eingabeaufforderung ausführen.

Die in Ihrer AWS CDK App definierten [Stacks](#) können mithilfe der folgenden Befehle einzeln oder zusammen synthetisiert und bereitgestellt werden. Im Allgemeinen sollten Sie sich im Hauptverzeichnis Ihres Projekts befinden, wenn Sie sie ausgeben.

- `cdk synth`: Stellt eine AWS CloudFormation Vorlage aus einem oder mehreren Stacks in Ihrer AWS CDK App zusammen.
- `cdk deploy`: Stellt die durch einen oder mehrere Stacks in Ihrer AWS CDK App definierten Ressourcen in bereit AWS.

Sie können die Namen mehrerer Stacks angeben, die in einem einzigen Befehl synthetisiert oder bereitgestellt werden sollen. Wenn Ihre App nur einen Stack definiert, müssen Sie ihn nicht angeben.

```
cdk synth                # app defines single stack
cdk deploy Happy Grumpy # app defines two or more stacks; two are deployed
```

Sie können auch die Platzhalter `*` (beliebige Anzahl von Zeichen) und `?` (beliebiges einzelnes Zeichen) verwenden, um Stacks nach Mustern zu identifizieren. Wenn Sie Platzhalter verwenden, schließen Sie das Muster in Anführungszeichen ein. Andernfalls versucht die Shell möglicherweise, sie auf die Namen der Dateien im aktuellen Verzeichnis zu erweitern, bevor sie an das AWS CDK Toolkit übergeben werden.

```
cdk synth "Stack?"      # Stack1, StackA, etc.
cdk deploy "*Stack"    # PipeStack, LambdaStack, etc.
```

Tip

Sie müssen Stacks nicht explizit synthetisieren, bevor Sie sie bereitstellen. `cdk deploy` führt diesen Schritt aus, damit Sie sicherstellen können, dass Ihr neuester Code bereitgestellt wird.

Eine vollständige Dokumentation des `cdk` Befehls finden Sie unter [the section called “AWS CDK Toolkit”](#).

Migrieren von AWS CDK v1 zu AWS CDK v2

Version 2 von AWS Cloud Development Kit (AWS CDK) ist darauf ausgelegt, das Schreiben von Infrastruktur als Code in Ihrer bevorzugten Programmiersprache zu erleichtern. In diesem Thema werden die Änderungen zwischen v1 und v2 des beschriebenen AWS CDK.

Tip

Um Stacks zu identifizieren, die mit AWS CDK v1 bereitgestellt werden, verwenden Sie das Dienstprogramm [awscdk-v1-stack-](#).

Die wichtigsten Änderungen von AWS CDK v1 zu CDK v2 sind die folgenden.

- AWS CDK v2 konsolidiert die stabilen Teile der AWS Construct Library, einschließlich der Core-Bibliothek, in einem einzigen Paket, `aws-cdk-lib`. Entwickler müssen keine zusätzlichen Pakete mehr für die einzelnen AWS Services installieren, die sie verwenden. Dieser Single-Package-Ansatz bedeutet auch, dass Sie die Versionen der verschiedenen CDK-Bibliothekspakete nicht synchronisieren müssen.

L1-Konstrukte (CfnXXXX), die die genauen Ressourcen darstellen, die in verfügbar sind AWS CloudFormation, gelten immer als stabil und sind daher in `aws-cdk-lib` enthalten.

- Experimentelle Module, bei denen wir noch mit der Community zusammenarbeiten, um neue [L2- oder L3-Konstrukte](#) zu entwickeln, sind nicht in `aws-cdk-lib` enthalten. Stattdessen werden sie als einzelne Pakete verteilt. Experimentelle Pakete werden mit einem `alpha` Suffix und einer semantischen Versionsnummer benannt. Die semantische Versionsnummer entspricht der ersten Version der AWS Konstruktbibliothek, mit der sie kompatibel sind, auch mit einem `alpha` Suffix. Konstrukte werden in `aws-cdk-lib` verschoben, nachdem sie als stabil eingestuft wurden, sodass die Construct Library die strenge semantische Versionsverwaltung einhalten kann.

Die Stabilität wird auf Serviceebene angegeben. Wenn wir beispielsweise mit der Erstellung eines oder mehrerer [L2-Konstrukte](#) für Amazon beginnen AppFlow, die bei diesem Schreiben nur L1-Konstrukte enthält, werden diese zuerst in einem Modul namens `aws-cdk/aws-appflow-alpha`. Dann wechseln sie zu `aws-cdk-lib` wenn wir der Meinung sind, dass die neuen Konstrukte den grundlegenden Anforderungen der Kunden entsprechen.

Sobald ein Modul als stabil eingestuft und in `aws-cdk-lib` integriert wurde, werden neue APIs unter Verwendung der im nächsten Aufzählungspunkt beschriebenen „BetaN“-Konvention hinzugefügt.

Mit jeder Version des wird eine neue Version jedes experimentellen Moduls veröffentlicht AWS CDK. In den meisten Fällen müssen sie jedoch nicht synchron gehalten werden. Sie können `aws-cdk-lib` oder das experimentelle Modul jederzeit aktualisieren. Die Ausnahme besteht darin, dass zwei oder mehr verwandte experimentelle Module dieselbe Version haben müssen, wenn sie voneinander abhängen.

- Bei stabilen Modulen, denen neue Funktionen hinzugefügt werden, erhalten neue APIs (unabhängig davon, ob es sich um vollständig neue Konstrukte oder neue Methoden oder Eigenschaften auf einem vorhandenen `Beta1` Konstrukt handelt) ein Suffix, während die Arbeit läuft. (Von `Beta2`, usw. befolgt `Beta3`, wenn grundlegende Änderungen erforderlich sind.) Eine Version der API ohne das Suffix wird hinzugefügt, wenn die API als stabil eingestuft wird. Alle Methoden außer den neuesten (ob Beta oder Final) sind dann veraltet.

Wenn wir beispielsweise `grantPower()` einem Konstrukt eine neue Methode hinzufügen, erscheint sie zunächst als `grantPowerBeta1()`. Wenn grundlegende Änderungen erforderlich sind (z. B. ein neuer erforderlicher Parameter oder eine neue Eigenschaft), würde die nächste Version der Methode den Namen `grantPowerBeta2()` usw. haben. Wenn die Arbeit abgeschlossen ist und die API abgeschlossen ist, wird die Methode `grantPower()` (ohne Suffix) hinzugefügt und die BetaN-Methoden sind veraltet.

Alle Beta-APIs verbleiben bis zur nächsten Hauptversion (3.0) in der Construct Library, und ihre Signaturen werden sich nicht ändern. Wenn Sie sie verwenden, werden Warnungen zur Veralterung angezeigt. Sie sollten daher so schnell wie möglich zur endgültigen Version der API wechseln. Keine zukünftigen AWS CDK 2.x-Versionen werden Ihre Anwendung jedoch beeinträchtigen.

- Die `Construct` Klasse wurde aus dem AWS CDK in eine separate Bibliothek zusammen mit verwandten Typen extrahiert. Dies geschieht, um die Bemühungen zur Anwendung des Construct Programming Model auf andere Domains zu unterstützen. Wenn Sie Ihre eigenen Konstrukte schreiben oder verwandte APIs verwenden, müssen Sie das `constructs` Modul als Abhängigkeit deklarieren und geringfügige Änderungen an Ihren Importen vornehmen. Wenn Sie erweiterte Funktionen verwenden, wie z. B. das Anbinden des CDK-App-Lebenszyklus, sind möglicherweise weitere Änderungen erforderlich. Ausführliche Informationen [finden Sie unter RFC](#).

- Veraltete Eigenschaften, Methoden und Typen in AWS CDK v1.x und seiner Construct Library wurden vollständig aus der CDK-v2-API entfernt. In den meisten unterstützten Sprachen erzeugen diese APIs Warnungen unter v1.x, sodass Sie möglicherweise bereits zu den Ersatz-APIs migriert haben. Eine vollständige [Liste der veralteten APIs](#) in CDK v1.x ist auf verfügbar GitHub.
- Verhalten, das durch Feature-Flags in AWS CDK v1.x ausgelöst wurde, ist in CDK v2 standardmäßig aktiviert. Die früheren Feature-Flags werden nicht mehr benötigt und werden in den meisten Fällen nicht unterstützt. Einige sind immer noch verfügbar, damit Sie unter sehr spezifischen Umständen zum CDK-v1-Verhalten zurückkehren können. Weitere Informationen finden Sie unter [the section called "Aktualisieren von Feature-Flags"](#).
- Mit CDK v2 müssen die Umgebungen, in denen Sie bereitstellen, mit dem modernen Bootstrap-Stack bootstrapped werden. Der Legacy-Bootstrap-Stack (Standard unter v1) wird nicht mehr unterstützt. CDK v2 erfordert außerdem eine neue Version des modernen Stacks. Um Ihre vorhandenen Umgebungen zu aktualisieren, starten Sie sie neu. Es ist nicht mehr erforderlich, Feature-Flags oder Umgebungsvariablen festzulegen, um den modernen Bootstrap-Stack zu verwenden.

Important

Die moderne Bootstrap-Vorlage gewährt effektiv die Berechtigungen `--cloudformation-execution-policies`, die von der für jedes AWS Konto in der `--trust` Liste impliziert werden. Standardmäßig erweitert dies die Berechtigungen zum Lesen und Schreiben in jede Ressource im Bootstrapped-Konto. Stellen Sie sicher, dass Sie [den Bootstrapping-Stack mit Richtlinien und vertrauenswürdigen Konten konfigurieren](#), mit denen Sie vertraut sind.

Neue Voraussetzungen

Die meisten Anforderungen für AWS CDK v2 sind dieselben wie für AWS CDK v1.x. Zusätzliche Anforderungen sind hier aufgeführt.

- Für TypeScript Entwickler ist TypeScript 3.8 oder höher erforderlich.
- Eine neue Version des CDK Toolkits ist für die Verwendung mit CDK v2 erforderlich. Da CDK v2 jetzt allgemein verfügbar ist, ist v2 die Standardversion bei der Installation des CDK Toolkits. Es ist abwärtskompatibel mit CDK-v1-Projekten, sodass Sie die frühere Version nur installiert lassen müssen, wenn Sie CDK-v1-Projekte erstellen möchten. Um ein Upgrade durchzuführen, geben Sie `ausnpm install -g aws-cdk`.

Upgrade von der AWS CDK v2-Entwicklervorschau

Wenn Sie die CDK-v2-Entwicklervorschau verwenden, haben Sie in Ihrem Projekt Abhängigkeiten von einer Release-Candidate-Version des AWS CDK, z. B. `2.0.0-rc1`. Aktualisieren Sie diese auf `2.0.0` und aktualisieren Sie dann die in Ihrem Projekt installierten Module.

TypeScript

```
npm install oder yarn install
```

JavaScript

```
npm install oder yarn install
```

Python

```
python -m pip install -r requirements.txt
```

Java

```
mvn package
```

C#

```
dotnet restore
```

Go

```
go get
```

Nachdem Sie Ihre Abhängigkeiten aktualisiert haben, geben Sie ein, `npm update -g aws-cdk` um das CDK Toolkit auf die Release-Version zu aktualisieren.

Migrieren von AWS CDK v1 zu CDK v2

Um Ihre App zu AWS CDK v2 zu migrieren, aktualisieren Sie zunächst die Feature-Flags in `cdk.json`. Aktualisieren Sie dann die Abhängigkeiten und Importe Ihrer App nach Bedarf für die Programmiersprache, in der sie geschrieben ist.

Aktualisieren auf ein aktuelles v1

Wir sehen in einem Schritt eine Reihe von Kunden, die von einer alten Version von AWS CDK v1 auf die neueste Version von v2 aktualisieren. Es ist zwar durchaus möglich, dies zu tun, aber Sie würden sowohl ein Upgrade über mehrere Jahre hinweg vornehmen (was möglicherweise nicht alle die gleiche Menge an Entwicklungstests hatten, die wir heute haben), als auch ein Upgrade auf Versionen mit neuen Standardeinstellungen und einer anderen Codeorganisation durchführen.

Um ein sicheres Upgrade zu ermöglichen und die Quellen unerwarteter Änderungen leichter zu diagnostizieren, empfehlen wir Ihnen, diese beiden Schritte zu trennen: zuerst auf die neueste Version aktualisieren und anschließend den Wechsel zu v2 durchführen.

Aktualisieren von Feature-Flags

Entfernen Sie die folgenden v1-Feature-Flags aus `cdk.json` wenn sie vorhanden sind, da diese alle standardmäßig in AWS CDK v2 aktiv sind. Wenn ihr alter Effekt für Ihre Infrastruktur wichtig ist, müssen Sie Änderungen am Quellcode vornehmen. Weitere Informationen finden Sie [in der Liste der Flags auf GitHub](#).

- `@aws-cdk/core:enableStackNameDuplicates`
- `aws-cdk:enableDiffNoFail`
- `@aws-cdk/aws-ecr-assets:dockerIgnoreSupport`
- `@aws-cdk/aws-secretsmanager:parseOwnedSecretName`
- `@aws-cdk/aws-kms:defaultKeyPolicies`
- `@aws-cdk/aws-s3:grantWriteWithoutAcl`
- `@aws-cdk/aws-efs:defaultEncryptionAtRest`

Eine Handvoll v1-Feature-Flags können auf `false` gesetzt werden, um zu bestimmten AWS CDK v1-Verhaltensweisen zurückzukehren. Eine vollständige Referenz finden Sie unter [the section called “Zurücksetzen auf das Verhalten v1”](#) oder in der Liste GitHub auf [GitHub](#).

Verwenden Sie für beide Arten von Flags den `cdk diff` Befehl, um die Änderungen an Ihrer synthetisierten Vorlage zu überprüfen, um festzustellen, ob sich die Änderungen an einem dieser Flags auf Ihre Infrastruktur auswirken.

Kompatibilität mit CDK Toolkit

CDK v2 erfordert v2 oder höher des CDK Toolkits. Diese Version ist abwärtskompatibel mit CDK-v1-Apps. Daher können Sie eine einzelne global installierte Version des CDK Toolkits mit all Ihren AWS CDK Projekten verwenden, unabhängig davon, ob sie v1 oder v2 verwenden. Eine Ausnahme besteht darin, dass CDK Toolkit v2 nur CDK-v2-Projekte erstellt.

Wenn Sie sowohl v1- als auch v2-CDK-Projekte erstellen müssen, installieren Sie CDK Toolkit v2 nicht global. (Entfernen Sie es, wenn Sie es bereits installiert haben: `npm remove -g aws-cdk`.) Um das CDK Toolkit aufzurufen, verwenden Sie `npx` um v1 oder v2 des CDK Toolkits wie gewünscht auszuführen.

```
npx aws-cdk@1.x init app --language typescript
npx aws-cdk@2.x init app --language typescript
```

Tip

Richten Sie Befehlszeilen-Aliase ein, damit Sie die `cdk1` Befehle `cdk` und verwenden können, um die gewünschte Version des CDK Toolkits aufzurufen.

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 $*
```

Aktualisieren von Abhängigkeiten und Importen

Aktualisieren Sie die Abhängigkeiten Ihrer App und installieren Sie dann die neuen Pakete. Aktualisieren Sie abschließend die Importe in Ihrem Code.

TypeScript

Anwendungen

Aktualisieren Sie für CDK-Apps `package.json` wie folgt. Entfernen Sie Abhängigkeiten von einzelnen stabilen Modulen im v1-Stil und legen Sie die niedrigste Version von fest, die `aws-cdk-lib` Sie für Ihre Anwendung benötigen (hier 2.0.0).

Experimentelle Konstrukte werden in separaten, unabhängig versionierten Paketen mit Namen bereitgestellt, die mit `alpha` enden, und einer Alpha-Versionsnummer. Die Alpha-Versionsnummer entspricht der ersten Version von `aws-cdk-lib` mit der sie kompatibel sind. Hier haben wir `aws-codestar` auf `v2.0.0-alpha.1` angeheftet.

```
{
  "dependencies": {
    "aws-cdk-lib": "^2.0.0",
    "@aws-cdk/aws-codestar-alpha": "2.0.0-alpha.1",
    "constructs": "^10.0.0"
  }
}
```

Bibliotheken erstellen

Richten Sie für Konstruktbibliotheken die niedrigste Version von ein, die `aws-cdk-lib` Sie für Ihre Anwendung benötigen (hier 2.0.0), und aktualisieren Sie `package.json` wie folgt.

Beachten Sie, dass sowohl als Peer-Abhängigkeit als auch als Entwicklungsabhängigkeit `aws-cdk-lib` angezeigt wird.

```
{
  "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

Sie sollten einen Sprung der Hauptversion für die Versionsnummer Ihrer Bibliothek durchführen, wenn Sie eine v2-kompatible Bibliothek veröffentlichen, da dies eine

grundlegende Änderung für Bibliotheksverbraucher ist. Es ist nicht möglich, sowohl CDK v1 als auch v2 mit einer einzigen Bibliothek zu unterstützen. Um weiterhin Kunden zu unterstützen, die v1 noch verwenden, können Sie die frühere Version parallel beibehalten oder ein neues Paket für v2 erstellen.

Es liegt an Ihnen, wie lange Sie v1 AWS CDK -Kunden weiterhin unterstützen möchten. Sie könnten Ihr Signal aus dem Lebenszyklus von CDK v1 selbst ziehen, das am 1. Juni 2022 in die Wartung aufgenommen wurde und end-of-life am 1. Juni 2023 erreichen wird. Ausführliche Informationen finden Sie unter [AWS CDK Wartungsrichtlinie](#).

Sowohl Bibliotheken als auch Apps

Installieren Sie die neuen Abhängigkeiten, indem Sie `npm install` oder `ausführenyarn install`.

Ändern Sie Ihre Importe, um sie Construct aus dem neuen constructs Modul, Kerntypen wie App und Stack von der obersten Ebene von zu importieren, und stabile Module der Construct Library für die Servicesaws-cdk-lib, die Sie aus Namespaces unter verwendenaws-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

Aktualisieren Sie `package.json` wie folgt. Entfernen Sie Abhängigkeiten von einzelnen stabilen Modulen im v1-Stil und legen Sie die niedrigste Version von fest, die `aws-cdk-lib` Sie für Ihre Anwendung benötigen (hier 2.0.0).

Experimentelle Konstrukte werden in separaten, unabhängig versionierten Paketen mit Namen bereitgestellt, die mit `alpha` und einer Alpha-Versionsnummer. Die Alpha-Versionsnummer entspricht der ersten Version von `aws-cdk-lib` mit der sie kompatibel sind. Hier haben wir `aws-codestar` auf `v2.0.0-alpha.1` angeheftet.

```
{
  "dependencies": {
    "aws-cdk-lib": "^2.0.0",
    "@aws-cdk/aws-codestar-alpha": "2.0.0-alpha.1",
```

```
    "constructs": "^10.0.0"  
  }  
}
```

Installieren Sie die neuen Abhängigkeiten, indem Sie `npm install` oder `ausführen yarn install`.

Ändern Sie die Importe Ihrer App, um Folgendes zu tun:

- Importieren `Construct` aus dem neuen `constructs` Modul
- Importieren von Kerntypen wie `App` und `Stack` von der obersten Ebene von `aws-cdk-lib`
- Importieren von Modulen der AWS Construct Library aus Namespaces unter `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

Aktualisieren Sie `requirements.txt` oder die `install_requires` Definition in `setup.py` wie folgt. Entfernen Sie Abhängigkeiten von einzelnen stabilen Modulen im v1-Stil.

Experimentelle Konstrukte werden in separaten, unabhängig versionierten Paketen mit Namen bereitgestellt, die mit `alpha` und einer Alpha-Versionsnummer. Die Alpha-Versionsnummer entspricht der ersten Version von `aws-cdk-lib` mit der sie kompatibel sind. Hier haben wir `aws-codestar` auf `v2.0.0alpha1` angeheftet.

```
install_requires=[  
    "aws-cdk-lib>=2.0.0",  
    "constructs>=10.0.0",  
    "aws-cdk.aws-codestar-alpha>=2.0.0alpha1",  
    # ...  
],
```

Tip

Deinstallieren Sie alle anderen Versionen von AWS CDK Modulen, die bereits in der virtuellen Umgebung Ihrer App installiert sind, mithilfe von `pip uninstall`.

```
Installieren Sie dann die neuen Abhängigkeiten mit python -m pip install -r requirements.txt.
```

Ändern Sie die Importe Ihrer App, um Folgendes zu tun:

- Importieren Construct aus dem neuen constructs Modul
- Kerntypen wie App und aus der Stackobersten Ebene von importieren aws_cdk
- Importieren von Modulen der AWS Konstruktbibliothek aus Namespaces unter 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

Entfernen Sie `software.amazon.awscdk` Abhängigkeiten für stabile Module und ersetzen Sie sie durch Abhängigkeiten von `software.constructs` (für `Construct`) und `software.amazon.awscdk`.

Experimentelle Konstrukte werden in separaten, unabhängig versionierten Paketen mit Namen bereitgestellt, die mit `alpha` und einer Alpha-Versionsnummer. Die Alpha-Versionsnummer entspricht der ersten Version von `aws-cdk-lib` mit der sie kompatibel sind. Hier haben wir `aws-codestar` auf `v2.0.0-alpha.1` angeheftet.

```
<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>
```

Installieren Sie die neuen Abhängigkeiten, indem Sie ausführen `mvn package`.

Ändern Sie Ihren Code, um Folgendes zu tun:

- Importieren Construct aus der neuen `software.constructs` Bibliothek
- Importieren von Kernklassen wie `Stack` und `App` aus `software.amazon.awscdk`
- Importieren von Servicekonstrukten aus `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#

Die einfachste Möglichkeit, die Abhängigkeiten einer C#-CDK-Anwendung zu aktualisieren, besteht darin, die `.csproj` Datei manuell zu bearbeiten. Entfernen Sie alle stabilen `Amazon.CDK.*` Paketreferenzen und ersetzen Sie sie durch Verweise auf die `Constructs` Pakete `Amazon.CDK.Lib` und `.`

Experimentelle Konstrukte werden in separaten, unabhängig versionierten Paketen mit Namen bereitgestellt, die mit `alpha` enden und einer Alpha-Versionsnummer. Die Alpha-Versionsnummer entspricht der ersten Version von `aws-cdk-lib` mit der sie kompatibel sind. Hier haben wir `aws-codestar` auf `v2.0.0-alpha.1` angeheftet.

```
<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" />
```

Installieren Sie die neuen Abhängigkeiten, indem Sie ausführendotnet `restore`.

Ändern Sie die Importe in Ihren Quelldateien wie folgt.

```
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

Geben Sie `ingo get`, um Ihre Abhängigkeiten auf die neueste Version zu aktualisieren, und aktualisieren Sie die `-.modDatei` Ihres Projekts.

Testen Ihrer migrierten App vor der Bereitstellung

Bevor Sie Ihre Stacks bereitstellen, verwenden Sie `cdk diff` um nach unerwarteten Änderungen an den Ressourcen zu suchen. Änderungen an logischen IDs (wodurch Ressourcen ersetzt werden) werden nicht erwartet.

Zu den erwarteten Änderungen gehören unter anderem:

- Änderungen an der `CDKMetadata` Ressource.
- Aktualisierte Komponenten-Hashes.
- Änderungen im Zusammenhang mit der Stack-Synthetik im neuen Stil. Gilt, wenn Ihre App den Legacy-Stack-Syntheizer in v1 verwendet hat. (CDK v2 unterstützt den Legacy-Stack-Syntheizer nicht.)
- Das Hinzufügen einer `CheckBootstrapVersion` Regel.

Unerwartete Änderungen werden in der Regel nicht durch ein Upgrade auf AWS CDK v2 selbst verursacht. In der Regel sind sie das Ergebnis veralteten Verhaltens, das zuvor durch Feature-Flags geändert wurde. Dies ist ein Problem des Upgrades von einer Version von CDK vor etwa 1.85.x. Sie würden dieselben Änderungen sehen, die auf die neueste Version v1.x aktualisiert wurden. Sie können dies normalerweise wie folgt lösen:

1. Aktualisieren Sie Ihre App auf die neueste Version v1.x

2. Entfernen von Feature-Flags
3. Überarbeiten Sie Ihren Code nach Bedarf
4. Bereitstellen
5. Upgrade auf v2

Note

Wenn Ihre aktualisierte App nach dem zweistufigen Upgrade nicht bereitgestellt werden kann, [melden Sie das Problem](#).

Wenn Sie bereit sind, die Stacks in Ihrer App bereitzustellen, sollten Sie zuerst eine Kopie bereitstellen, damit Sie sie testen können. Der einfachste Weg, dies zu tun, besteht darin, es in einer anderen Region bereitzustellen. Sie können jedoch auch die IDs Ihrer Stacks ändern. Stellen Sie nach dem Testen sicher, dass Sie die Testkopie mit `löschendk destroy`.

Fehlerbehebung

TypeScript **'from' expected - oder -';' expected**Fehler bei Importen

Aktualisieren Sie auf TypeScript 3.8 oder höher.

Ausführen von „cdk bootstrap“

Wenn Sie einen Fehler wie den folgenden sehen:

```
# MyStack failed: Error: MyStack: SSM parameter /cdk-bootstrap/hnb659fds/version not
found. Has the environment been bootstrapped? Please run 'cdk bootstrap' (see https://
docs.aws.amazon.com/cdk/latest/guide/bootstrapping.html)
  at CloudFormationDeployments.validateBootstrapStackVersion (.../aws-cdk/lib/api/
cloudformation-deployments.ts:323:13)
  at processTicksAndRejections (internal/process/task_queues.js:97:5)
MyStack: SSM parameter /cdk-bootstrap/hnb659fds/version not found. Has the environment
been bootstrapped? Please run 'cdk bootstrap' (see https://docs.aws.amazon.com/cdk/
latest/guide/bootstrapping.html)
```

AWS CDK v2 erfordert einen aktualisierten Bootstrap-Stack, und außerdem benötigen alle v2-Bereitstellungen Bootstrap-Ressourcen. (Mit v1 können Sie einfache Stacks ohne Bootstrapping bereitstellen.) Vollständige Details finden Sie unter [the section called “Bootstrapping”](#).

Suchen von v1-Stacks

Bei der Migration Ihrer CDK-Anwendung von v1 zu v2 möchten Sie möglicherweise die bereitgestellten AWS CloudFormation Stacks identifizieren, die mit v1 erstellt wurden. Führen Sie dazu den folgenden Befehl aus:

```
npx awscdk-v1-stack-finder
```

Einzelheiten zur Verwendung finden Sie unter awscdk-v1-stack- [README](#).

Migrieren Sie bestehende Ressourcen und AWS CloudFormation Vorlagen auf die AWS CDK

Die CDK Migrate-Funktion befindet sich in der Vorschauversion für AWS CDK und kann sich ändern.

Verwenden Sie die AWS Cloud Development Kit (AWS CDK) Befehlszeilenschnittstelle (AWS CDK CLI), um bereitgestellte AWS Ressourcen, bereitgestellte AWS CloudFormation Stacks und lokale AWS CloudFormation Vorlagen zu migrieren. AWS CDK

Themen

- [Wie funktioniert die Migration](#)
- [Vorteile von CDK Migrate](#)
- [Überlegungen](#)
- [Voraussetzungen](#)
- [Beginnen Sie mit CDK Migrate](#)
- [Migrieren Sie von einem AWS CloudFormation Stack](#)
- [Aus einer AWS CloudFormation Vorlage migrieren](#)
- [Migrieren Sie von bereitgestellten Ressourcen](#)
- [Verwalten und implementieren Sie Ihre CDK-App](#)

Wie funktioniert die Migration

Verwenden Sie den AWS CDK CLI `cdk migrate` Befehl, um aus den folgenden Quellen zu migrieren:

- Bereitgestellte AWS Ressourcen.
- Bereitgestellte AWS CloudFormation Stapel.
- Lokale AWS CloudFormation Vorlagen.

Eingesetzte AWS Ressourcen

Sie können bereitgestellte AWS Ressourcen aus einer bestimmten Umgebung (AWS-Konto und AWS-Region) migrieren, die keinem AWS CloudFormation Stack zugeordnet sind.

Der AWS CDK CLI verwendet den IaC-Generator-Service, um in Ihrer AWS Umgebung nach Ressourcen zu suchen und Ressourcendetails zu sammeln. Weitere Informationen zum IaC-Generator finden Sie im Benutzerhandbuch unter [Generieren von Vorlagen für vorhandene Ressourcen](#).AWS CloudFormation

Nach dem Sammeln der Ressourcendetails AWS CDK CLI erstellt der eine neue CDK-App, die einen einzelnen Stapel mit Ihren migrierten Ressourcen enthält.

Bereitgestellte Stapel AWS CloudFormation

Sie können einen einzelnen AWS CloudFormation Stack in eine neue AWS CDK App migrieren. Das AWS CDK CLI ruft die AWS CloudFormation Vorlage Ihres Stacks ab und erstellt eine neue CDK-App. Die CDK-App wird aus einem einzigen Stack bestehen, der Ihren migrierten AWS CloudFormation Stack enthält.

Lokale Vorlagen AWS CloudFormation

Sie können von einer lokalen AWS CloudFormation Vorlage aus migrieren. Lokale Vorlagen können bereitgestellte Ressourcen enthalten oder auch nicht. Das AWS CDK CLI erstellt eine neue CDK-App, die einen einzelnen Stapel mit Ihren Ressourcen enthält.

Nach der Migration können Sie Ihre CDK-App verwalten, ändern und bereitstellen, um Ihre Ressourcen bereitzustellen oder AWS CloudFormation zu aktualisieren.

Vorteile von CDK Migrate

Die Migration von Ressourcen zu AWS CDK war in der Vergangenheit ein manueller Prozess, der Zeit und Fachwissen erforderte, AWS CDK bis er überhaupt AWS CloudFormation begonnen werden konnte. Mit CDK Migrate wird Ihnen ein Großteil des Migrationsaufwands in einem Bruchteil der Zeit AWS CDK CLI erleichtert. Mit CDK Migrate können Sie schnell mit der AWS CDK Entwicklung und Verwaltung neuer und vorhandener Anwendungen beginnen. AWS

Überlegungen

Allgemeine Überlegungen

CDK Migrate im Vergleich zu CDK Import

Der `cdk import` Befehl kann bereitgestellte Ressourcen in eine neue oder bestehende CDK-App importieren. Beim Import muss jede Ressource manuell als L1-Konstrukt in Ihrer App definiert werden. Wir empfehlen `cdk import` die Verwendung, um eine oder mehrere Ressourcen gleichzeitig in eine neue oder bestehende CDK-App zu importieren. Weitere Informationen hierzu finden Sie unter [Importieren vorhandener Ressourcen in einen Stack](#).

Der `cdk migrate` Befehl migriert von bereitgestellten Ressourcen, bereitgestellten AWS CloudFormation Stacks oder lokalen AWS CloudFormation Vorlagen in eine neue CDK-App. Während der Migration werden Ihre Ressourcen in die AWS CDK CLI neue CDK-App importiert. `cdk import` Das generiert AWS CDK CLI außerdem L1-Konstrukte für jede Ressource für Sie. Wir empfehlen die Verwendung `cdk migrate` beim Import aus einer unterstützten Migrationsquelle in eine neue AWS CDK App.

CDK Migrate erstellt nur L1-Konstrukte

Die neu erstellte CDK-App wird nur L1-Konstrukte enthalten. Sie können Ihrer App nach der Migration Konstrukte auf höherer Ebene hinzufügen.

CDK Migrate erstellt CDK-Apps, die einen einzigen Stack enthalten

Die neu erstellte CDK-App wird einen einzelnen Stapel enthalten.

Bei der Migration bereitgestellter Ressourcen werden alle migrierten Ressourcen in einem einzigen Stack in der neuen CDK-App enthalten sein.

Bei der Migration von AWS CloudFormation Stacks können Sie in der neuen CDK-App nur einen einzelnen AWS CloudFormation Stack in einen einzigen Stack migrieren.

Ressourcen migrieren

Projektelemente, wie z. B. AWS Lambda Code, werden nicht direkt in die neue CDK-App migriert. Nach der Migration können Sie Asset-Werte angeben, um sie in die CDK-App aufzunehmen.

Zustandsbehaftete Ressourcen migrieren

Wenn Sie statusbehaftete Ressourcen wie Datenbanken und Amazon Simple Storage Service (Amazon S3) -Buckets migrieren, möchten Sie in den meisten Fällen die vorhandene Ressource migrieren, anstatt eine neue Ressource zu erstellen.

Gehen Sie wie folgt vor, um statusbehaftete Ressourcen zu migrieren und beizubehalten:

- Stellen Sie sicher, dass Ihre statusbehaftete Ressource den Import unterstützt. Weitere Informationen finden Sie im AWS CloudFormation Benutzerhandbuch unter [Unterstützung von Ressourcentypen](#).
- Stellen Sie nach der Migration sicher, dass die logische ID der migrierten Ressource in der neuen CDK-App mit der logischen ID der bereitgestellten Ressource übereinstimmt.
- Wenn Sie von einem AWS CloudFormation Stack migrieren, stellen Sie sicher, dass der Stack-Name in der neuen CDK-App mit dem Stack übereinstimmt. AWS CloudFormation
- Stellen Sie die CDK-App mit demselben AWS Konto und derselben AWS-Region migrierten Ressource bereit.

Überlegungen bei der Migration von einer Vorlage AWS CloudFormation

CDK Migrate unterstützt die Migration einzelner Vorlagen

Bei der Migration von AWS CloudFormation Vorlagen können Sie eine einzelne Vorlage für die Migration auswählen. Verschachtelte Vorlagen werden nicht unterstützt.

Migrieren von Vorlagen mit systemeigenen Funktionen

Bei der Migration von einer AWS CloudFormation Vorlage, die systeminterne Funktionen verwendet, AWS CDK CLI wird versucht, Ihre Logik mit der Klasse in die CDK-App zu migrieren. Für weitere Informationen finden Sie unter [Klasse Fn](#) in der API-Referenz. AWS Cloud Development Kit (AWS CDK)

Überlegungen bei der Migration von bereitgestellten Ressourcen

Einschränkungen beim Scannen

Beim Scannen Ihrer Umgebung nach Ressourcen unterliegt der IaC-Generator bestimmten Einschränkungen hinsichtlich der Daten, die er abrufen kann, und Beschränkungen bezüglich

der Kontingente beim Scannen. Weitere Informationen finden Sie unter [Überlegungen](#) im AWS CloudFormation Benutzerhandbuch.

Voraussetzungen

Gehen Sie wie folgt vor, bevor Sie den `cdk migrate` Befehl verwenden:

1. Richten Sie die Authentifizierung mit ein AWS. Anweisungen finden Sie unter [Schritt 2: Programmgesteuerten Zugriff konfigurieren](#).
2. Installieren oder aktualisieren Sie das AWS CDK CLI. Installationsanweisungen finden Sie unter [Schritt 3: Installieren der AWS CDKCLI](#).

Beginnen Sie mit CDK Migrate

Führen Sie zunächst den AWS CDK CLI `cdk migrate` Befehl in einem Verzeichnis Ihrer Wahl aus. Geben Sie je nach Art der Migration, die Sie durchführen, die erforderlichen und optionalen Optionen an.

Eine vollständige Liste und Beschreibung der Optionen, die Sie zusammen verwenden können `cdk migrate`, finden Sie unter [cdk migrate](#).

Im Folgenden sind einige wichtige Optionen aufgeführt, die Sie möglicherweise bereitstellen möchten.

Stack name

Die einzige erforderliche Option ist `--stack-name`. Verwenden Sie diese Option, um einen Namen für den Stack anzugeben, der nach der Migration in der AWS CDK App erstellt wird. Der Stack-Name wird bei der Bereitstellung auch als Name Ihres AWS CloudFormation Stacks verwendet.

Sprache

Wird verwendet `--language`, um die Programmiersprache der neuen CDK-App anzugeben.

AWS Konto und AWS-Region

Das AWS CDK CLI ruft AWS Konto und AWS-Region Informationen aus Standardquellen ab. Weitere Informationen finden Sie unter [Schritt 2: Programmgesteuerten Zugriff konfigurieren](#). Sie können die `--region` Optionen `--account` und mit verwenden `cdk migrate`, um andere Werte anzugeben.

Ausgabeverzeichnis Ihres neuen CDK-Projekts

Standardmäßig erstellt AWS CDK CLI das ein neues CDK-Projekt in Ihrem Arbeitsverzeichnis und verwendet den von Ihnen angegebenen Wert, um den `--stack-name` Projektordner zu benennen. Wenn derzeit ein Ordner mit demselben Namen existiert, AWS CDK CLI wird dieser Ordner überschrieben.

Mit der Option können Sie einen anderen Ausgabepfad für den neuen CDK-Projektordner angeben. `--output-path`

Migrationsquelle

Geben Sie eine Option an, um die Quelle anzugeben, von der Sie migrieren.

- `--from-path`— Migrieren Sie von einer lokalen AWS CloudFormation Vorlage.
- `--from-scan`— Migrieren Sie von bereitgestellten Ressourcen in einem AWS Konto und AWS-Region.
- `--from-stack`— Migrieren Sie von einem AWS CloudFormation Stack aus.

Abhängig von Ihrer Migrationsquelle können Sie zusätzliche Optionen zur Anpassung des `cdk migrate` Befehls bereitstellen.

Migrieren Sie von einem AWS CloudFormation Stack

Um von einem bereitgestellten AWS CloudFormation Stack zu migrieren, geben Sie die `--from-stack` Option an. Geben Sie den Namen Ihres bereitgestellten AWS CloudFormation Stacks mit ein `--stack-name`. Im Folgenden wird ein Beispiel gezeigt:

```
$ cdk migrate --from-stack --stack-name "myCloudFormationStack"
```

Das AWS CDK CLI wird Folgendes tun:

1. Rufen Sie die AWS CloudFormation Vorlage Ihres bereitgestellten Stacks ab.
2. Führen Sie `cdk init`, um eine neue CDK-App zu initialisieren.
3. Erstellen Sie in der CDK-App einen Stack, der Ihren migrierten Stack enthält. AWS CloudFormation

Wenn Sie von einem bereitgestellten AWS CloudFormation Stack migrieren, wird AWS CDK CLI versucht, die logischen IDs der bereitgestellten Ressourcen und den Namen des bereitgestellten

AWS CloudFormation Stacks den migrierten Ressourcen und dem Stack in der neuen CDK-App zuzuordnen.

Nach der Migration können Sie Ihre CDK-App normal verwalten und ändern. Bei der Bereitstellung AWS CloudFormation wird die Bereitstellung aufgrund des passenden AWS CloudFormation Stack-Namens als AWS CloudFormation Stack-Update identifiziert. Ressourcen mit übereinstimmenden logischen IDs werden aktualisiert. Weitere Informationen zur Bereitstellung finden Sie unter [Verwalten und implementieren Sie Ihre CDK-App](#).

Aus einer AWS CloudFormation Vorlage migrieren

CDK Migrate unterstützt die Migration von AWS CloudFormation Vorlagen, die in oder formatiert sind. JSON YAML

Um von einer lokalen AWS CloudFormation Vorlage zu migrieren, verwenden Sie die `--from-path` Option und geben Sie einen Pfad zur lokalen Vorlage an. Sie müssen auch die erforderliche `--stack-name` Option angeben. Im Folgenden wird ein Beispiel gezeigt:

```
$ cdk migrate --from-path "./template.json" --stack-name "myCloudFormationStack"
```

Sie AWS CDK CLI werden Folgendes tun:

1. Rufen Sie Ihre lokale AWS CloudFormation Vorlage ab.
2. Führen Sie `auscdk init`, um eine neue CDK-App zu initialisieren.
3. Erstellen Sie in der CDK-App einen Stack, der Ihre migrierte Vorlage enthält. AWS CloudFormation

Nach der Migration können Sie Ihre CDK-App normal verwalten und ändern. Bei der Bereitstellung haben Sie die folgenden Optionen:

- AWS CloudFormation Stack aktualisieren — Wenn die lokale AWS CloudFormation Vorlage zuvor bereitgestellt wurde, können Sie den bereitgestellten AWS CloudFormation Stack aktualisieren.
- Einen neuen AWS CloudFormation Stack bereitstellen — Wenn die lokale AWS CloudFormation Vorlage nie bereitgestellt wurde oder wenn Sie einen neuen Stack aus einer zuvor bereitgestellten Vorlage erstellen möchten, können Sie einen neuen AWS CloudFormation Stack bereitstellen.

Migrieren Sie von einer AWS SAM Vorlage

Um von einer AWS Serverless Application Model (AWS SAM) -Vorlage zu migrieren, müssen Sie sie zuerst in eine AWS CloudFormation Vorlage konvertieren oder bereitstellen, um einen AWS CloudFormation Stack zu erstellen.

Um eine AWS SAM Vorlage in zu konvertieren AWS CloudFormation, können Sie den AWS SAM CLI `sam validate --debug` Befehl verwenden. Möglicherweise müssen Sie `false` in Ihrer `samconfig.toml` Datei `lint` auf einstellen, bevor Sie diesen Befehl ausführen können.

Um in einen AWS CloudFormation Stapel zu konvertieren, stellen Sie die AWS SAM Vorlage mithilfe von bereit AWS SAM CLI. Migrieren Sie dann vom bereitgestellten Stack.

Migrieren Sie von bereitgestellten Ressourcen

Um von bereitgestellten AWS Ressourcen zu migrieren, geben Sie die `--from-scan` Option an. Sie müssen auch die erforderliche `--stack-name` Option angeben. Im Folgenden wird ein Beispiel gezeigt:

```
$ cdk migrate --from-scan --stack-name "myCloudFormationStack"
```

Sie AWS CDK CLI werden Folgendes tun:

1. Scannen Sie Ihr Konto nach Ressourcen- und Immobiliendetails - Der AWS CDK CLI verwendet den IaC-Generator, um Ihr Konto zu scannen und Details zu sammeln.
2. AWS CloudFormation Vorlage generieren — Nach dem Scannen AWS CDK CLI verwendet der IaC-Generator, um eine Vorlage zu erstellen. AWS CloudFormation
3. Initialisieren Sie eine neue CDK-App und migrieren Sie Ihre Vorlage — Das AWS CDK CLI führt `cdk init` zur Initialisierung einer neuen AWS CDK App durch und migriert Ihre AWS CloudFormation Vorlage in die CDK-App als einen einzigen Stapel.

Verwenden Sie Filter

Standardmäßig AWS CDK CLI wird die gesamte AWS Umgebung gescannt und Ressourcen bis zur maximalen Kontingentgrenze des IaC-Generators migriert. Sie können Filter mit dem angeben AWS CDK CLI, nach welchen Kriterien Ressourcen von Ihrem Konto zur neuen CDK-App migriert werden. Weitere Informationen hierzu finden Sie unter [--filter](#).

Mit dem IaC-Generator nach Ressourcen suchen

Abhängig von der Anzahl der Ressourcen in Ihrem Konto kann das Scannen einige Minuten dauern. Während des Scanvorgangs wird ein Fortschrittsbalken angezeigt.

Unterstützte Ressourcentypen

Der AWS CDK CLI migriert Ressourcen, die vom IaC-Generator unterstützt werden. Eine vollständige Liste finden Sie im AWS CloudFormation Benutzerhandbuch unter [Unterstützung für Ressourcentypen](#).

Löst Eigenschaften auf, die nur über Schreibzugriff verfügen

Einige unterstützte Ressourcen enthalten schreibgeschützte Eigenschaften. In diese Eigenschaften kann geschrieben werden, um die Eigenschaft zu konfigurieren, sie können jedoch nicht vom IaC-Generator gelesen oder AWS CloudFormation der Wert abgerufen werden. Beispielsweise kann eine Eigenschaft, die zur Angabe eines Datenbankkennworts verwendet wird, aus Sicherheitsgründen schreibgeschützt sein.

Beim Scannen von Ressourcen während der Migration erkennt der IaC-Generator Ressourcen, die möglicherweise schreibgeschützte Eigenschaften enthalten, und kategorisiert sie in einen der folgenden Typen:

- **MUTUALLY_EXCLUSIVE_PROPERTIES**— Dabei handelt es sich um schreibgeschützte Eigenschaften für eine bestimmte Ressource, die austauschbar sind und einem ähnlichen Zweck dienen. Eine der sich gegenseitig ausschließenden Eigenschaften ist erforderlich, um Ihre Ressource zu konfigurieren. Bei den `ZipFile` Eigenschaften `S3BucketImageUri`, und einer `AWS::Lambda::Function` Ressource handelt es sich beispielsweise um sich gegenseitig ausschließende, schreibgeschützte Eigenschaften. Jede von ihnen kann verwendet werden, um Ihre Funktionsressourcen zu spezifizieren, aber Sie müssen eines verwenden.
- **MUTUALLY_EXCLUSIVE_TYPES**— Dabei handelt es sich um erforderliche schreibgeschützte Eigenschaften, die mehrere Konfigurationstypen akzeptieren. Beispielsweise akzeptiert die `Body` Eigenschaft einer `AWS::ApiGateway::RestApi` Ressource einen Objekt- oder Zeichenfolgentyp.
- **UNSUPPORTED_PROPERTIES**— Dabei handelt es sich um reine Schreibeigenschaften, die nicht unter die anderen beiden Kategorien fallen. Es handelt sich entweder um optionale Eigenschaften oder um erforderliche Eigenschaften, die eine Reihe von Objekten akzeptieren.

Weitere Informationen zu schreibgeschützten Eigenschaften und dazu, wie IaC Generator sie verwaltet, wenn nach bereitgestellten Ressourcen gesucht und AWS CloudFormation Vorlagen erstellt werden, finden Sie unter [IaC-Generator und schreibgeschützte](#) Eigenschaften im Benutzerhandbuch.AWS CloudFormation

Nach der Migration müssen Sie in der neuen CDK-App Eigenschaftswerte mit Schreibschutz angeben. Daraufhin AWS CDK CLI wird ein Abschnitt mit Warnungen an die ReadMe Datei des CDK-Projekts angehängt, um alle schreibgeschützten Eigenschaften zu dokumentieren, die vom IaC-Generator identifiziert wurden. Im Folgenden wird ein Beispiel gezeigt:

```
# 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
```

- Warnungen sind in Überschriften angeordnet, die die logische ID der Ressource angeben, der sie zugeordnet sind.
- Warnungen werden nach Typ kategorisiert. Diese Typen stammen direkt vom IaC-Generator.

Um Eigenschaften aufzulösen, bei denen nur Schreibzugriff möglich ist

1. Identifizieren Sie im Abschnitt „Warnungen“ der Datei Ihres CDK-Projekts die Eigenschaften, bei denen nur Schreibzugriff besteht, die behoben werden sollen. ReadMe Hier können Sie sich die Ressourcen in Ihrer CDK-App notieren, die schreibgeschützte Eigenschaften enthalten können, und die erkannten schreibgeschützten Eigenschaftstypen identifizieren.
 - a. Legen Sie nämlich fest `MUTUALLY_EXCLUSIVE_PROPERTIES`, welche sich gegenseitig ausschließende Eigenschaft in Ihrer App konfiguriert werden soll. AWS CDK
 - b. Stellen Sie `MUTUALLY_EXCLUSIVE_TYPES` nämlich fest, welchen akzeptierten Typ Sie für die Konfiguration der Eigenschaft verwenden werden.
 - c. Stellen Sie für fest `UNSUPPORTED_PROPERTIES`, ob die Eigenschaft optional oder erforderlich ist. Konfigurieren Sie dann nach Bedarf.
2. Verschaffen Sie sich anhand der Anweisungen des [laC-Generators und der schreibgeschützten Eigenschaften einen Überblick](#) über die Bedeutung der Warnungstypen.
3. In deiner CDK-App werden Eigenschaftswerte, die nur mit Schreibzugriff aufgelöst werden sollen, auch im Abschnitt deiner App angegeben. Props Geben Sie hier die richtigen Werte ein. Objektbeschreibungen und Anleitungen finden Sie in der [AWS CDK API-Referenz](#).

Im Folgenden finden Sie ein Beispiel für den Props Abschnitt innerhalb einer migrierten CDK-App mit zwei Eigenschaften, die nur beim Schreiben verwendet werden müssen, die aufgelöst werden müssen:

```
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;  
}
```

Sobald Sie alle Eigenschaftswerte für schreibgeschützte Eigenschaften aufgelöst haben, können Sie sich auf die Bereitstellung vorbereiten.

Die Datei migrate.json

Die AWS CDK CLI erstellt während der Migration eine `migrate.json` Datei in Ihrem AWS CDK Projekt. Diese Datei enthält Referenzinformationen zu Ihren bereitgestellten Ressourcen. Wenn Sie Ihre CDK-App zum ersten Mal bereitstellen, AWS CDK CLI verwendet sie diese Datei, um auf Ihre bereitgestellten Ressourcen zu verweisen, ordnet Ihre Ressourcen dem neuen AWS CloudFormation Stack zu und löscht die Datei.

Verwalten und implementieren Sie Ihre CDK-App

Bei der Migration zu ist AWS CDK die neue CDK-App möglicherweise nicht sofort einsatzbereit. In diesem Thema werden Maßnahmen beschrieben, die Sie bei der Verwaltung und Bereitstellung Ihrer neuen CDK-App berücksichtigen sollten.

Bereiten Sie sich auf die Bereitstellung vor

Vor der Bereitstellung müssen Sie Ihre CDK-App vorbereiten.

Synthetisieren Sie Ihre App

Verwenden Sie den `cdk synth` Befehl, um den Stack in Ihrer CDK-App zu einer Vorlage zu synthetisieren. [AWS CloudFormation](#)

Wenn Sie von einem bereitgestellten AWS CloudFormation Stack oder einer Vorlage migriert haben, können Sie die synthetisierte Vorlage mit der migrierten Vorlage vergleichen, um Ressourcen- und Eigenschaftswerte zu überprüfen.

Weitere Informationen über `cdk synth` finden Sie unter [Synthetisieren von Stacks](#).

Führen Sie einen Vergleich durch

Wenn Sie von einem bereitgestellten AWS CloudFormation Stack migriert haben, können Sie den Befehl `cdk diff` verwenden, um einen Vergleich mit dem Stack in Ihrer neuen CDK-App durchzuführen.

Weitere Informationen über `cdk diff` finden Sie unter [Stapel vergleichen](#)

Bootstrap für deine Umgebung

Wenn Sie die Bereitstellung zum ersten Mal aus einer AWS Umgebung heraus durchführen, verwenden Sie diese Option, `cdk bootstrap` um Ihre Umgebung vorzubereiten. Weitere Informationen hierzu finden Sie unter [Bootstrapping](#).

Stellen Sie Ihre CDK-App bereit

Wenn Sie eine CDK-App bereitstellen, AWS CDK CLI nutzt diese den AWS CloudFormation Dienst, um Ihre Ressourcen bereitzustellen. Ressourcen werden in der CDK-App in einem einzigen Stapel gebündelt und als einziger Stapel bereitgestellt. AWS CloudFormation

Je nachdem, von wo aus Sie migriert haben, können Sie die Bereitstellung verwenden, um einen neuen AWS CloudFormation Stack zu erstellen oder einen vorhandenen Stack zu aktualisieren. AWS CloudFormation

Bereitstellen, um einen neuen AWS CloudFormation Stack zu erstellen

Wenn Sie von bereitgestellten Ressourcen migriert haben, AWS CDK CLI wird bei der Bereitstellung automatisch ein neuer AWS CloudFormation Stack erstellt. Ihre bereitgestellten Ressourcen werden in den neuen AWS CloudFormation Stack aufgenommen.

Wenn Sie von einer lokalen AWS CloudFormation Vorlage migriert haben, die nie bereitgestellt wurde, AWS CDK CLI wird bei der Bereitstellung automatisch ein neuer AWS CloudFormation Stack erstellt.

Wenn Sie von einem bereitgestellten AWS CloudFormation Stack oder einer lokalen AWS CloudFormation Vorlage migriert haben, die zuvor bereitgestellt wurde, können Sie die Bereitstellung durchführen, um einen neuen AWS CloudFormation Stack zu erstellen. Gehen Sie wie folgt vor, um einen neuen Stack zu erstellen:

- Stellen Sie es in einer neuen AWS Umgebung bereit. Dies besteht aus der Verwendung eines anderen AWS Kontos oder der Bereitstellung in einer anderen AWS-Region.
- Wenn Sie einen neuen Stack in derselben AWS Umgebung wie der migrierte Stack oder die Vorlage bereitstellen möchten, müssen Sie den Stack-Namen in Ihrer CDK-App auf einen neuen Wert ändern. Sie müssen auch alle logischen IDs der Ressourcen in Ihrer CDK-App ändern. Anschließend können Sie die Bereitstellung in derselben Umgebung durchführen, um einen neuen Stack und neue Ressourcen zu erstellen.

Bereitstellen, um einen vorhandenen AWS CloudFormation Stack zu aktualisieren

Wenn Sie von einem bereitgestellten AWS CloudFormation Stack oder einer lokalen AWS CloudFormation Vorlage migriert haben, die zuvor bereitgestellt wurde, können Sie eine Bereitstellung durchführen, um den vorhandenen AWS CloudFormation Stack zu aktualisieren.

Stellen Sie sicher, dass der Stack-Name in Ihrer CDK-App mit dem Stack-Namen des bereitgestellten AWS CloudFormation Stacks übereinstimmt, und stellen Sie die Lösung in derselben AWS Umgebung bereit.

Konfigurieren Sie Sicherheitsanmeldedaten für den AWS CDKCLI

Wenn Sie die verwenden, AWS Cloud Development Kit (AWS CDK) um Anwendungen in Ihrer lokalen Umgebung zu entwickeln, verwenden Sie in erster Linie die AWS CDK Befehlszeilenschnittstelle (AWS CDK CLI) für die Interaktion AWS , Bereitstellung und Verwaltung Ihrer CDK-Stacks. Um das CDK für die Interaktion CLI zu verwenden AWS, müssen Sie Sicherheitsanmeldedaten auf Ihrem lokalen Computer konfigurieren. Dadurch AWS erfahren Sie, wer Sie sind und welche Berechtigungen Sie haben.

Weitere Informationen zu Sicherheitsanmeldedaten finden Sie unter [AWS Sicherheitsanmeldedaten](#) im IAM-Benutzerhandbuch.

Themen

- [Voraussetzungen](#)
- [Wie konfiguriert man Sicherheitsanmeldedaten](#)
- [Zusätzliche Informationen](#)
- [Konfiguration und Verwaltung von CLI CDK-Sicherheitsanmeldedaten für IAM Identity Center-Benutzer](#)

Voraussetzungen

Wenn Sie mit AWS oder dem noch nicht vertraut sind AWS CDK, füllen Sie die Voraussetzungen in diesem Abschnitt aus.

Erstellen Sie einen AWS-Konto Benutzer mit Administratorrechten

Bevor Sie Ihre Methode zur Authentifizierung mit Sicherheitsanmeldedaten festlegen, müssen Sie oder Ihre Organisation über einen AWS-Konto Administratorbenutzer verfügen. Weitere Informationen finden Sie unter [Getting up with IAM](#) im IAM-Benutzerhandbuch.

Sie können IAM mit verschiedenen Methoden verwalten, z. B. über die AWS Konsole AWS Command Line Interface (AWS CLI) oder über Anwendungsschnittstellen (APIs) in den zugehörigen SDKs Wenn Sie IAM mit dem CDK verwendenCLI, verwenden Sie das hauptsächlich AWS CLI zur Konfiguration und Verwaltung von Sicherheitsanmeldeinformationen. Weitere Informationen finden

Sie unter [AWS Command Line Interface \(CLI\) und Software Development Kits \(SDKs\)](#) im IAM-Benutzerhandbuch.

Bestimmen Sie Ihre Methode zur Erstellung und Verwaltung von Benutzern

Sobald Sie Ihren AWS-Konto und Ihren Administratorbenutzer erstellt haben, sollten Sie Ihre Methode zur Benutzerverwaltung festlegen und dabei das Konzept der Berechtigungen mit den geringsten Rechten für Ihre Benutzer einhalten. [Dies ist eine IAM-Best-Practice-Empfehlung.](#) Informationen zu den verschiedenen Methoden zur Benutzerverwaltung finden Sie im IAM-Benutzerhandbuch unter [Überblick über AWS Identitätsmanagement: Benutzer.](#)

Wenn Ihr Unternehmen über eine Methode zur Benutzerverwaltung verfügt, folgen Sie deren Anweisungen. Andernfalls empfehlen wir die Verwendung, AWS IAM Identity Center um Benutzer zu erstellen und zu verwalten. Mit IAM Identity Center können Sie Benutzer und Berechtigungen von einem zentral verwalteten Service aus verwalten AWS-Konten. Sie können Ihren Benutzern auch kurzfristige Anmeldeinformationen für die Authentifizierung zur Verfügung stellen AWS. Eine Einführung finden Sie unter [Was ist IAM Identity Center?](#) im AWS IAM Identity Center Benutzerhandbuch.

Sobald Sie mit der Erstellung von Benutzern beginnen, müssen diese Sicherheitsanmeldedaten auf ihrem lokalen Computer konfigurieren. Benutzer können das verwenden AWS CLI , um dies zu erreichen.

Installieren Sie das AWS CLI

Als Benutzer verwenden Sie die, AWS CLI um Konfigurations- und Anmeldeinformationsdateien auf Ihrem lokalen Computer zu erstellen und zu verwalten. Diese Dateien werden zum Speichern, Verwalten und Generieren von Sicherheitsanmeldeinformationen verwendet, die Sie mit dem CLI CDK verwenden können.

Informationen zur AWS CLI Installation von finden [Sie AWS CLI im AWS Command Line Interface Benutzerhandbuch unter Installation oder Aktualisierung auf die neueste Version von.](#)

Weitere Informationen zu diesen Dateien finden Sie im AWS Command Line Interface Benutzerhandbuch unter [Konfiguration und Einstellungen für Anmeldeinformationsdateien.](#)

Installieren Sie den AWS CDKCLI

Verwenden Sie denNode Package Manager, um das CDK CLI zu installieren. In den meisten Fällen empfehlen wir, die neueste Version global zu installieren, indem Sie Folgendes ausführen:

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

Wie konfiguriert man Sicherheitsanmeldedaten

Wie Sie Sicherheitsanmeldedaten konfigurieren, hängt davon ab, wie Sie oder Ihre Organisation Benutzer verwalten. Wir empfehlen, dass Sie die verwenden, AWS CLI um Sicherheitsanmeldeinformationen für das CDK CLI zu konfigurieren und zu verwalten. Dazu gehört die Verwendung von AWS CLI Befehlen wie `aws configure` der Konfiguration von Sicherheitsanmeldeinformationen auf Ihrem lokalen Computer. Sie können jedoch alternative Methoden verwenden, z. B. Ihre `config credentials` Dateien manuell aktualisieren oder Umgebungsvariablen festlegen.

Anleitungen zur Konfiguration von Sicherheitsanmeldedaten mithilfe von sowie Informationen zur Konfiguration und zur Priorität von Anmeldeinformationen bei Verwendung verschiedener Methoden finden Sie unter [Authentifizierungs- und Zugangsdaten](#) im AWS Command Line Interface Benutzerhandbuch. AWS CLI Das CDK CLI verwendet dieselbe Konfiguration und denselben Vorrang der Anmeldeinformationen. AWS CLI Die `--profile` Befehlszeilenoption hat Vorrang vor Umgebungsvariablen. Wenn Sie sowohl die Umgebungsvariablen als `AWS_PROFILE` auch die `CDK_DEFAULT_PROFILE` Umgebungsvariablen konfiguriert haben, hat die `AWS_PROFILE` Umgebungsvariable Vorrang.

Wenn Sie grundlegende Einstellungen, einschließlich Sicherheitsanmeldedaten, lieber schnell konfigurieren möchten, finden Sie weitere Informationen unter [Einrichten von AWS CLI im AWS Command Line Interface](#) Benutzerhandbuch.

Sobald Sie die Sicherheitsanmeldedaten auf Ihrem lokalen Computer konfiguriert haben, können Sie mit AWS dem CDK CLI interagieren. Weitere Informationen zur Verwendung des CDK CLI mit Ihrer Methode zur Benutzerverwaltung finden Sie im Folgenden:

- [Konfiguration und Verwaltung von CLI CDK-Sicherheitsanmeldedaten für IAM Identity Center-Benutzer.](#)

Zusätzliche Informationen

Informationen zu den verschiedenen Möglichkeiten, mit denen Sie sich je nach Benutzertyp anmelden können, finden Sie unter [Was ist AWS Anmeldung?](#) AWS im AWS Anmelde-Benutzerhandbuch.

Referenzinformationen zur Verwendung AWS SDKs und Tools, einschließlich des AWS CLI, finden Sie im Referenzhandbuch [AWSSDKs und im Tools-Referenzhandbuch](#).

Konfiguration und Verwaltung von CLI CDK-Sicherheitsanmeldedaten für IAM Identity Center-Benutzer

Benutzer, die mit erstellt und verwaltet werden, AWS IAM Identity Center können kurzfristige Anmeldeinformationen für die AWS CDK Befehlszeilenschnittstelle (AWS CDK CLI) generieren.

Eine Einführung in Sicherheitsanmeldedaten mit dem CDKCLI, einschließlich der Voraussetzungen, finden Sie unter [Konfigurieren Sie Sicherheitsanmeldedaten für den AWS CDKCLI](#).

Themen

- [Konfigurieren Sie das AWS CLI für die Verwendung von IAM Identity Center](#)
- [Verwenden Sie die AWS CLI , um kurzfristige Anmeldeinformationen zu erhalten](#)
- [Konfigurieren Sie kurzfristige Anmeldeinformationen manuell](#)
- [Verwenden Sie kurzfristige Anmeldeinformationen mit dem CDK CLI](#)
- [Beispiel: Verwenden Sie die AWS CLI , um kurzfristige Sicherheitsanmeldedaten als IAM Identity Center-Benutzer zu konfigurieren](#)

Konfigurieren Sie das AWS CLI für die Verwendung von IAM Identity Center

Sie können den so konfigurieren, dass er AWS CLI sich mit dem IAM Identity Center authentifiziert. Dies ist der empfohlene Ansatz zur Konfiguration von Sicherheitsanmeldedaten für IAM Identity Center-Benutzer. Die IAM Identity Center-Informationen, die zur Generierung kurzfristiger Anmeldeinformationen benötigt werden, werden in der `config` Datei auf Ihrem lokalen Computer gespeichert. Sie verwenden die AWS CLI `aws configure sso`, um diese Datei zu konfigurieren. Weitere Informationen finden [Sie im AWS Command Line Interface Benutzerhandbuch unter Konfiguration für AWS CLI die Verwendung von AWS IAM Identity Center](#).

Nachdem Sie diese Anweisungen ausgeführt haben, sollte Ihre `config` Datei mindestens ein benanntes Profil und `sso-session` haben. Sie können diese Informationen verwenden, um kurzfristige Anmeldeinformationen für Ihr Profil anzufordern und zu aktualisieren.

Verwenden Sie die AWS CLI , um kurzfristige Anmeldeinformationen zu erhalten

Sie können den AWS CLI `aws sso login` Befehl verwenden, um kurzfristige Anmeldeinformationen für Ihr Profil zu erhalten. Sie können den `aws sso login` Befehl auch verwenden, um zwischen Profilen zu wechseln. Anweisungen zum Abrufen kurzfristiger Anmeldeinformationen und zum Wechseln von Profilen finden Sie im AWS Command Line Interface Benutzerhandbuch unter [Verwenden eines benannten IAM Identity Center-Profiles](#).

Konfigurieren Sie kurzfristige Anmeldeinformationen manuell

Als Alternative zur Verwendung von können Sie Ihre kurzfristigen Anmeldeinformationen vom AWS Zugriffsportaal abrufen `credentials` und Ihre `config` Dateien manuell aktualisieren, um Sicherheitsanmeldeinformationen zu konfigurieren. AWS CLI Anweisungen finden Sie im Benutzerhandbuch unter [Authentifizieren mit kurzfristigen AWS Command Line Interface Anmeldeinformationen](#).

Verwenden Sie kurzfristige Anmeldeinformationen mit dem CDK CLI

Das CDK CLI verwendet die kurzfristigen Anmeldeinformationen, die Sie mit den Dateien generieren AWS CLI oder die Sie manuell in den Dateien `credentials` und `config` konfigurieren. Sobald Sie die kurzfristigen Anmeldeinformationen konfiguriert haben, können Sie mit dem CDK CLI interagieren. AWS

Wenn Sie das CDK CLI für die Interaktion mit Ihren CDK-Stacks verwenden AWS, verwenden Sie die `--profile` Option mit jedem CLI CDK-Befehl, um das benannte Profil anzugeben, das Sie zur Generierung der kurzfristigen Anmeldeinformationen verwendet haben. Wenn Sie das default Profil als Ihr IAM Identity Center-Profil konfiguriert haben, müssen Sie kein Profil angeben.

Beispiel: Verwenden Sie die AWS CLI , um kurzfristige Sicherheitsanmeldedaten als IAM Identity Center-Benutzer zu konfigurieren

In diesem Beispiel konfigurieren wir AWS Command Line Interface (AWS CLI), um unseren Benutzer mit der AWS IAM Identity Center-Token-Provider-Konfiguration zu authentifizieren. Mit der Konfiguration des SSO-Token-Anbieters können aktualisierte Authentifizierungstoken AWS CLI automatisch abgerufen werden, um kurzfristige Anmeldeinformationen zu generieren, die wir mit der AWS Cloud Development Kit (AWS CDK) Befehlszeilenschnittstelle () verwenden können. AWS CDK CLI

Themen

- [Voraussetzungen](#)
- [Schritt 1: Konfigurieren Sie den AWS CLI](#)
- [Schritt 2: Verwenden Sie die AWS CLI , um Sicherheitsanmeldeinformationen zu generieren](#)
- [Schritt 3: Verwenden Sie das CDK CLI](#)

Voraussetzungen

In diesem Beispiel wird davon ausgegangen, dass die folgenden Voraussetzungen erfüllt sind:

- Voraussetzungen, die für die Einrichtung AWS und Installation unserer CLI Starttools erforderlich sind. Weitere Informationen finden Sie unter [Voraussetzungen](#).
- Das IAM Identity Center wurde von unserer Organisation als Methode zur Benutzerverwaltung eingerichtet.
- Im IAM Identity Center wurde mindestens ein Benutzer erstellt.

Schritt 1: Konfigurieren Sie den AWS CLI

Ausführliche Anweisungen [AWS CLI zu diesem Schritt finden Sie im AWS Command Line Interface Benutzerhandbuch unter Konfiguration der Anmeldeinformationen des IAM Identity Center-Tokenanbieters mit automatischer Authentifizierungsaktualisierung](#).

Wir melden uns bei dem von unserer Organisation bereitgestellten AWS Zugriffsportal an, um unsere IAM Identity Center-Informationen zu sammeln. Dazu gehören die SSO-Start-URL und die SSO-Region.

Als Nächstes verwenden wir den AWS CLI `aws configure sso` Befehl, um ein IAM Identity Center-Profil und `sso-session` auf unserem lokalen Computer zu konfigurieren:

```
$ aws configure sso
SSO session name (Recommended): my-sso
SSO start URL [None]: https://my-sso-portal.awsapps.com/start
SSO region [None]: us-east-1
SSO registration scopes [sso:account:access]: <ENTER>
```

Die AWS CLI Versuche, unseren Standardbrowser zu öffnen, um den Anmeldevorgang für unser IAM Identity Center-Konto zu starten. Wenn unser Browser nicht geöffnet werden kann, erhalten Sie

Anweisungen zum manuellen Starten des Anmeldevorgangs. AWS CLI Dieser Vorgang verknüpft die IAM Identity Center-Sitzung mit unserer aktuellen AWS CLI Sitzung.

Nachdem wir unsere Sitzung eingerichtet haben, AWS CLI werden die folgenden für uns AWS-Konten verfügbaren angezeigt:

```
There are 2 AWS accounts available to you.  
> DeveloperAccount, developer-account-admin@example.com (123456789011)  
   ProductionAccount, production-account-admin@example.com (123456789022)
```

Wir verwenden die Pfeiltasten, um unsere auszuwählen DeveloperAccount.

Als Nächstes AWS CLI werden die IAM-Rollen angezeigt, die uns von unserem ausgewählten Konto aus zur Verfügung stehen:

```
Using the account ID 123456789011  
There are 2 roles available to you.  
> ReadOnly  
   FullAccess
```

Wir verwenden die Pfeiltasten zur Auswahl FullAccess.

Als Nächstes werden wir AWS CLI aufgefordert, die Konfiguration abzuschließen, indem wir ein Standardausgabeformat, einen Standard AWS-Region und einen Namen für unser Profil angeben:

```
CLI default client Region [None]: us-west-2 <ENTER>>  
CLI default output format [None]: json <ENTER>  
CLI profile name [123456789011_FullAccess]: my-dev-profile <ENTER>
```

Daraufhin AWS CLI wird eine letzte Meldung angezeigt, die zeigt, wie das benannte Profil verwendet wird AWS CLI mit:

To use this profile, specify the profile name using --profile, as shown:

```
aws s3 ls --profile my-dev-profile
```

Nach Abschluss dieses Schritts sieht unsere config Datei wie folgt aus:

```
[profile my-dev-profile]
```

```
sso_session = my-sso
sso_account_id = 123456789011
sso_role_name = fullAccess
region = us-west-2
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_registration_scopes = sso:account:access
```

Wir können dieses `sso-session` und das benannte Profil jetzt verwenden, um Sicherheitsanmeldeinformationen anzufordern.

Schritt 2: Verwenden Sie die AWS CLI , um Sicherheitsanmeldeinformationen zu generieren

Ausführliche Anweisungen zu diesem Schritt finden Sie im AWS Command Line Interface Benutzerhandbuch unter [Verwenden eines benannten IAM Identity Center-Profiles](#).

Wir verwenden den AWS CLI `aws sso login` Befehl, um Sicherheitsanmeldedaten für unser Profil anzufordern:

```
$ aws sso login --profile my-dev-profile
```

AWS CLI Es versucht, unseren Standardbrowser zu öffnen und überprüft unsere IAM-Anmeldung. Wenn wir derzeit nicht bei IAM Identity Center angemeldet sind, werden wir aufgefordert, den Anmeldevorgang abzuschließen. Wenn unser Browser nicht geöffnet werden kann, erhalten Sie Anweisungen zum manuellen Starten des Autorisierungsvorgangs. AWS CLI

Nach erfolgreicher Anmeldung werden unsere Anmeldeinformationen für die IAM Identity Center-Sitzung zwischengespeichert. Diese Anmeldeinformationen enthalten einen Ablaufzeitstempel. Wenn sie ablaufen, fordern sie AWS CLI uns auf, uns erneut bei IAM Identity Center anzumelden.

Mithilfe gültiger IAM Identity Center-Anmeldeinformationen ruft das auf AWS CLI sichere Weise AWS Anmeldeinformationen für die in unserem Profil angegebene IAM-Rolle ab. Von hier aus können wir das AWS CDK CLI mit unseren Anmeldeinformationen verwenden.

Schritt 3: Verwenden Sie das CDK CLI

Bei jedem CLI CDK-Befehl verwenden wir die `--profile` Option, um das benannte Profil anzugeben, für das wir Anmeldeinformationen generiert haben. Wenn unsere Anmeldeinformationen gültig sind, führt das CDK den CLI Befehl erfolgreich aus. Im Folgenden wird ein Beispiel gezeigt:

```
$ cdk diff --profile my-dev-profile
Stack CdkAppStack
Hold on while we create a read-only change set to get a diff with accurate replacement
  information (use --no-change-set to use a less accurate but faster template-only diff)
Resources
[-] AWS::S3::Bucket myBucket myBucket5AF9C99B destroy

Outputs
[-] Output BucketRegion: {"Value":{"Ref":"AWS::Region"}}

# Number of stacks with differences: 1
```

Wenn unsere Anmeldeinformationen ablaufen, wird eine Fehlermeldung wie die folgende angezeigt:

```
$ cdk diff --profile my-dev-profile
Stack CdkAppStack

Unable to resolve AWS account to use. It must be either configured when you define your
  CDK Stack, or through the environment
```

Um unsere Anmeldeinformationen zu aktualisieren, verwenden wir den AWS CLI `aws sso login` folgenden Befehl:

```
$ aws sso login --profile my-dev-profile
```


Konfigurieren Sie Umgebungen für die Verwendung mit dem AWS CDK

Sie können AWS Umgebungen auf verschiedene Arten konfigurieren, um sie mit dem zu verwenden AWS Cloud Development Kit (AWS CDK). Die beste Methode zur Verwaltung von AWS Umgebungen hängt von Ihren spezifischen Anforderungen ab.

Jeder CDK-Stack in Ihrer Anwendung muss irgendwann einer Umgebung zugeordnet werden, um zu bestimmen, wo der Stack bereitgestellt wird.

Eine Einführung in AWS Umgebungen finden Sie unter [Umgebungen](#).

Themen

- [Wo Sie Umgebungen angeben können](#)
- [Priorität der Umgebung mit dem AWS CDK](#)
- [Wann müssen Umgebungen angegeben werden](#)
- [Wie spezifiziert man Umgebungen mit dem AWS CDK](#)
- [Überlegungen bei der Konfiguration von Umgebungen mit dem AWS CDK](#)
- [Beispiele](#)

Wo Sie Umgebungen angeben können

Sie können Umgebungen in Anmeldeinformationen und Konfigurationsdateien angeben oder indem Sie die [env](#) Eigenschaft des Stack Konstrukts aus der AWS Construct-Bibliothek verwenden.

Anmeldeinformationen und Konfigurationsdateien

Sie können das AWS Command Line Interface (AWS CLI) verwenden, um config Dateien zu erstellen `credentials`, in denen Ihre AWS Umgebungsinformationen gespeichert, organisiert und verwaltet werden. Weitere Informationen zu diesen Dateien finden Sie im AWS Command Line Interface Benutzerhandbuch unter [Konfiguration und Einstellungen für Anmeldeinformationsdateien](#).

Die in diesen Dateien gespeicherten Werte sind nach Profilen organisiert. Wie Sie Ihre Profile und die Schlüssel-Wert-Paare in diesen Dateien benennen, hängt von Ihrer Methode zur Konfiguration des

programmatischen Zugriffs ab. Weitere Informationen zu den verschiedenen Methoden finden Sie unter [Konfigurieren Sie Sicherheitsanmeldedaten für den AWS CDKCLI](#)

Im Allgemeinen AWS CDK löst der AWS-Konto Informationen aus Ihrer `credentials` Datei und AWS-Region Informationen aus Ihrer `config` Datei auf.

Sobald Sie Ihre `credentials` `config` Dateien konfiguriert haben, können Sie die zu verwendende Umgebung mit den AWS CDK CLI und über Umgebungsvariablen angeben.

env-Eigenschaft des Stack-Konstrukts

Sie können die Umgebung für jeden Stapel angeben, indem Sie die `env` Eigenschaft des Stack Konstrukts verwenden. Diese Eigenschaft definiert ein Konto und eine Region, die verwendet werden sollen. Sie können hartcodierte Werte an diese Eigenschaft oder Umgebungsvariablen übergeben, die vom CDK angeboten werden.

Um Umgebungsvariablen zu übergeben, verwenden Sie die Umgebungsvariablen `AWS_DEFAULT_ACCOUNT` und `AWS_DEFAULT_REGION`. Diese Umgebungsvariablen können Werte aus Ihren `config` Dateien `credentials` und übergeben. Sie können auch Logik in Ihrem CDK-Code verwenden, um die Werte dieser Umgebungsvariablen zu bestimmen.

Priorität der Umgebung mit dem AWS CDK

Wenn Sie mehrere Methoden zur Angabe von Umgebungen verwenden, gilt AWS CDK für die folgende Priorität:

1. Hartcodierte Werte, die mit der `env` Eigenschaft des Konstrukts angegeben werden. Stack
2. `AWS_DEFAULT_ACCOUNT` und `AWS_DEFAULT_REGION` Umgebungsvariablen, die mit der `env` Eigenschaft des Stack Konstrukts angegeben sind.
3. Umgebungsinformationen, die mit dem Profil aus Ihren `config` Dateien `credentials` und verknüpft sind und CLI mithilfe der `--profile` Option an das CDK übergeben wurden.
4. Das `default` Profil aus Ihren `config` Dateien `credentials` und.

Wann müssen Umgebungen angegeben werden

Wenn Sie mit dem CDK entwickeln, definieren Sie zunächst CDK-Stacks, die Konstrukte enthalten, die Ressourcen repräsentieren. AWS Als Nächstes synthetisieren Sie jeden CDK-Stapel zu einer Vorlage. AWS CloudFormation Anschließend stellen Sie die CloudFormation Vorlage in Ihrer

Umgebung bereit. Wie Sie Umgebungen angeben, bestimmt, wann Ihre Umgebungsinformationen angewendet werden, und kann sich auf das Verhalten und die Ergebnisse von CDK auswirken.

Geben Sie Umgebungen bei der Vorlagensynthese an

Wenn Sie Umgebungsinformationen mithilfe der `env` Eigenschaft des `Stack` Konstrukts angeben, werden Ihre Umgebungsinformationen bei der Vorlagensynthese verwendet. Wird ausgeführt `cdk synth` oder `cdk deploy` erzeugt eine umgebungsspezifische Vorlage `CloudFormation` .

Wenn Sie innerhalb der `env` Eigenschaft Umgebungsvariablen verwenden, müssen Sie die `--profile` Option mit CLI CDK-Befehlen verwenden, um das Profil mit Ihren Umgebungsinformationen aus Ihren Anmeldeinformationen und Konfigurationsdateien zu übergeben. Diese Informationen werden dann bei der Vorlagensynthese verwendet, um eine umgebungsspezifische Vorlage zu erstellen.

Die in der `CloudFormation` Vorlage enthaltenen Umgebungsinformationen haben Vorrang vor anderen Methoden. Wenn Sie beispielsweise eine andere Umgebung angeben `cdk deploy --profile profile`, wird das Profil ignoriert.

Wenn Sie Umgebungsinformationen auf diese Weise bereitstellen, können Sie in Ihrer CDK-App umgebungsabhängigen Code und Logik verwenden. Das bedeutet auch, dass die synthetisierte Vorlage je nach Maschine, Benutzer oder Sitzung, unter der sie synthetisiert wurde, unterschiedlich sein kann. Dieser Ansatz ist während der Entwicklung oft akzeptabel oder wünschenswert, wird jedoch nicht für den produktiven Einsatz empfohlen.

Geben Sie die Umgebungen bei der Stack-Bereitstellung an

Wenn Sie keine Umgebung anhand der `env` Eigenschaft des `Stack` Konstrukts angeben, erstellt das CDK bei der CLI Synthese eine umgebungsunabhängige Vorlage `CloudFormation` . Anschließend können Sie die Umgebung angeben, in der die Bereitstellung erfolgen soll, indem Sie `cdk deploy --profile profile`

Wenn Sie bei der Bereitstellung einer umgebungsunabhängigen Vorlage kein Profil angeben, versucht das CDKCLI, Umgebungswerte aus Ihrem `default credentials` Profil und den Dateien bei der Bereitstellung zu verwenden. `config`

Wenn Umgebungsinformationen bei der Bereitstellung nicht verfügbar sind, versucht es, AWS `CloudFormation` Umgebungsinformationen bei der Bereitstellung anhand von umgebungsbezogenen Attributen wie, und aufzulösen. `stack.account` `stack.region` `stack.availabilityZones`

Bei umgebungsunabhängigen Stacks können Konstrukte innerhalb des Stacks keine Umgebungsinformationen verwenden, und Sie können keine Logik verwenden, die Umgebungsinformationen benötigt. Sie können beispielsweise keinen Code schreiben `if (stack.region === 'us-east-1')` oder Konstruktmethoden verwenden, für die Umgebungsinformationen erforderlich sind, wie z. [Vpc.fromLookup](#). Um diese Funktionen verwenden zu können, müssen Sie eine Umgebung mit der `env` Eigenschaft angeben.

Bei umgebungsunabhängigen Stacks verfügt jedes Konstrukt, das Availability Zones verwendet, über zwei Availability Zones, sodass der Stack in jeder Region bereitgestellt werden kann.

Wie spezifiziert man Umgebungen mit dem AWS CDK

Spezifizieren Sie hartcodierte Umgebungen für jeden Stack

Verwenden Sie die `env` Eigenschaft des Stack Konstrukts, um AWS Umgebungswerte für Ihren Stack anzugeben. Im Folgenden wird ein Beispiel gezeigt:

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 });
```

Go

```
env_EU := awscdk.Environment{
    Account: jsii.String("8373873873"),
    Region:  jsii.String("eu-west-1"),
}

env_USA := awscdk.Environment{
    Account: jsii.String("2383838383"),
    Region:  jsii.String("us-west-2"),
}

MyFirstStack(app, "first-stack-us", &awscdk.StackProps{
    Env: &env_USA,
})

MyFirstStack(app, "first-stack-eu", &awscdk.StackProps{
    Env: &env_EU,
})
```

Wir empfehlen diesen Ansatz für Produktionsumgebungen. Indem Sie die Umgebung auf diese Weise explizit angeben, können Sie sicherstellen, dass der Stack immer in der spezifischen Umgebung bereitgestellt wird.

Geben Sie Umgebungen mithilfe von Umgebungsvariablen an

Das AWS CDK stellt zwei Umgebungsvariablen bereit, die Sie in Ihrem CDK-Code verwenden können: `CDK_DEFAULT_ACCOUNT` und `CDK_DEFAULT_REGION`. Wenn Sie diese Umgebungsvariablen innerhalb der `env` Eigenschaft Ihrer Stack-Instance verwenden, können Sie mithilfe der CLI `--profile` CDK-Option Umgebungsinformationen aus Ihren Anmeldeinformationen und Konfigurationsdateien übergeben.

Im Folgenden finden Sie ein Beispiel für die Angabe dieser Umgebungsvariablen:

TypeScript

Greifen Sie über das `process` Objekt von Node auf Umgebungsvariablen zu.

Note

Sie benötigen das `DefinitelyTyped` Modul zur Verwendung `process` in TypeScript. `cdk init` installiert dieses Modul für Sie. Sie sollten dieses Modul jedoch manuell installieren, wenn Sie mit einem Projekt arbeiten, das vor dem Hinzufügen erstellt wurde, oder wenn Sie Ihr Projekt nicht mit `cdk init` eingerichtet haben.

```
npm install @types/node
```

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

JavaScript

Greifen Sie über das `process` Objekt von Node auf Umgebungsvariablen zu.

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

Python

Verwenden Sie das `environ` Wörterbuch des `os` Moduls, um auf Umgebungsvariablen zuzugreifen.

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

Java

Wird verwendet `System.getenv()`, um den Wert einer Umgebungsvariablen abzurufen.

```

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#

Wird verwendet `System.Environment.GetEnvironmentVariable()`, um den Wert einer Umgebungsvariablen abzurufen.

```

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() });
```

Go

```
import "os"  
  
MyDevStack(app, "dev", &awscdk.StackProps{  
  Env: &awscdk.Environment{  
    Account: jsii.String(os.Getenv("CDK_DEFAULT_ACCOUNT")),  
    Region:  jsii.String(os.Getenv("CDK_DEFAULT_REGION")),  
  },  
})
```

Indem Sie Umgebungen mithilfe von Umgebungsvariablen angeben, können Sie denselben CDK-Stack zu AWS CloudFormation Vorlagen für verschiedene Umgebungen synthetisieren lassen. Das bedeutet, dass Sie denselben CDK-Stack in verschiedenen AWS Umgebungen bereitstellen können, ohne Ihren CDK-Code ändern zu müssen. Sie müssen nur das Profil angeben, das bei der Ausführung verwendet werden soll. `cdk synth`

Dieser Ansatz eignet sich hervorragend für Entwicklungsumgebungen, wenn derselbe Stack in verschiedenen Umgebungen bereitgestellt wird. Wir empfehlen diesen Ansatz jedoch nicht für Produktionsumgebungen, da derselbe CDK-Code je nach Maschine, Benutzer oder Sitzung, unter der er synthetisiert wird, unterschiedliche Vorlagen synthetisieren kann.

Spezifizieren Sie Umgebungen anhand Ihrer Anmeldeinformationen und Konfigurationsdateien mit dem CDK CLI

Wenn Sie eine umgebungsunabhängige Vorlage bereitstellen, verwenden Sie die `--profile` Option zusammen mit einem beliebigen CLI CDK-Befehl, um das zu verwendende Profil anzugeben. Im Folgenden finden Sie ein Beispiel für die Bereitstellung eines CDK-Stacks, der `myStack` anhand des `prod` Profils benannt wird, das in den Dateien und definiert ist: `credentials config`

```
$ cdk deploy myStack --profile prod
```

Weitere Informationen zu dieser `--profile` Option sowie zu anderen CLI CDK-Befehlen und -Optionen finden Sie unter [AWS CDKCLIBefehlsreferenz](#)

Überlegungen bei der Konfiguration von Umgebungen mit dem AWS CDK

Dienste, die Sie mithilfe von Konstrukten in Ihren Stacks definieren, müssen die Region unterstützen, in der Sie die Bereitstellung durchführen. Eine Liste der unterstützten Dienste AWS-Services pro Region finden Sie unter [AWS Dienste nach Region](#).

Sie benötigen gültige AWS Identity and Access Management (IAM-) Anmeldeinformationen, um Stack-Bereitstellungen mit dem AWS CDK in Ihren angegebenen Umgebungen durchführen zu können.

Beispiele

Synthetisieren Sie eine umgebungsunabhängige Vorlage aus einem CDK-Stack CloudFormation

In diesem Beispiel erstellen wir eine CloudFormation umgebungsunabhängige Vorlage aus unserem CDK-Stack. Wir können diese Vorlage dann in jeder Umgebung bereitstellen.

Im Folgenden finden Sie unser Beispiel für einen CDK-Stack. Dieser Stack definiert einen Amazon S3 S3-Bucket und eine CloudFormation Stack-Ausgabe für die Region des Buckets. Für dieses Beispiel `env` ist nicht definiert:

TypeScript

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

    // Create the S3 bucket
    const bucket = new s3.Bucket(this, 'myBucket', {
      removalPolicy: cdk.RemovalPolicy.DESTROY,
    });

    // Create an output for the bucket's Region
    new cdk.CfnOutput(this, 'BucketRegion', {
      value: bucket.env.region,
    });
  }
}
```

```
}
```

JavaScript

```
class CdkAppStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    // Create the S3 bucket
    const bucket = new s3.Bucket(this, 'myBucket', {
      removalPolicy: cdk.RemovalPolicy.DESTROY,
    });

    // Create an output for the bucket's Region
    new cdk.CfnOutput(this, 'BucketRegion', {
      value: bucket.env.region,
    });
  }
}
```

Python

```
class CdkAppStack(cdk.Stack):

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

        # Create the S3 bucket
        bucket = s3.Bucket(self, 'myBucket',
            removal_policy=cdk.RemovalPolicy.DESTROY
        )

        # Create an output for the bucket's Region
        cdk.CfnOutput(self, 'BucketRegion',
            value=bucket.env.region
        )
```

Java

```
public class CdkAppStack extends Stack {

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

```

    super(scope, id, props);

    // Create the S3 bucket
    Bucket bucket = Bucket.Builder.create(this, "myBucket")
        .removalPolicy(RemovalPolicy.DESTROY)
        .build();

    // Create an output for the bucket's Region
    CfnOutput.Builder.create(this, "BucketRegion")
        .value(this.getRegion())
        .build();
}
}

```

C#

```

namespace MyCdkApp
{
    public class CdkAppStack : Stack
    {
        public CdkAppStack(Construct scope, string id, IStackProps props = null) :
        base(scope, id, props)
        {
            // Create the S3 bucket
            var bucket = new Bucket(this, "myBucket", new BucketProps
            {
                RemovalPolicy = RemovalPolicy.DESTROY
            });

            // Create an output for the bucket's Region
            new CfnOutput(this, "BucketRegion", new CfnOutputProps
            {
                Value = this.Region
            });
        }
    }
}

```

Go

```

func NewCdkAppStack(scope constructs.Construct, id string, props *CdkAppStackProps)
awsdk.Stack {
    stack := awsdk.NewStack(scope, &id, &props.StackProps)
}

```

```
// Create the S3 bucket
bucket := awss3.NewBucket(stack, jsii.String("myBucket"), &awss3.BucketProps{
    RemovalPolicy: awscdk.RemovalPolicy_DESTROY,
})

// Create an output for the bucket's Region
awscdk.NewCfnOutput(stack, jsii.String("BucketRegion"), &awscdk.CfnOutputProps{
    Value: stack.Region(),
})

return stack
}
```

Wenn wir ausführen `cdk synth`, CLI erzeugt das CDK eine CloudFormation Vorlage mit dem Pseudo-Parameter `AWS::Region` als Ausgabewert für die Region des Buckets. Dieser Parameter wird bei der Bereitstellung behoben:

```
Outputs:
  BucketRegion:
    Value:
      Ref: AWS::Region
```

Um diesen Stack in einer Umgebung bereitzustellen, die im dev Profil unserer Anmeldeinformationen und Konfigurationsdateien angegeben ist, führen wir Folgendes aus:

```
$ cdk deploy CdkAppStack --profile dev
```

Wenn wir kein Profil angeben, versucht das CDK, CLI die Umgebungsinformationen aus dem default Profil in unseren Anmeldeinformationen und Konfigurationsdateien zu verwenden.

Verwenden Sie Logik, um Umgebungsinformationen bei der Vorlagensynthese zu ermitteln

In diesem Beispiel konfigurieren wir die `env` Eigenschaft unserer `stack` Instanz so, dass sie einen gültigen Ausdruck verwendet. Wir geben zwei zusätzliche Umgebungsvariablen an, `CDK_DEPLOY_ACCOUNT` und `CDK_DEPLOY_REGION`. Diese Umgebungsvariablen können bei der Synthese Standardwerte außer Kraft setzen, sofern sie existieren:

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() });

```

Go

```

var account, region string
var b bool

if account, b = os.LookupEnv("CDK_DEPLOY_ACCOUNT"); !b || len(account) == 0 {
    account = os.Getenv("CDK_DEFAULT_ACCOUNT")
}
if region, b = os.LookupEnv("CDK_DEPLOY_REGION"); !b || len(region) == 0 {
    region = os.Getenv("CDK_DEFAULT_REGION")
}

MyDevStack(app, "dev", &awscdk.StackProps{

```

```
Env: &awsdk.Environment{
  Account: &account,
  Region: &region,
},
})
```

Wenn die Umgebung unseres Stacks auf diese Weise deklariert ist, können wir dann ein kurzes Skript oder eine Batch-Datei schreiben, Variablen über Befehlszeilenargumente setzen und dann `aufrufencdk deploy`. Im Folgenden wird ein Beispiel gezeigt. Alle Argumente, die über die ersten beiden hinausgehen, werden übergeben `cdk deploy`, um Befehlszeilenoptionen oder Argumente anzugeben:

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
```

Speichern Sie das Skript unter `auscdk-deploy-to.sh`, `chmod +x cdk-deploy-to.sh` um es ausführbar zu machen.

Windows

```
@findstr /B /V @ %~dpx0 > %~dpx0.ps1 && powershell -ExecutionPolicy Bypass
%~dpx0.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
}
```

Die Windows-Version des Skripts bietet PowerShell dieselbe Funktionalität wie die MacOS/Linux-Version. Es enthält auch Anweisungen, mit denen es als Batch-Datei ausgeführt werden kann, sodass es einfach von einer Befehlszeile aus aufgerufen werden kann. Sie sollte gespeichert werden als `cdk-deploy-to.bat`. Die Datei `cdk-deploy-to.ps1` wird erstellt, wenn die Batch-Datei aufgerufen wird.

Wir können dann zusätzliche Skripts schreiben, die das `cdk-deploy-to` Skript für die Bereitstellung in bestimmten Umgebungen verwenden. Im Folgenden wird ein Beispiel gezeigt:

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 %*
```

Im Folgenden finden Sie ein Beispiel, in dem das `cdk-deploy-to` Skript für die Bereitstellung in mehreren Umgebungen verwendet wird. Wenn die erste Bereitstellung fehlschlägt, wird der Prozess gestoppt:

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 %*
```

Bootstrap Ihre Umgebung für die Verwendung mit dem AWS CDK

Führen Sie ein Bootstrapping für Ihre AWS Umgebung durch, um sie für AWS Cloud Development Kit (AWS CDK) Stack-Bereitstellungen vorzubereiten.

- Eine Einführung in Umgebungen finden Sie unter. [Umgebungen](#)
- Eine Einführung in Bootstrapping finden Sie unter. [Bootstrapping](#)

Themen

- [Wie booten Sie Ihre Umgebung](#)
- [Wann sollten Sie Ihre Umgebung booten](#)
- [Passen Sie das Bootstrapping an](#)
- [Bootstrapping mit CDK Pipelines](#)
- [Versionsverlauf der Bootstrap-Vorlage](#)
- [Führen Sie ein Upgrade von einer älteren auf eine moderne Bootstrap-Vorlage durch](#)
- [Gehen Sie auf die Ergebnisse von Security Hub ein](#)
- [Überlegungen](#)
- [AWS CDK Bootstrapping anpassen](#)

Wie booten Sie Ihre Umgebung

Sie können die AWS CDK Befehlszeilenschnittstelle (AWS CDK CLI) oder Ihr bevorzugtes AWS CloudFormation Bereitstellungstool verwenden, um Ihre Umgebung zu booten.

Verwenden Sie das CDK CLI

Sie können den CLI `cdk bootstrap` CDK-Befehl verwenden, um Ihre Umgebung zu booten. Diese Methode empfehlen wir, wenn Sie keine wesentlichen Änderungen am Bootstrapping benötigen.

Bootstrap aus einem beliebigen Arbeitsverzeichnis

Um von einem beliebigen Arbeitsverzeichnis aus zu booten, geben Sie die Umgebung für das Bootstrapping als Befehlszeilenargument an. Im Folgenden wird ein Beispiel gezeigt:

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

Bei der Angabe eines Arguments ist das `aws://` Präfix optional. Folgendes ist gültig:

```
$ cdk bootstrap 123456789012/us-east-1
```

Um mehrere Umgebungen gleichzeitig zu booten, geben Sie mehrere Argumente an:

```
$ cdk bootstrap aws://123456789012/us-east-1 aws://123456789012/us-east-2
```

Bootstrap aus dem übergeordneten Verzeichnis eines CDK-Projekts

Sie können `cdk bootstrap` vom übergeordneten Verzeichnis eines CDK-Projekts aus starten, das eine Datei enthält. `cdk.json` Wenn Sie keine Umgebung als Argument angeben, bezieht das CDK CLI Umgebungsinformationen aus Standardquellen, wie z. B. Ihren `credentials` AND-Dateien oder anderen Umgebungsinformationen, die für Ihren `config` CDK-Stack angegeben sind.

Wenn Sie vom übergeordneten Verzeichnis eines CDK-Projekts aus booten, haben Umgebungen, die über Befehlszeilenargumente bereitgestellt werden, Vorrang vor anderen Quellen.

Um eine Umgebung zu booten, die in Ihren `config` und `credentials` -Dateien angegeben ist, verwenden Sie die Option: `--profile`

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

Weitere Informationen zum `cdk bootstrap` Befehl und zu den unterstützten Optionen finden Sie unter [cdk bootstrap](#).

Verwenden Sie ein beliebiges AWS CloudFormation Tool

Sie können die [Bootstrap-Vorlage](#) aus dem GitHub `aws-cdk`-Repository kopieren oder die Vorlage mit dem Befehl abrufen. `cdk bootstrap --show-template` Verwenden Sie dann ein beliebiges AWS CloudFormation Tool, um die Vorlage in Ihrer Umgebung bereitzustellen.

Mit dieser Methode können Sie AWS CloudFormation StackSets oder verwenden AWS Control Tower. Sie können auch die AWS CloudFormation Konsole oder die AWS Command Line

Interface (AWS CLI) verwenden. Sie können Änderungen an Ihrer Vorlage vornehmen, bevor Sie sie bereitstellen. Diese Methode ist möglicherweise flexibler und eignet sich für umfangreiche Bereitstellungen.

Im Folgenden finden Sie ein Beispiel für die Verwendung der `--show-template` Option zum Abrufen und Speichern der Bootstrap-Vorlage auf Ihrem lokalen Computer:

macOS/Linux

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

Windows

PowerShell Muss unter Windows verwendet werden, um die Kodierung der Vorlage beizubehalten.

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

Um diese Vorlage mithilfe des CDK bereitzustellenCLI, können Sie Folgendes ausführen:

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

Im Folgenden finden Sie ein Beispiel für die Verwendung von AWS CLI zur Bereitstellung der Vorlage:

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
```

Informationen zum Bootstrapping mehrerer Umgebungen finden Sie unter [Bootstrapping multiple AWS-Konten for use CloudFormation StackSets im AWS CDK AWS Cloud Operations & Migrations-Blog](#). CloudFormation StackSets

Wann sollten Sie Ihre Umgebung booten

Sie müssen jede Umgebung bootstrappen, bevor Sie sie in der Umgebung bereitstellen. Wenn Sie versuchen, einen CDK-Stack in einer Umgebung bereitzustellen, für die kein Bootstrapping durchgeführt wurde, wird ein Fehler wie der folgende angezeigt:

```
$ cdk deploy  
  
# Synthesis time: 2.02s  
  
# Deployment failed: Error: BootstrapExampleStack: 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)
```

Es ist in Ordnung, eine Umgebung mehr als einmal zu booten. Wenn eine Umgebung bereits gebootet wurde, wird der Bootstrap-Stack bei Bedarf aktualisiert. Andernfalls wird nichts passieren.

Aktualisiere deinen Bootstrap-Stack

In regelmäßigen Abständen aktualisiert das CDK-Team die Bootstrap-Vorlage auf eine neue Version. In diesem Fall empfehlen wir Ihnen, Ihren Bootstrap-Stack zu aktualisieren. Wenn Sie den Bootstrapping-Prozess nicht angepasst haben, können Sie Ihren Bootstrap-Stack aktualisieren, indem Sie dieselben Schritte ausführen, die Sie beim ursprünglichen Bootstrapping Ihrer Umgebung unternommen haben. Weitere Informationen finden Sie unter [Versionsverlauf der Bootstrap-Vorlage](#).

Passen Sie das Bootstrapping an

Wenn die Standard-Bootstrap-Vorlage nicht Ihren Anforderungen entspricht, können Sie das Bootstrapping von Ressourcen auf folgende Weise an Ihre Umgebung anpassen:

- Verwenden Sie Befehlszeilenoptionen zusammen mit dem `cdk bootstrap` Befehl — Diese Methode eignet sich am besten für kleine, spezifische Änderungen, die über Befehlszeilenoptionen unterstützt werden.
- Ändern Sie die Standard-Bootstrap-Vorlage und stellen Sie sie bereit — Diese Methode eignet sich am besten für komplexe Änderungen oder wenn Sie die vollständige Kontrolle über die Konfiguration der beim Bootstrapping bereitgestellten Ressourcen haben möchten.

Weitere Informationen zum Anpassen von Bootstrapping finden Sie unter [AWS CDK Bootstrapping anpassen](#)

Bootstrapping mit CDK Pipelines

Wenn Sie CDK Pipelines für die Bereitstellung in der Umgebung eines anderen Kontos verwenden und eine Meldung wie die folgende erhalten:

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

Diese Fehlermeldung bedeutet, dass die entsprechenden IAM-Rollen in der anderen Umgebung nicht vorhanden sind. Die wahrscheinlichste Ursache ist, dass für die Umgebung kein Bootstrapping durchgeführt wurde. Starten Sie die Umgebung und versuchen Sie es erneut.

Schützen Sie Ihren Bootstrap-Stack vor dem Löschen

Wenn ein Bootstrap-Stack gelöscht wird, werden auch die AWS Ressourcen gelöscht, die ursprünglich in der Umgebung zur Unterstützung von CDK-Bereitstellungen bereitgestellt wurden. Dies führt dazu, dass die Pipeline nicht mehr funktioniert. In diesem Fall gibt es keine allgemeine Lösung für die Wiederherstellung.

Nachdem Ihre Umgebung gebootet wurde, sollten Sie den Bootstrap-Stack der Umgebung nicht löschen und neu erstellen. Versuchen Sie stattdessen, den Bootstrap-Stack auf eine neue Version zu aktualisieren, indem Sie den Befehl erneut ausführen. `cdk bootstrap`

Zum Schutz vor versehentlichem Löschen Ihres Bootstrap-Stacks empfehlen wir Ihnen, die `--termination-protection` Option mit dem `cdk bootstrap` Befehl zum Aktivieren des Kündigungsschutzes bereitzustellen. Sie können den Kündigungsschutz für neue oder bestehende Bootstrap-Stacks aktivieren. Anweisungen zum Aktivieren des Kündigungsschutzes finden Sie unter [Kündigungsschutz für den Bootstrap-Stack aktivieren](#).

Versionsverlauf der Bootstrap-Vorlage

Die Bootstrap-Vorlage ist versioniert und entwickelt sich im Laufe der Zeit mit der selbst. AWS CDK Wenn Sie Ihre eigene Bootstrap-Vorlage bereitstellen, halten Sie sie mit der kanonischen Standardvorlage auf dem neuesten Stand. Sie möchten sicherstellen, dass Ihre Vorlage weiterhin mit allen CDK-Funktionen funktioniert.

Note

Frühere Versionen der Bootstrap-Vorlage erstellten standardmäßig AWS KMS key in jeder Bootstrap-Umgebung eine. Um Gebühren für den KMS-Schlüssel zu vermeiden, starten Sie diese Umgebungen neu mit. `--no-bootstrap-customer-key` Die aktuelle Standardeinstellung ist kein KMS-Schlüssel, wodurch diese Gebühren vermieden werden können.

Dieser Abschnitt enthält eine Liste der Änderungen, die in den einzelnen Versionen vorgenommen wurden.

Vorlagenversion	AWS CDK Version	Änderungen
1	1.40.0	Erste Version der Vorlage mit Bucket, Schlüssel, Repository und Rollen.
2	1.45.0	Teilen Sie die Rolle zur Veröffentlichung von Inhalten in separate Rollen für die Veröffentlichung von Dateien und Bildern auf.
3	1.46.0	Fügen Sie <code>FileAsset KeyArn</code> Export hinzu, um Benutzern von Inhalten Entschlüsselungsberechtigungen hinzufügen zu können.

Vorlagenversion	AWS CDK Version	Änderungen
4	1.61.0	AWS KMS Berechtigungen sind jetzt über Amazon S3 implizit und nicht mehr erforderlich. <code>FileAssetKeyArn</code> . Fügen Sie den <code>CdkBootstrapVersion</code> SSM-Parameter hinzu, damit die Bootstrap-Stack-Version verifiziert werden kann, ohne den Stacknamen zu kennen.
5	1.87.0	Die Bereitstellungsrolle kann den SSM-Parameter lesen.
6	1.108.0	Fügen Sie die Suchrolle getrennt von der Bereitstellungsrolle hinzu.
6	1.109.0	Fügen Sie den Rollen „Bereitstellung“, „Dateiveröffentlichung“ und „Image-Publishing“ ein <code>aws-cdk:bootstrap-role</code> Tag hinzu.
7	1.110.0	Die Bereitstellungsrolle kann Buckets im Zielkonto nicht mehr direkt lesen. (Bei dieser Rolle handelt es sich jedoch praktisch um eine Administratorrolle und sie könnte ihre AWS CloudFormation Berechtigungen ohnehin nutzen, um den Bucket lesbar zu machen).

Vorlagenversion	AWS CDK Version	Änderungen
8	1.114.0	Die Suchrolle hat volle Leseberechtigungen für die Zielumgebung und verfügt auch über ein Tag. <code>aws-cdk:bootstrap-role</code>
9	2.1.0	Behebt, dass Amazon S3 S3-Asset-Uploads nicht durch häufig referenzierte Verschlüsselungs-SCP abgelehnt werden.
10	2.4.0	Amazon ECR ScanOnPush ist jetzt standardmäßig aktiviert.
11	2.18.0	Fügt eine Richtlinie hinzu, die es Lambda ermöglicht, Daten aus Amazon ECR-Repos abzurufen, sodass es einen Neustart übersteht.
12	2.20.0	Fügt Unterstützung für experimentelle Anwendungen hinzu <code>cdk import</code> .
13	2.25.0	Macht Container-Images in von Bootstrap erstellten Amazon ECR-Repositorys unveränderlich.

Vorlagenversion	AWS CDK Version	Änderungen
14	2.34.0	Deaktiviert standardmäßig das Amazon ECR-Bildscannen auf Repository-Ebene, um das Bootstrapping von Regionen zu ermöglichen, die das Scannen von Bildern nicht unterstützen.
15	2,60,0	KMS-Schlüssel können nicht markiert werden.
16	2,69,0	Behebt die Suche nach KMS.2 durch Security Hub.
17	2,72,0	Behebt den Security Hub Hub-Befund ECR.3 .
18	2,80,0	Die für Version 16 vorgenommenen Änderungen wurden rückgängig gemacht, da sie nicht in allen Partitionen funktionieren und daher nicht empfohlen werden.
19	2,106,1	Die an Version 18 vorgenommenen Änderungen, bei denen die AccessControl Eigenschaft aus der Vorlage entfernt wurde, wurden rückgängig gemacht. (#27964)

Vorlagenversion	AWS CDK Version	Änderungen
20	2,119,0	Fügen Sie der Rolle „AWS CloudFormation IAM-Bereitstellung“ eine <code>ssm:GetParameters</code> Aktion hinzu. Weitere Informationen finden Sie unter #28336 .

Führen Sie ein Upgrade von einer älteren auf eine moderne Bootstrap-Vorlage durch

Die AWS CDK Version 1 unterstützte zwei Bootstrapping-Vorlagen, Legacy und Modern. CDK v2 unterstützt nur das moderne Template. Als Referenz finden Sie hier die wichtigsten Unterschiede zwischen diesen beiden Vorlagen.

Funktion	Legacy (nur Version 1)	Modern (v1 und v2)
Kontoübergreifende Berechtigungen	Nicht zulässig	Zulässig
AWS CloudFormation Berechtigungen	Wird unter Verwendung der Berechtigungen des aktuellen Benutzers (bestimmt durch AWS Profil, Umgebungsvariablen usw.) bereitgestellt	Wird mit den Berechtigungen bereitgestellt, die bei der Bereitstellung des Bootstrap-Stacks angegeben wurden (z. B. mithilfe von <code>--trust</code>)
Versioning	Es ist nur eine Version des Bootstrap-Stacks verfügbar	Der Bootstrap-Stack ist versioniert; neue Ressourcen können in future Versionen hinzugefügt werden, und AWS CDK Apps können eine Mindestversion erfordern
Ressourcen *	Amazon-S3-Bucket	Amazon-S3-Bucket

Funktion	Legacy (nur Version 1)	Modern (v1 und v2)
		AWS KMS key
		IAM-Rollen
		Amazon ECR-Repository
		SSM-Parameter für die Versionierung
Benennung von Ressourcen	Automatisch generiert	Deterministisch
Bucket-Verschlüsselung	Standardschlüssel	AWS standardmäßig verwaltet er Schlüssel. Sie können die Verwendung eines vom Kunden verwalteten Schlüssel s anpassen.

* Wir werden der Bootstrap-Vorlage nach Bedarf zusätzliche Ressourcen hinzufügen.

Eine Umgebung, die mithilfe des Legacy-Templates gebootet wurde, muss durch Re-Bootstrapping aktualisiert werden, sodass sie das moderne Template für CDK v2 verwenden kann. Stellen Sie alle AWS CDK Anwendungen in der Umgebung mindestens einmal erneut bereit, bevor Sie den Legacy-Bucket löschen.

Gehen Sie auf die Ergebnisse von Security Hub ein

Wenn Sie verwenden AWS Security Hub, werden Ihnen möglicherweise Ergebnisse zu einigen Ressourcen gemeldet, die durch den AWS CDK Bootstrapping-Prozess erstellt wurden. Die Ergebnisse von Security Hub helfen Ihnen dabei, Ressourcenkonfigurationen zu finden, die Sie auf Richtigkeit und Sicherheit überprüfen sollten. Wir haben diese spezifischen Ressourcenkonfigurationen mit AWS Security überprüft und sind überzeugt, dass sie kein Sicherheitsproblem darstellen.

[KMS.2] IAM-Prinzipale sollten keine IAM-Inline-Richtlinien haben, die Entschlüsselungsaktionen für alle KMS-Schlüssel zulassen

Die Bereitstellungsrolle (`DeploymentActionRole`) gewährt die Berechtigung zum Lesen verschlüsselter Daten, was für kontoübergreifende Bereitstellungen mit CDK Pipelines erforderlich ist. Richtlinien in dieser Rolle gewähren nicht Zugriff auf alle Daten. Es gewährt nur die Erlaubnis, verschlüsselte Daten aus Amazon S3 zu lesen AWS KMS, und nur, wenn diese Ressourcen dies aufgrund ihrer Bucket- oder Schlüsselrichtlinie zulassen.

Im Folgenden finden Sie einen Auszug dieser beiden Anweisungen in der Deploy-Rolle aus der Bootstrap-Vorlage:

```
DeploymentActionRole:
  Type: AWS::IAM::Role
  Properties:
    ...
  Policies:
    - PolicyDocument:
      Statement:
        ...
        - Sid: PipelineCrossAccountArtifactsBucket
          Effect: Allow
          Action:
            - s3:GetObject*
            - s3:GetBucket*
            - s3:List*
            - s3:Abort*
            - s3>DeleteObject*
            - s3:PutObject*
          Resource: "*"
          Condition:
            StringNotEquals:
              s3:ResourceAccount:
                Ref: AWS::AccountId
    - Sid: PipelineCrossAccountArtifactsKey
      Effect: Allow
      Action:
        - kms:Decrypt
        - kms:DescribeKey
        - kms:Encrypt
        - kms:ReEncrypt*
        - kms:GenerateDataKey*
```

```
Resource: "*"
Condition:
  StringEquals:
    kms:ViaService:
      Fn::Sub: s3.${AWS::Region}.amazonaws.com
  ...
```

Warum meldet Security Hub das?

Die Richtlinien enthalten eine Klausel `Resource: *` in Kombination mit einer `Condition` Klausel. Security Hub kennzeichnet den `*` Platzhalter. Dieser Platzhalter wird verwendet, weil zum Zeitpunkt des Bootstrappings des Kontos der von CDK Pipelines für den CodePipeline Artefakt-Bucket erstellte AWS KMS Schlüssel noch nicht existiert und daher von ARN nicht in der Bootstrap-Vorlage referenziert werden kann. Darüber hinaus berücksichtigt Security Hub die `Condition` Klausel nicht, wenn diese Markierung aktiviert wird. Dies `Condition` beschränkt `Resource: *` sich auf Anfragen, die mit demselben AWS-Konto AWS KMS Schlüssel gestellt werden. Diese Anfragen müssen genauso AWS-Region wie der AWS KMS Schlüssel von Amazon S3 stammen.

Muss ich diesen Befund korrigieren?

Solange Sie den AWS KMS Schlüssel in Ihrer Bootstrap-Vorlage nicht so geändert haben, dass er zu freizügig ist, gewährt die Deploy-Rolle nicht mehr Zugriff als erforderlich. Daher ist es nicht erforderlich, dieses Ergebnis zu korrigieren.

Was ist, wenn ich diesen Befund korrigieren möchte?

Wie Sie dieses Problem beheben, hängt davon ab, ob Sie CDK Pipelines für kontoübergreifende Bereitstellungen verwenden werden oder nicht.

Um das Problem mit dem Security Hub zu beheben und CDK Pipelines für kontoübergreifende Bereitstellungen zu verwenden

1. Falls Sie dies noch nicht getan haben, stellen Sie den CDK-Bootstrap-Stack mit dem folgenden Befehl bereit. `cdk bootstrap`
2. Falls Sie dies nicht getan haben, erstellen und implementieren Sie Ihr CDK. Pipeline Anweisungen finden Sie unter [Kontinuierliche Integration und Bereitstellung \(CI/CD\) mithilfe von CDK Pipelines](#).
3. Besorgen Sie sich den AWS KMS Schlüssel-ARN des CodePipeline Artefakt-Buckets. Diese Ressource wird während der Pipelineerstellung erstellt.

4. Besorgen Sie sich eine Kopie der CDK-Bootstrap-Vorlage, um sie zu ändern. Im Folgenden finden Sie ein Beispiel unter Verwendung von: AWS CDK CLI

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

5. Ändern Sie die Vorlage, indem Sie Resource: * die PipelineCrossAccountArtifactsKey Anweisung durch Ihren ARN-Wert ersetzen.
6. Stellen Sie die Vorlage bereit, um Ihren Bootstrap-Stack zu aktualisieren. Das Folgende ist ein Beispiel für die Verwendung des CLI CDK:

```
$ cdk bootstrap aws://account-id/region --template bootstrap-template.yaml
```

Um das Security Hub Hub-Problem zu beheben, wenn Sie keine CDK Pipelines für kontoübergreifende Bereitstellungen verwenden

1. Besorgen Sie sich eine Kopie der CDK-Bootstrap-Vorlage, um sie zu ändern. Im Folgenden finden Sie ein Beispiel für die Verwendung des CDK: CLI

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

2. Löschen Sie die PipelineCrossAccountArtifactsKey Anweisungen PipelineCrossAccountArtifactsBucket und aus der Vorlage.
3. Stellen Sie die Vorlage bereit, um Ihren Bootstrap-Stack zu aktualisieren. Das Folgende ist ein Beispiel für die Verwendung des CLI CDK:

```
$ cdk bootstrap aws://account-id/region --template bootstrap-template.yaml
```

Überlegungen

Da beim Bootstrapping Ressourcen in Ihrer Umgebung bereitgestellt werden, können AWS Gebühren anfallen, wenn Sie diese Ressourcen zusammen mit dem verwenden. AWS CDK

AWS CDK Bootstrapping anpassen

Sie können das AWS Cloud Development Kit (AWS CDK) Bootstrapping mithilfe der AWS CDK Befehlszeilenschnittstelle (AWS CDK CLI) oder durch Ändern und Bereitstellen der AWS CloudFormation Bootstrap-Vorlage anpassen.

Eine Einführung in Bootstrapping finden Sie unter. [Bootstrapping](#)

Themen

- [Verwenden Sie das CDKCLI, um Bootstrapping anzupassen](#)
- [Ändern Sie die Standard-Bootstrap-Vorlage](#)
- [Folgen Sie dem Bootstrap-Vorlagenvertrag](#)

Verwenden Sie das CDKCLI, um Bootstrapping anzupassen

Im Folgenden finden Sie einige Beispiele dafür, wie Sie das Bootstrapping mithilfe des CDK anpassen können. CLI Eine Liste aller `cdk bootstrap` Optionen finden Sie unter. [cdk bootstrap](#)

Den Namen des Amazon S3 S3-Buckets überschreiben

Verwenden Sie die `--bootstrap-bucket-name` Option, um den standardmäßigen Amazon S3 S3-Bucket-Namen zu überschreiben. Dies kann erfordern, dass Sie die Vorlagensynthese ändern. Weitere Informationen finden Sie unter [Passen Sie die CDK-Stack-Synthese an](#).

Ändern Sie serverseitige Verschlüsselungsschlüssel für den Amazon S3 S3-Bucket

Standardmäßig ist der Amazon S3 S3-Bucket im Bootstrap-Stack so konfiguriert, dass er AWS verwaltete Schlüssel für die serverseitige Verschlüsselung verwendet. Um einen vorhandenen, vom Kunden verwalteten Schlüssel zu verwenden, verwenden Sie die `--bootstrap-kms-key-id` Option und geben Sie einen Wert für den zu verwendenden Schlüssel AWS Key Management Service (AWS KMS) an. Wenn Sie mehr Kontrolle über den Verschlüsselungsschlüssel haben möchten, stellen Sie sicher, `--bootstrap-customer-key` dass Sie einen vom Kunden verwalteten Schlüssel verwenden.

Ordnen Sie verwaltete Richtlinien der Bereitstellungsrolle zu, die übernommen wurde von AWS CloudFormation

Standardmäßig werden Stacks mithilfe der `AdministratorAccess` Richtlinie mit vollen Administratorrechten bereitgestellt. Wenn Sie Ihre eigenen verwalteten Richtlinien verwenden möchten, verwenden Sie die `--cloudformation-execution-policies` Option und geben

Sie die ARNs der verwalteten Richtlinien an, die der Bereitstellungsrolle zugewiesen werden sollen.

Um mehrere Richtlinien bereitzustellen, übergeben Sie ihnen eine einzelne, durch Kommas getrennte Zeichenfolge:

```
$ cdk bootstrap --cloudformation-execution-policies "arn:aws:iam::aws:policy/  
AWSLambda_FullAccess,arn:aws:iam::aws:policy/AWSCodeDeployFullAccess"
```

Um Bereitstellungsfehler zu vermeiden, stellen Sie sicher, dass die von Ihnen angegebenen Richtlinien für alle Bereitstellungen ausreichen, die Sie in der Umgebung durchführen, für die das Bootstrapping ausgeführt wird.

Ändern Sie den Qualifier, der den Namen der Ressourcen in Ihrem Bootstrap-Stack hinzugefügt wird

Standardmäßig wird der `hnb659fds` Qualifier zur physischen ID der Ressourcen in Ihrem Bootstrap-Stack hinzugefügt. Um diesen Wert zu ändern, verwenden Sie die `--qualifier` Option.

Diese Änderung ist nützlich, wenn mehrere Bootstrap-Stacks in derselben Umgebung bereitgestellt werden, um Namenskonflikte zu vermeiden.

Die Änderung des Qualifizierers dient der Namensisolierung zwischen automatisierten Tests des CDK selbst. Wenn Sie die der CloudFormation Ausführungsrolle erteilten IAM-Berechtigungen nicht sehr genau eingrenzen können, bietet die Verwendung von zwei verschiedenen Bootstrap-Stacks in einem einzigen Konto keine Vorteile für die Isolierung von Rechten. Daher ist es normalerweise nicht erforderlich, diesen Wert zu ändern.

Wenn Sie den Qualifier ändern, muss Ihre CDK-App den geänderten Wert an den Stack-Synthesizer übergeben. Weitere Informationen finden Sie unter [Passen Sie die CDK-Stack-Synthese an](#).

Fügen Sie dem Bootstrap-Stack Tags hinzu

Verwenden Sie die `--tags` Option im Format von `KEY=VALUE`, um Ihrem Bootstrap-Stack CloudFormation Tags hinzuzufügen.

Geben Sie weitere Optionen an AWS-Konten, die in der Umgebung bereitgestellt werden können, für die das Bootstrapping ausgeführt wird

Verwenden Sie die `--trust` Option, um weitere bereitzustellen AWS-Konten, die in der Umgebung bereitgestellt werden dürfen, für die das Bootstrapping ausgeführt wird. Standardmäßig ist das Konto, das das Bootstrapping durchführt, immer vertrauenswürdig.

Diese Option ist nützlich, wenn Sie eine Umgebung bootstrappen, in der ein CDK Pipeline aus einer anderen Umgebung bereitgestellt werden soll.

Wenn Sie diese Option verwenden, müssen Sie auch Folgendes angeben. `--cloudformation-execution-policies`

Um vertrauenswürdige Konten zu einem vorhandenen Bootstrap-Stack hinzuzufügen, müssen Sie alle Konten angeben, denen Sie vertrauen möchten, einschließlich der Konten, die Sie möglicherweise zuvor angegeben haben. Wenn Sie nur neue Konten angeben, denen Sie vertrauen können, werden die zuvor vertrauenswürdigen Konten entfernt.

Im Folgenden finden Sie ein Beispiel, bei dem zwei Konten als vertrauenswürdig eingestuft werden:

```
$ cdk bootstrap aws://123456789012/us-west-2 --trust 234567890123 --
trust 987654321098 --cloudformation-execution-policies arn:aws:iam::aws:policy/
AdministratorAccess
# Bootstrapping environment aws://123456789012/us-west-2...
Trusted accounts for deployment: 234567890123, 987654321098
Trusted accounts for lookup: (none)
Execution policies: arn:aws:iam::aws:policy/AdministratorAccess
CDKToolkit: creating CloudFormation changeset...
# Environment aws://123456789012/us-west-2 bootstrapped.
```

Geben Sie zusätzliche Informationen an AWS-Konten, die in der Umgebung, in der das Bootstrapping ausgeführt wird, nach Informationen suchen können

Verwenden Sie die `--trust-for-lookup` Option, um anzugeben AWS-Konten, dass sie in der Umgebung, in der das Bootstrapping ausgeführt wird, nach Kontextinformationen suchen dürfen. Diese Option ist nützlich, um Konten die Erlaubnis zu geben, Stacks zu synthetisieren, die in der Umgebung bereitgestellt werden, ohne ihnen tatsächlich die Erlaubnis zu geben, diese Stacks direkt bereitzustellen.

Aktivieren Sie den Kündigungsschutz für den Bootstrap-Stack

Wenn ein Bootstrap-Stack gelöscht wird, werden die AWS Ressourcen, die ursprünglich in der Umgebung bereitgestellt wurden, ebenfalls gelöscht. Nach dem Bootstrapping Ihrer Umgebung empfehlen wir, den Bootstrap-Stack der Umgebung nicht zu löschen und neu zu erstellen, es sei denn, Sie tun dies absichtlich. Versuchen Sie stattdessen, den Bootstrap-Stack auf eine neue Version zu aktualisieren, indem Sie den Befehl erneut ausführen. `cdk bootstrap`

Verwenden Sie die `--termination-protection` Option, um die Einstellungen für den Kündigungsschutz für den Bootstrap-Stack zu verwalten. Durch die Aktivierung des Kündigungsschutzes verhindern Sie, dass der Bootstrap-Stack und seine Ressourcen versehentlich gelöscht werden. Dies ist besonders wichtig, wenn Sie CDK verwenden, Pipelines da es keine allgemeine Wiederherstellungsoption gibt, falls Sie den Bootstrap-Stack versehentlich löschen.

Nachdem Sie den Kündigungsschutz aktiviert haben, können Sie die AWS CLI oder die AWS CloudFormation OR-Konsole zur Überprüfung verwenden.

Um den Kündigungsschutz zu aktivieren

1. Führen Sie den folgenden Befehl aus, um den Kündigungsschutz für einen neuen oder vorhandenen Bootstrap-Stack zu aktivieren:

```
$ cdk bootstrap --termination-protection
```

2. Verwenden Sie die CloudFormation Konsole oder die AWS CLI, um dies zu überprüfen. Im Folgenden finden Sie ein Beispiel unter Verwendung der AWS CLI. Wenn Sie Ihren Bootstrap-Stack-Namen geändert haben, ersetzen Sie `CDKToolkit` durch Ihren Stack-Namen:

```
$ aws cloudformation describe-stacks --stack-name CDKToolkit --query  
"Stacks[0].EnableTerminationProtection"  
true
```

Ändern Sie die Standard-Bootstrap-Vorlage

Wenn Sie mehr Anpassungen benötigen, als das CDK CLI bieten kann, können Sie die Bootstrap-Vorlage nach Bedarf ändern. Stellen Sie dann die Vorlage bereit, um Ihre Umgebung zu booten.

Um die Standard-Bootstrap-Vorlage zu ändern und bereitzustellen

1. Rufen Sie die Standard-Bootstrap-Vorlage mithilfe der `--show-template` Option ab. Standardmäßig gibt das CDK CLI die Vorlage in Ihrem Terminalfenster aus. Sie können den CLI-Befehl ändern, um die Vorlage auf Ihrem lokalen Computer zu speichern. Im Folgenden wird ein Beispiel gezeigt:

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

2. Ändern Sie die Bootstrap-Vorlage nach Bedarf. Alle Änderungen, die Sie vornehmen, sollten dem Bootstrapping-Vorlagenvertrag entsprechen. Weitere Informationen zum Bootstrapping-Vorlagenvertrag finden Sie unter [Folgen Sie dem Bootstrap-Vorlagenvertrag](#)

Um sicherzustellen, dass Ihre Anpassungen später nicht versehentlich von einem Benutzer überschrieben werden, der die Standardvorlage `cdk bootstrap` verwendet, ändern Sie den Standardwert des Vorlagenparameters. `BootstrapVariant` Das CDK erlaubt nur CLI das Überschreiben des Bootstrap-Stacks mit Vorlagen, die dieselbe `BootstrapVariant` und eine gleiche oder eine höhere Version als die aktuell bereitgestellte Vorlage haben.

3. Stellen Sie Ihre geänderte Vorlage mit Ihrer bevorzugten AWS CloudFormation Bereitstellungsmethode bereit. Im Folgenden finden Sie ein Beispiel, das das CDK CLI verwendet:

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

Folgen Sie dem Bootstrap-Vorlagenvertrag

Wenn Sie das Bootstrapping anpassen, müssen Sie möglicherweise das Verhalten der Stack-Synthese anpassen. Dadurch wird sichergestellt, dass Ihre synthetisierte CloudFormation Vorlage mit Ihrem Bootstrap-Stack kompatibel bleibt. Weitere Informationen finden Sie unter [Passen Sie die CDK-Stack-Synthese an](#).

Die einfachste Methode zur Anpassung der Stack-Synthese besteht darin, die `DefaultStackSynthesizer` Klasse in Ihrer Stack Instanz zu ändern. Wenn Sie Anpassungen benötigen, die über das hinausgehen, was diese Klasse bieten kann, können Sie Ihren eigenen Synthesizer als eine Klasse schreiben, die implementiert [IStackSynthesizer](#) (vielleicht abgeleitet von `DefaultStackSynthesizer`).

Wenn Sie Bootstrapping anpassen, halten Sie sich an den Bootstrap-Vorlagenvertrag, um die Kompatibilität mit zu gewährleisten. `DefaultStackSynthesizer` Wenn Sie das Bootstrapping über den Bootstrap-Vorlagenvertrag hinaus ändern, müssen Sie Ihren eigenen Synthesizer schreiben.

Versionsverwaltung

Die Bootstrap-Vorlage sollte eine Ressource zum Erstellen eines Amazon EC2 Systems Manager (SSM) -Parameters mit einem bekannten Namen und einer Ausgabe enthalten, die die Version der Vorlage widerspiegelt:

```
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]
```

Rollen

Das `DefaultStackSynthesizer` erfordert fünf IAM-Rollen für fünf verschiedene Zwecke. Wenn Sie die Standardrollen nicht verwenden, müssen Sie Ihre IAM-Rollen-ARNs in Ihrem Objekt angeben. `DefaultStackSynthesizer` Die Rollen lauten wie folgt:

- Die Rolle der Bereitstellung wird vom CDK CLI und von AWS CodePipeline zur Bereitstellung in einer Umgebung übernommen. Es `AssumeRolePolicy` steuert, wer die Bereitstellung in der Umgebung durchführen kann. In der Vorlage können Sie sehen, welche Berechtigungen diese Rolle benötigt.
- Die Lookup-Rolle wird vom CDK übernommenCLI, um Kontext-Lookups in einer Umgebung durchzuführen. Es `AssumeRolePolicy` steuert, wer in der Umgebung bereitstellen kann. Die Berechtigungen, die diese Rolle benötigt, können der Vorlage entnommen werden.
- Die Rolle beim Veröffentlichen von Dateien und das Veröffentlichen von Bildern werden vom CDK CLI und von AWS CodeBuild Projekten zur Veröffentlichung von Assets in einer Umgebung übernommen. Sie werden verwendet, um in den Amazon S3 S3-Bucket bzw. das Amazon ECR-Repository zu schreiben. Diese Rollen benötigen Schreibzugriff auf diese Ressourcen.
- Die AWS CloudFormation Ausführungsrolle wird übergeben, AWS CloudFormation um die eigentliche Bereitstellung durchzuführen. Ihre Berechtigungen sind die Berechtigungen, unter

denen die Bereitstellung ausgeführt wird. Die Berechtigungen werden als Parameter, der ARNs für verwaltete Richtlinien auflistet, an den Stack übergeben.

Outputs

Das CDK CLI erfordert, dass die folgenden CloudFormation Ausgaben auf dem Bootstrap-Stack vorhanden sind:

- `BucketName`— Der Name des Datei-Asset-Buckets.
- `BucketDomainName`— Der Datei-Asset-Bucket im Domainnamenformat.
- `BootstrapVersion`— Die aktuelle Version des Bootstrap-Stacks.

Verlauf der Vorlage

Die Bootstrap-Vorlage ist versioniert und entwickelt sich im Laufe der Zeit mit der selbst. AWS CDK Wenn Sie Ihre eigene Bootstrap-Vorlage bereitstellen, halten Sie sie mit der kanonischen Standardvorlage auf dem neuesten Stand. Sie möchten sicherstellen, dass Ihre Vorlage weiterhin mit allen CDK-Funktionen funktioniert. Weitere Informationen finden Sie unter [Versionsverlauf der Bootstrap-Vorlage](#).

Konfiguration und Anpassung der CDK-Stack-Synthese

Bevor Sie einen AWS Cloud Development Kit (AWS CDK) Stack bereitstellen können, muss er zunächst synthetisiert werden. Bei der Stack-Synthese wird eine AWS CloudFormation Vorlage aus einem CDK-Stack erstellt. Die CloudFormation Vorlage stellen Sie dann bereit, um Ihre Ressourcen bereitzustellen. AWS

Themen

- [Konfigurieren Sie die CDK-Stack-Synthese](#)
- [Synthetisieren Sie einen CDK-Stack](#)
- [Passen Sie die CDK-Stack-Synthese an](#)

Konfigurieren Sie die CDK-Stack-Synthese

Sie können die CDK-Stack-Synthese mithilfe der Synthesizer-Eigenschaft Ihrer Instanz konfigurieren. [Stack](#) Diese Eigenschaft gibt an, wie die Vorlage des Stacks synthetisiert wird. Das Folgende ist ein einfaches Beispiel:

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
    })
});
```

Standardmäßig die AWS CDK Verwendungszwecke [DefaultStackSynthesizer](#) für die `synthesizer` Eigenschaft. Wenn Sie keinen Wert für `angebensynthesizer`, verwendet `DefaultStackSynthesizer` das CDK.

Damit die Synthese ordnungsgemäß funktioniert, muss das CDK über die Bootstrap-Ressourcen in der Umgebung Bescheid wissen, in der Sie sie bereitstellen werden. Dadurch wird sichergestellt, dass die Ressourcen aus Ihrer synthetisierten Vorlage korrekt mit Ressourcen aus dem Bootstrap-Stack interagieren.

Wenn Sie das Bootstrapping nicht geändert haben, z. B. Änderungen am Bootstrap-Stack oder an der Vorlage vorgenommen haben, müssen Sie die Stack-Synthese nicht ändern. Sie müssen nicht einmal die Synthesizer-Eigenschaft angeben. Das CDK verwendet die

`DefaultStackSynthesizer` Standardklasse, um die CDK-Stack-Synthese so zu konfigurieren, dass sie ordnungsgemäß mit Ihrem Bootstrap-Stack interagiert.

Synthetisieren Sie einen CDK-Stack

Verwenden Sie den AWS CDK Befehl Command Line Interface (`cdk synth`), um einen CDK-Stack zu synthetisieren. Weitere Informationen zu diesem Befehl, einschließlich Optionen, die Sie mit diesem Befehl verwenden können, finden Sie unter [cdk synth](#).

Wenn Ihre CDK-App einen einzelnen Stack enthält oder um alle Stapel zu synthetisieren, müssen Sie den CDK-Stacknamen nicht als Argument angeben. Standardmäßig synthetisiert das CDK CLI CDK-Stacks zu Vorlagen. AWS CloudFormation Eine JSON formatierte Vorlage für jeden Stapel wird im Verzeichnis gespeichert. `cdk.out` Wenn Ihre App einen einzelnen Stapel enthält, wird eine YAML formatierte Vorlage gedruckt. `stdout` Im Folgenden wird ein Beispiel gezeigt:

```
$ cdk synth
Resources:
  CDKMetadata:
    Type: AWS::CDK::Metadata
    Properties:
      Analytics: v2:deflate64:H4sIAAAAAAAAAA/
yXGyQ2AIBAAwFr4wwrEDugAK0DAZEGXhEMfxt6N8TWjQc0SJHNxEz5kseMK99Kdz9xsZGMro/
r43RQK2LHQw6mECK1Np5agFCiWGqKogzoeEezvC612NX1aAAAA
    Metadata:
      aws:cdk:path: CdkAppStack/CDKMetadata/Default
    Condition: CDKMetadataAvailable
  ...
```

Wenn Ihre CDK-App mehrere Stapel enthält, können Sie die logische ID eines Stacks angeben, um einen einzelnen Stapel zu synthetisieren. Im Folgenden wird ein Beispiel gezeigt:

```
$ cdk synth MyStackName
```

Wenn Sie keinen Stack synthetisieren und ausführen `cdk deploy`, synthetisiert das CDK Ihren Stack vor der Bereitstellung CLI automatisch.

Passen Sie die CDK-Stack-Synthese an

Wenn Sie das Bootstrapping ändern, müssen Sie möglicherweise die CDK-Stack-Synthese anpassen. Um die CDK-Stack-Synthese anzupassen, können Sie die `DefaultStackSynthesizer`

Eigenschaften in Ihrer Instanz ändern. Stack Wenn keine dieser Eigenschaften die von Ihnen benötigte Anpassung ermöglicht, können Sie Ihren Synthesizer als Klasse schreiben, die implementiert `IStackSynthesizer` (vielleicht abgeleitet von). `DefaultStackSynthesizer`

Ändern Sie den Qualifier

Der Qualifier wird dem Namen der Ressourcen hinzugefügt, die beim Bootstrapping erstellt wurden. Wenn Sie den Qualifier geändert haben, müssen Sie die CDK-Stack-Synthese so anpassen, dass derselbe Qualifier verwendet wird.

Um den Qualifier zu ändern, konfigurieren Sie den, `DefaultStackSynthesizer` indem Sie entweder den Synthesizer mit der Qualifier-Eigenschaft instanziiieren oder indem Sie den Qualifier als Kontextschlüssel in der Datei Ihres CDK-Projekts konfigurieren. `cdk.json`

Im Folgenden finden Sie ein Beispiel für die Instanziierung des Synthesizers mit der Eigenschaft: `qualifier`

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"
    })
});
```

Im Folgenden finden Sie ein Beispiel für die Konfiguration des Qualifizierers als Kontextschlüssel in: `cdk.json`

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

Ressourcennamen ändern

Alle anderen `DefaultStackSynthesizer` Eigenschaften beziehen sich auf die Namen der Ressourcen in der Bootstrap-Vorlage. Sie müssen diese Eigenschaften nur angeben, wenn Sie die Bootstrap-Vorlage und die Ressourcennamen oder das Benennungsschema geändert haben.

Alle Eigenschaften akzeptieren die speziellen Platzhalter `${Qualifier}`, `${AWS::Partition}${AWS::AccountId}`, und `${AWS::Region}`. Diese Platzhalter werden jeweils durch die Werte des `qualifier` Parameters und der AWS Partition, die Konto-ID und die AWS-Region Werte für die Umgebung des Stacks ersetzt.

Das folgende Beispiel zeigt die am häufigsten verwendeten Eigenschaften für `DefaultStackSynthesizer` zusammen mit ihren Standardwerten, als ob Sie den `Synthesizer` instanziierten würden. Eine vollständige Liste finden Sie unter: [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,
})
```

Entwickeln von AWS CDK Anwendungen

Entwickeln Sie AWS Cloud Development Kit (AWS CDK) Anwendungen.

Themen

- [Anpassen von Konstrukten aus der AWS Construct Library](#)
- [Abrufen eines Werts aus einer Umgebungsvariablen](#)
- [Verwenden eines - AWS CloudFormation Werts](#)
- [Eine bestehende AWS CloudFormation Vorlage importieren](#)
- [Abrufen eines Werts aus dem Systems Manager Parameter Store](#)
- [Abrufen eines Werts von AWS Secrets Manager](#)
- [Einen CloudWatch Alarm festlegen](#)
- [Speichern und Abrufen von Kontextvariablenwerten](#)
- [Verwenden von Ressourcen aus der AWS CloudFormation öffentlichen Registrierung](#)

Anpassen von Konstrukten aus der AWS Construct Library

Passen Sie Konstrukte aus der AWS Construct Library mithilfe von Escape-Schraffuren, Rohüberschreibungen und benutzerdefinierten Ressourcen an.

Themen

- [Verwenden von Escape-Schraffuren](#)
- [Escape-Schraffuren entfernen](#)
- [Rohüberschreibungen](#)
- [Benutzerdefinierte Ressourcen](#)

Verwenden von Escape-Schraffuren

Die AWS Construct Library bietet [Konstrukte](#) unterschiedlicher Abstraktionsstufen.

Auf der höchsten Ebene sind Ihre AWS CDK Anwendung und die darin enthaltenen Stacks selbst Abstraktionen Ihrer gesamten Cloud-Infrastruktur oder erhebliche Teile davon. Sie können parametrisiert werden, um sie in verschiedenen Umgebungen oder für unterschiedliche Anforderungen bereitzustellen.

Abstraktionen sind leistungsstarke Tools zum Entwerfen und Implementieren von Cloud-Anwendungen. Die AWS CDK bietet Ihnen nicht nur die Möglichkeit, mit ihren Abstraktionen zu erstellen, sondern auch neue Abstraktionen zu erstellen. Mithilfe der vorhandenen Open-Source-L2- und L3-Konstrukte als Leitfaden können Sie Ihre eigenen L2- und L3-Konstrukte erstellen, um die bewährten Methoden und Meinungen Ihrer eigenen Organisation widerzuspiegeln.

Keine Abstraktion ist perfekt, und selbst gute Abstraktionen können nicht jeden möglichen Anwendungsfall abdecken. Während der Entwicklung finden Sie möglicherweise ein Konstrukt, das fast Ihren Anforderungen entspricht und eine kleine oder große Anpassung erfordert.

Aus diesem Grund AWS CDK bietet die Möglichkeiten, das Konstruktmodell zu verlassen. Dazu gehört der Wechsel zu einer Abstraktion auf niedrigerer Ebene oder zu einem anderen Modell. Mit Escape-Schraffuren können Sie das AWS CDK Paradigma umgehen und es an Ihre Bedürfnisse anpassen. Anschließend können Sie Ihre Änderungen in ein neues Konstrukt einpacken, um die zugrunde liegende Komplexität zu abstrahieren und eine saubere API für andere Entwickler bereitzustellen.

Im Folgenden finden Sie Beispiele für Situationen, in denen Sie Escape-Schraffuren verwenden können:

- Ein - AWS Service-Feature ist über verfügbar AWS CloudFormation, aber es gibt dafür keine L2-Konstrukte.
- Ein - AWS Service-Feature ist über verfügbar AWS CloudFormation, und es gibt L2-Konstrukte für den Service, aber diese stellen das Feature noch nicht bereit. Da L2-Konstrukte vom CDK-Team kuratiert werden, sind sie möglicherweise nicht sofort für neue Funktionen verfügbar.
- Die Funktion ist noch nicht über verfügbar AWS CloudFormation .

Informationen dazu, ob ein Feature über verfügbar ist AWS CloudFormation, finden Sie unter [AWS Ressourcen- und Eigenschaftstypen – Referenz](#).

Entwickeln von Escape-Schraffuren für L1-Konstrukte

Wenn für den Service keine L2-Konstrukte verfügbar sind, können Sie die automatisch generierten L1-Konstrukte verwenden. Diese Ressourcen können anhand ihres Namens erkannt werden, der mit `beginntCfn`, z. B. `CfnBucket` oder `CfnRole`. Sie instanziiieren sie genau so, wie Sie die entsprechende AWS CloudFormation Ressource verwenden würden.

Um beispielsweise einen Low-Level-Amazon S3-Bucket L1 mit aktivierter Analyse zu instanziiieren, würden Sie etwas wie das Folgende schreiben.

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",
    // ...
}
});
```

Es kann selten vorkommen, dass Sie eine Ressource definieren möchten, die keine entsprechende `CfnXxx` Klasse hat. Dies könnte ein neuer Ressourcentyp sein, der noch nicht in der AWS CloudFormation Ressourcenspezifikation veröffentlicht wurde. In solchen Fällen können Sie die `cdk.CfnResource` direkt instanzieren und den Ressourcentyp und die Eigenschaften angeben. Dies wird im folgenden Beispiel veranschaulicht.

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"
            }
        }
    }
}
```

```
});
```

Entwickeln von Escape-Schraffuren für L2-Konstrukte

Wenn einem L2-Konstrukt ein Feature fehlt oder Sie versuchen, ein Problem zu umgehen, können Sie das L1-Konstrukt ändern, das durch das L2-Konstrukt gekapselt ist.

Alle L2-Konstrukte enthalten darin das entsprechende L1-Konstrukt. Beispielsweise umschließt das High-Level-BucketKonstrukt das Low-Level-CfnBucketKonstrukt. Da der direkt der AWS CloudFormation Ressource CfnBucket entspricht, werden alle Funktionen bereitgestellt, die über verfügbar sind AWS CloudFormation.

Der grundlegende Ansatz, um Zugriff auf das L1-Konstrukt zu erhalten, besteht darin, `construct.node.defaultChild` (Python: `default_child`) zu verwenden, es in den richtigen Typ umzuwandeln (falls erforderlich) und seine Eigenschaften zu ändern. Betrachten wir erneut das Beispiel eines 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",  
        // ...  
    }  
};
```

Sie können dieses Objekt auch verwenden, um AWS CloudFormation Optionen wie Metadata und zu ändernUpdatePolicy.

TypeScript

```
cfBucket.cfnOptions.metadata = {  
  MetadataKey: 'MetadataValue'  
};
```

JavaScript

```
cfBucket.cfnOptions.metadata = {  
  MetadataKey: 'MetadataValue'  
};
```

Python

```
cf_bucket.cfn_options.metadata = {  
    "MetadataKey": "MetadataValue"  
}
```

Java

```
cfBucket.getCfnOptions().setMetadata(java.util.Map.of( // Java 9+  
    "MetadataKey", "Metadatavalue"));
```

C#

```
cfBucket.CfnOptions.Metadata = new Dictionary<string, object>  
{  
    ["MetadataKey"] = "Metadatavalue"  
};
```

Escape-Schraffuren entfernen

Die bietet AWS CDK auch die Möglichkeit, eine Abstraktionsstufe zu erreichen, die wir möglicherweise als „Entflief“-Schraffur bezeichnen. Wenn Sie ein L1-Konstrukt haben, z. B. `CfnBucket`, können Sie ein neues L2-Konstrukt (Bucket in diesem Fall) erstellen, um das L1-Konstrukt zu umschließen.

Dies ist praktisch, wenn Sie eine L1-Ressource erstellen, sie aber mit einem Konstrukt verwenden möchten, das eine L2-Ressource erfordert. Es ist auch hilfreich, wenn Sie Convenience-Methoden wie `grantXxxxx()` verwenden möchten, die auf dem L1-Konstrukt nicht verfügbar sind.

Sie wechseln mit einer statischen Methode in der L2-Klasse auf die höhere Abstraktionsstufe, z. B. `Bucket.fromCfnBucket()` für Amazon S3 `fromCfnXXXX()`-Buckets. Die L1-Ressource ist der einzige Parameter.

TypeScript

```
b1 = new s3.CfnBucket(this, "buck09", { ... });
b2 = s3.Bucket.fromCfnBucket(b1);
```

JavaScript

```
b1 = new s3.CfnBucket(this, "buck09", { ...} );
b2 = s3.Bucket.fromCfnBucket(b1);
```

Python

```
b1 = s3.CfnBucket(self, "buck09", ...)
b2 = s3.from_cfn_bucket(b1)
```

Java

```
CfnBucket b1 = CfnBucket.Builder.create(this, "buck09")
    // ....
    .build();
IBucket b2 = Bucket.fromCfnBucket(b1);
```

C#

```
var b1 = new CfnBucket(this, "buck09", new CfnBucketProps { ... });
var v2 = Bucket.FromCfnBucket(b1);
```

L2-Konstrukte, die aus L1-Konstrukten erstellt wurden, sind Proxy-Objekte, die sich auf die L1-Ressource beziehen, ähnlich denen, die aus Ressourcennamen, ARNs oder Nachschlagevorgängen erstellt wurden. Änderungen an diesen Konstrukten wirken sich nicht auf die endgültige synthetisierte AWS CloudFormation Vorlage aus (da Sie jedoch über die L1-Ressource verfügen, können Sie diese stattdessen ändern). Weitere Informationen zu Proxy-Objekten finden Sie unter [the section called “Verweisen auf Ressourcen in Ihrem Konto AWS”](#).

Um Verwirrung zu vermeiden, erstellen Sie nicht mehrere L2-Konstrukte, die sich auf dasselbe L1-Konstrukt beziehen. Wenn Sie beispielsweise das Bucket mit der -Technik im vorherigen Abschnitt `CfnBucket` aus einem extrahieren, sollten Sie keine zweite Instance erstellen, Bucket indem Sie `Bucket.fromCfnBucket()` mit diesem aufrufen `CfnBucket`. [???](#) Es funktioniert tatsächlich wie erwartet (nur einer `AWS::S3::Bucket` ist synthetisiert), aber es macht Ihren Code schwieriger zu verwalten.

Rohüberschreibungen

Wenn Eigenschaften im L1-Konstrukt fehlen, können Sie die gesamte Typisierung mithilfe von Rohüberschreibungen umgehen. Dadurch ist es auch möglich, synthetisierte Eigenschaften zu löschen.

Verwenden Sie eine der `addOverride` Methoden (Python: `add_override`), wie im folgenden Beispiel gezeigt.

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");
```

Benutzerdefinierte Ressourcen

Wenn die Funktion nicht über AWS CloudFormation, sondern nur über einen direkten API-Aufruf verfügbar ist, müssen Sie eine AWS CloudFormation benutzerdefinierte -Ressource schreiben, um den benötigten API-Aufruf durchzuführen. Sie können die verwenden AWS CDK , um benutzerdefinierte Ressourcen zu schreiben und sie in eine reguläre Konstruktschnittstelle zu packen. Aus Sicht eines Konsumenten Ihres Konstrukts wird sich die Erfahrung nativ anhören.

Beim Erstellen einer benutzerdefinierten Ressource muss eine Lambda-Funktion geschrieben werden UPDATE, die auf die DELETE Lebenszyklusereignisse CREATE, und einer Ressource reagiert.

Wenn Ihre benutzerdefinierte Ressource nur einen einzigen API-Aufruf tätigen muss, sollten Sie die verwenden [AwsCustomResource](#). Dadurch ist es möglich, beliebige SDK-Aufrufe während einer - AWS CloudFormation Bereitstellung durchzuführen. Andernfalls sollten Sie Ihre eigene Lambda-Funktion schreiben, um die Arbeit auszuführen, die Sie erledigen müssen.

Das Thema ist hier zu breit, um es vollständig abzudecken, aber die folgenden Links sollten Ihnen den Einstieg erleichtern:

- [Angepasste Ressourcen](#)
- [Beispiel für benutzerdefinierte Ressourcen](#)
- Ein vollwertigeres Beispiel finden Sie in der [DnsValidatedCertificate](#) Klasse in der CDK-Standardbibliothek. Dies wird als benutzerdefinierte Ressource implementiert.

Abrufen eines Werts aus einer Umgebungsvariablen

Um den Wert einer Umgebungsvariablen abzurufen, verwenden Sie Code wie den folgenden. Dieser Code ruft den Wert der Umgebungsvariablen abMYBUCKET.

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";
```

Verwenden eines - AWS CloudFormation Werts

Informationen [the section called "Parameter"](#) zur Verwendung von AWS CloudFormation Parametern mit der finden Sie unter AWS CDK.

Informationen zum Abrufen eines Verweises auf eine Ressource in einer vorhandenen AWS CloudFormation Vorlage finden Sie unter [the section called "Eine AWS CloudFormation Vorlage importieren"](#).

Eine bestehende AWS CloudFormation Vorlage importieren

Importieren Sie Ressourcen aus einer AWS CloudFormation Vorlage in Ihre AWS Cloud Development Kit (AWS CDK) Anwendungen, indem Sie das [cloudformation-include.CfnInclude](#) Konstrukt verwenden, um Ressourcen in L1-Konstrukte zu konvertieren.

Nach dem Import können Sie mit diesen Ressourcen in Ihrer App genauso arbeiten, als ob sie ursprünglich im AWS CDK Code definiert wären. Sie können diese L1-Konstrukte auch in Konstrukten auf höherer AWS CDK Ebene verwenden. Auf diese Weise können Sie beispielsweise die L2-Methoden zur Erteilung von Berechtigungen mit den von ihnen definierten Ressourcen verwenden.

Das `cloudformation-include.CfnInclude` Konstrukt fügt im Wesentlichen jeder Ressource in Ihrer AWS CloudFormation Vorlage einen AWS CDK API-Wrapper hinzu. Verwenden Sie diese Funktion, um Ihre vorhandenen AWS CloudFormation Vorlagen Stück für Stück in die AWS CDK Datei zu importieren. Auf diese Weise können Sie Ihre vorhandenen Ressourcen mithilfe von AWS CDK Konstrukten verwalten, um die Vorteile von Abstraktionen auf höherer Ebene zu nutzen. Sie können diese Funktion auch verwenden, um Ihre AWS CloudFormation Vorlagen an AWS CDK Entwickler zu verkaufen, indem Sie eine Konstrukt-API bereitstellen. AWS CDK

Note

AWS CDK Version 1 ist ebenfalls enthalten [aws-cdk-lib.CfnInclude](#), das zuvor für denselben allgemeinen Zweck verwendet wurde. Es fehlt jedoch ein Großteil der Funktionalität von `cloudformation-include.CfnInclude`.

Themen

- [Eine AWS CloudFormation Vorlage importieren](#)
- [Zugreifen auf importierte Ressourcen](#)
- [Parameter ersetzen](#)
- [Andere Vorlagenelemente](#)
- [Verschachtelte Stacks](#)

Eine AWS CloudFormation Vorlage importieren

Im Folgenden finden Sie eine AWS CloudFormation Beispielvorlage, anhand derer wir Beispiele für dieses Thema bereitstellen werden. Kopieren und speichern Sie die Vorlage `my-template.json`, um den Anweisungen zu folgen. Nachdem Sie diese Beispiele durchgearbeitet haben, können Sie weitere Informationen verwenden, indem Sie eine Ihrer vorhandenen bereitgestellten AWS CloudFormation Vorlagen verwenden. Sie können sie über die AWS CloudFormation Konsole abrufen.

```
{
  "Resources": {
    "MyBucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {
        "BucketName": "MyBucket",
      }
    }
  }
}
```

Sie können entweder mit JSON- oder YAML-Vorlagen arbeiten. Wir empfehlen JSON, sofern verfügbar, da YAML-Parser leicht variieren können, was sie akzeptieren.

Im Folgenden finden Sie ein Beispiel dafür, wie Sie die Beispielvorlage mithilfe von in Ihre AWS CDK App importieren können. `cloudformation-include` Vorlagen werden im Kontext eines CDK-Stacks importiert.

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"
            });
        }
    }
}
```

Standardmäßig wird beim Importieren einer Ressource die ursprüngliche logische ID der Ressource aus der Vorlage beibehalten. Dieses Verhalten eignet sich für den Import einer AWS CloudFormation Vorlage in die AWS CDK, wobei logische IDs beibehalten werden müssen. AWS CloudFormation benötigt diese Informationen, um diese importierten Ressourcen als dieselben Ressourcen aus der AWS CloudFormation Vorlage zu erkennen.

Wenn Sie einen AWS CDK Konstrukt-Wrapper für die Vorlage entwickeln, sodass er von anderen AWS CDK Entwicklern verwendet werden kann, lassen Sie sie stattdessen neue Ressourcen-

IDs AWS CDK generieren. Auf diese Weise kann das Konstrukt ohne Namenskonflikte mehrfach in einem Stapel verwendet werden. Setzen Sie dazu `false` beim Import der Vorlage die `preserveLogicalIds` Eigenschaft auf. Im Folgenden wird ein Beispiel gezeigt:

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
});
```

Um importierte Ressourcen unter die Kontrolle Ihrer AWS CDK App zu stellen, fügen Sie den Stack zuApp:

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");
        }
    }
}
```

Um sicherzustellen, dass keine unbeabsichtigten Änderungen an den AWS Ressourcen im Stack vorgenommen werden, können Sie einen Vergleich durchführen. Verwenden Sie den AWS CDK CLI `cdk diff` Befehl und lassen Sie alle AWS CDK-spezifischen Metadaten weg. Im Folgenden wird ein Beispiel gezeigt:

```
cdk diff --no-version-reporting --no-path-metadata --no-asset-metadata
```

Nachdem Sie eine AWS CloudFormation Vorlage importiert haben, sollte die AWS CDK App zur Informationsquelle für Ihre importierten Ressourcen werden. Um Änderungen an Ihren Ressourcen vorzunehmen, ändern Sie sie in Ihrer AWS CDK App und stellen Sie sie mit dem AWS CDK CLI `cdk deploy` Befehl bereit.

Zugreifen auf importierte Ressourcen

Der Name `template` im Beispielcode steht für die importierte AWS CloudFormation Vorlage. Verwenden Sie die [`getResource\(\)`](#) Methode des Objekts, um von dort aus auf eine Ressource zuzugreifen. Um auf die zurückgegebene Ressource als eine bestimmte Art von Ressource zuzugreifen, wandeln Sie das Ergebnis in den gewünschten Typ um. Dies ist in Python oder nicht erforderlich JavaScript. Im Folgenden wird ein Beispiel gezeigt:

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");
```

Aus diesem Beispiel `cfnBucket` geht es jetzt um eine Instanz der [aws-s3.CfnBucket](#) Klasse. Dies ist ein L1-Konstrukt, das die entsprechende AWS CloudFormation Ressource darstellt. Sie können es wie jede andere Ressource dieses Typs behandeln. Sie können beispielsweise seinen ARN-Wert mit der `bucket.attrArn` Eigenschaft abrufen.

Um die `CfnBucket` L1-Ressource stattdessen in eine [aws-s3.Bucket](#) L2-Instanz zu packen, verwenden Sie die statischen Methoden [fromBucketArn\(\)](#), [fromBucketAttributes\(\)](#), oder [fromBucketName\(\)](#). In der Regel ist die `fromBucketName()` Methode am praktischsten. Im Folgenden wird ein Beispiel gezeigt:

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);
```

Andere L2-Konstrukte verwenden ähnliche Methoden, um das Konstrukt aus einer vorhandenen Ressource zu erstellen.

Wenn Sie ein L1-Konstrukt in ein L2-Konstrukt einbinden, wird dadurch keine neue Ressource erstellt. In unserem Beispiel erstellen wir keinen zweiten S3; -Bucket. Stattdessen kapselt die neue Bucket Instanz die bestehende. CfnBucket

Aus dem Beispiel geht hervor, dass `bucket` es sich nun um ein Bucket L2-Konstrukt handelt, das sich wie jedes andere L2-Konstrukt verhält. Beispielsweise können Sie einer AWS Lambda Funktion Schreibzugriff auf den Bucket gewähren, indem Sie die praktische Methode des Buckets verwenden. [grantWrite\(\)](#) Sie müssen die erforderliche AWS Identity and Access Management (IAM-) Richtlinie nicht manuell definieren. Im Folgenden wird ein Beispiel gezeigt:

TypeScript

```
bucket.grantWrite(lambdaFunc);
```

JavaScript

```
bucket.grantWrite(lambdaFunc);
```

Python

```
bucket.grant_write(lambda_func)
```

Java

```
bucket.grantWrite(lambdaFunc);
```

C#

```
bucket.GrantWrite(lambdaFunc);
```

Parameter ersetzen

Wenn Ihre AWS CloudFormation Vorlage Parameter enthält, können Sie diese beim Import mithilfe der `parameters` Eigenschaft durch Werte für die Erstellungszeit ersetzen. Im folgenden Beispiel ersetzen wir den `UploadBucket` Parameter durch den ARN eines Buckets, der an anderer Stelle in unserem AWS CDK Code definiert ist.

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 }
    }
});
```

Andere Vorlagenelemente

Sie können jedes AWS CloudFormation Vorlagenelement importieren, nicht nur Ressourcen. Die importierten Elemente werden Teil des AWS CDK Stapels. Verwenden Sie die folgenden Methoden des `CfnInclude` Objekts, um diese Elemente zu importieren:

- [getCondition\(\)](#)— AWS CloudFormation [Bedingungen](#).
- [getHook\(\)](#)— AWS CloudFormation [Hooks](#) für blaue/grüne Bereitstellungen.
- [getMapping\(\)](#)— [Zuordnungen AWS CloudFormation](#) .
- [getOutput\(\)](#)— [Ausgänge AWS CloudFormation](#) .
- [getParameter\(\)](#)— AWS CloudFormation [Parameter](#).
- [getRule\(\)](#)— AWS CloudFormation [Regeln](#) für AWS Service Catalog Vorlagen.

Jede dieser Methoden gibt eine Instanz einer Klasse zurück, die den spezifischen AWS CloudFormation Elementtyp repräsentiert. Diese Objekte sind veränderbar. Änderungen, die Sie an ihnen vornehmen, werden in der Vorlage angezeigt, die aus dem AWS CDK Stapel generiert wird. Im Folgenden finden Sie ein Beispiel, das einen Parameter aus der Vorlage importiert und seinen Standardwert ändert:

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.setDefaultValues("AWS CDK")
```

C#

```
var cfnBucket = (CfnBucket)template.GetResource("MyBucket");  
var param = template.GetParameter("MyParameter");  
param.Default = "AWS CDK";
```

Verschachtelte Stacks

Sie können [verschachtelte Stapel](#) importieren, indem Sie sie entweder beim Import ihrer Hauptvorlage oder zu einem späteren Zeitpunkt angeben. Die verschachtelte Vorlage muss in einer lokalen Datei gespeichert werden, in der Hauptvorlage jedoch als `NestedStack` Ressource referenziert werden. Außerdem muss der im AWS CDK Code verwendete Ressourcenname mit dem Namen übereinstimmen, der für den verschachtelten Stapel in der Hauptvorlage verwendet wird.

Angesichts dieser Ressourcendefinition in der Hauptvorlage zeigt der folgende Code, wie der referenzierte verschachtelte Stapel in beide Richtungen importiert wird.

```
"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'
    });
```

Sie können mehrere verschachtelte Stapel mit beiden Methoden importieren. Wenn Sie die Hauptvorlage importieren, stellen Sie eine Zuordnung zwischen dem Ressourcennamen jedes verschachtelten Stapels und seiner Vorlagendatei bereit. Diese Zuordnung kann eine beliebige

Anzahl von Einträgen enthalten. Rufen Sie dies nach dem ersten Import für jeden verschachtelten Stapel `loadNestedStack()` einmal auf.

Nach dem Import eines verschachtelten Stacks können Sie mit der Methode der Hauptvorlage darauf zugreifen. [getNestedStack\(\)](#)

TypeScript

```
const nestedStack = mainTemplate.getNestedStack('NestedStack').stack;
```

JavaScript

```
const nestedStack = mainTemplate.getNestedStack('NestedStack').stack;
```

Python

```
nested_stack = main_template.get_nested_stack("NestedStack").stack
```

Java

```
NestedStack nestedStack = mainTemplate.getNestedStack("NestedStack").getStack();
```

C#

```
var nestedStack = mainTemplate.GetNestedStack("NestedStack").Stack;
```

Die `getNestedStack()` Methode gibt eine [IncludedNestedStack](#) Instanz zurück. Von dieser Instanz aus können Sie über die `stack` Eigenschaft auf die AWS CDK [NestedStack](#) Instanz zugreifen, wie im Beispiel gezeigt. Sie können auch über auf das ursprüngliche AWS CloudFormation Vorlagenobjekt zugreifen `includedTemplate`, aus dem Sie Ressourcen und andere AWS CloudFormation Elemente laden können.

Abrufen eines Werts aus dem Systems Manager Parameter Store

Der AWS Cloud Development Kit (AWS CDK) kann den Wert von AWS Systems Manager Parameter Store-Attributen abrufen. Während der Generierung AWS CDK erzeugt ein [Token](#), das von AWS CloudFormation während der Bereitstellung aufgelöst wird.

unterstützt AWS CDK das Abrufen von einfachen und sicheren Werten. Sie können eine bestimmte Version beider Arten von Werten anfordern. Bei reinen Werten können Sie die Version aus Ihrer Anfrage weglassen, um die neueste Version abzurufen. Für sichere Werte müssen Sie die Version angeben, wenn Sie den Wert des sicheren Attributs anfordern.

Note

In diesem Thema wird gezeigt, wie Attribute aus dem AWS Systems Manager Parameter Store gelesen werden. Sie können Secrets auch aus der lesen AWS Secrets Manager (siehe [Abrufen eines Werts von AWS Secrets Manager](#)).

Themen

- [Lesen von Systems Manager-Werten zur Bereitstellungszeit](#)
- [Systems Manager-Werte zur Synthetisierungszeit lesen](#)
- [Schreiben von Werten in Systems Manager](#)

Lesen von Systems Manager-Werten zur Bereitstellungszeit

Um Werte aus dem Systems Manager Parameter Store zu lesen, verwenden Sie die [valueForSecureStringParameter](#) Methoden [valueForStringParameter](#) und [valueForSecureStringParameter](#). Wählen Sie eine Methode aus, je nachdem, ob das gewünschte Attribut eine einfache Zeichenfolge oder ein sicherer Zeichenfolgenwert ist. Diese Methoden geben [Token](#) zurück, nicht den tatsächlichen Wert. Der Wert wird von AWS CloudFormation während der Bereitstellung aufgelöst. Im Folgenden wird ein Beispiel gezeigt:

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
```

Derzeit wird diese Funktion von einer [begrenzten Anzahl von - AWS Services](#) unterstützt.

Systems Manager-Werte zur Synthetisierungszeit lesen

Manchmal ist es nützlich, zur Synthetisierungszeit einen Parameter anzugeben. Auf diese Weise verwendet die AWS CloudFormation Vorlage immer denselben Wert, anstatt den Wert während der Bereitstellung aufzulösen.

Verwenden Sie die [-valueFromLookup](#) Methode (Python: `ValueFromLookup`), um einen Wert aus dem Systems Manager Parameter Store zur Synthetisierungszeit zu lesen `value_from_lookup`. Diese Methode gibt den tatsächlichen Wert des Parameters als [the section called "Kontext"](#) Wert zurück. Wenn der Wert noch nicht in der Befehlszeile zwischengespeichert `cdk.json` oder an die Befehlszeile übergeben ist, wird er vom aktuellen AWS Konto abgerufen. Aus diesem Grund muss der Stack mit expliziten AWS Umgebungsinformationen synthetisiert werden.

Im Folgenden wird ein Beispiel gezeigt:

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");
```

Es können nur einfache Systems Manager-Zeichenfolgen abgerufen werden. Sichere Zeichenfolgen können nicht abgerufen werden. Die neueste Version wird immer zurückgegeben. Bestimmte Versionen können nicht angefordert werden.

Important

Der abgerufene Wert wird in Ihrer synthetisierten AWS CloudFormation Vorlage angezeigt. Dies kann ein Sicherheitsrisiko sein, je nachdem, wer Zugriff auf Ihre AWS CloudFormation Vorlagen hat und um welche Art von Wert es sich handelt. Verwenden Sie diese Funktion im Allgemeinen nicht für Passwörter, Schlüssel oder andere Werte, die Sie privat halten möchten.

Schreiben von Werten in Systems Manager

Sie können die AWS CLI, die oder ein AWS SDK verwenden AWS Management Console, um Systems Manager-Parameterwerte festzulegen. In den folgenden Beispielen wird der CLI-Befehl [ssm put-parameter](#) verwendet.

```
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"
```

Wenn Sie einen bereits vorhandenen SSM-Wert aktualisieren, schließen Sie auch die `--overwrite` Option ein.

```
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"
```

Abrufen eines Werts von AWS Secrets Manager

Um Werte aus AWS Secrets Manager in Ihrer AWS CDK App zu verwenden, verwenden Sie die [fromSecretAttributes\(\)](#)-Methode. Es stellt einen Wert dar, der von Secrets Manager abgerufen und zum Zeitpunkt der AWS CloudFormation Bereitstellung verwendet wird. Im Folgenden wird ein Beispiel gezeigt:

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 = ...,
        });
    }
}

```

i Tip

Verwenden Sie den CLI-Befehl AWS CLI [create-secret](#), um ein Secret aus der Befehlszeile zu erstellen, z. B. beim Testen:

```
aws secretsmanager create-secret --name ImportedSecret --secret-string  
mygroovybucket
```

Der Befehl gibt einen ARN zurück, den Sie mit dem vorherigen Beispiel verwenden können.

Sobald Sie eine Secret Instance erstellt haben, können Sie den Wert des Secrets aus dem `secretValue` Attribut der Instance abrufen. Der Wert wird durch eine [SecretValue](#) Instance dargestellt, einen speziellen Typ von [the section called "Token"](#). Da es sich um ein Token handelt, hat es erst nach der Auflösung eine Bedeutung. Ihre CDK-App muss nicht auf ihren tatsächlichen Wert zugreifen. Stattdessen kann die App die `SecretValue` Instance (oder ihre Zeichenfolge oder numerische Darstellung) an eine beliebige CDK-Methode übergeben, die den Wert benötigt.

Einen CloudWatch Alarm festlegen

Verwenden Sie das [aws-cloudwatch](#)-Paket, um Amazon- CloudWatch Alarme für CloudWatch Metriken einzurichten. Sie können vordefinierte Metriken verwenden oder Ihre eigenen erstellen.

Themen

- [Verwenden einer vorhandenen Metrik](#)
- [Erstellen Ihrer eigenen Metrik](#)
- [Erstellen des Alarms](#)

Verwenden einer vorhandenen Metrik

Mit vielen Modulen AWS der Construct Library können Sie einen Alarm für eine vorhandene Metrik festlegen, indem Sie den Namen der Metrik an eine Convenience-Methode auf einer Instance eines Objekts mit Metriken übergeben. Bei einer Amazon SQS-Warteschlange können Sie beispielsweise die Metrik `ApproximateNumberOfMessagesVisible` aus der [Metrik\(\)](#)-Methode der Warteschlange abrufen:

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");
```

Erstellen Ihrer eigenen Metrik

Erstellen Sie Ihre eigene [Metrik](#) wie folgt, wobei der Namespace-Wert in etwa AWS/SQS für eine Amazon SQS-Warteschlange sein sollte. Sie müssen auch den Namen und die Dimension Ihrer Metrik angeben:

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" }
    }
});
```

Erstellen des Alarms

Sobald Sie eine Metrik haben, entweder eine vorhandene oder eine, die Sie definiert haben, können Sie einen Alarm erstellen. In diesem Beispiel wird der Alarm ausgelöst, wenn in zwei der letzten drei Auswertungszeiträume mehr als 100 Ihrer Metrik vorhanden sind. Sie können Vergleiche wie „weniger als“ in Ihren Alarmen über die `-comparisonOperator`Eigenschaft verwenden. `Greater-than-or-equal-to` ist die AWS CDK Standardeinstellung, daher müssen wir sie nicht angeben.

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
});
```

Eine alternative Möglichkeit, einen Alarm zu erstellen, ist die Verwendung der [createAlarm\(\)](#)-Methode der Metrik, die im Wesentlichen dieselben Eigenschaften wie der Alarm Konstruktor hat. Sie müssen die Metrik nicht übergeben, da sie bereits bekannt ist.

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
});
```

Speichern und Abrufen von Kontextvariablenwerten

Sie können Kontextvariablen mit der AWS Cloud Development Kit (AWS CDK) CLI oder in der `cdk.json` Datei angeben. Verwenden Sie dann die `-TryGetContext` Methode, um Werte abzurufen.

Themen

- [Angeben von Kontextvariablen](#)
- [Abrufen von Kontextvariablenwerten](#)

Angeben von Kontextvariablen

Sie können eine Kontextvariable entweder als Teil eines AWS CDK CLI -Befehls oder in `cdk.json` angeben.

Um eine Befehlszeilenkontextvariable zu erstellen, verwenden Sie die Option `---context (-c)`, wie im folgenden Beispiel gezeigt.

```
cdk synth -c bucket_name=mygroovybucket
```

Verwenden Sie den folgenden Code, um dieselbe Kontextvariable und denselben Wert in der `cdk.json` Datei anzugeben.

```
{
  "context": {
    "bucket_name": "myotherbucket"
  }
}
```

```
}  
}
```

Wenn Sie eine Kontextvariable sowohl mit der AWS CDK CLI Datei als auch mit der `cdk.json` Datei angeben, hat der AWS CDK CLI Wert Vorrang.

Abrufen von Kontextvariablenwerten

Um den Wert einer Kontextvariablen in Ihrer App abzurufen, verwenden Sie die `tryGetContext` Methode im Kontext eines Konstrukts. (Das heißt, wenn `this` oder `self` in Python eine Instance eines Konstrukts ist.)

In diesem Beispiel rufen wir den Wert der `bucket_name` Kontextvariablen ab. Wenn der angeforderte Wert nicht definiert ist, `tryGetContext` gibt `undefined` (`None` in Python; `null` in Java und C#; `nil` in Go) zurück, anstatt eine Ausnahme auszulösen.

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");
```

Außerhalb des Kontexts eines Konstrukts können Sie wie folgt über das App-Objekt auf die Kontextvariable zugreifen.

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");
```

Weitere Informationen zum Arbeiten mit Kontextvariablen finden Sie unter [the section called "Kontext"](#).

Verwenden von Ressourcen aus der AWS CloudFormation öffentlichen Registrierung

Mit der AWS CloudFormation öffentlichen Registrierung können Sie Erweiterungen sowohl öffentlich als auch privat verwalten, z. B. Ressourcen, Module und Hooks, die für die Verwendung in Ihrem verfügbar sind AWS-Konto. Sie können Erweiterungen öffentlicher Ressourcen in Ihren AWS Cloud Development Kit (AWS CDK) Anwendungen mit dem [CfnResource](#) Konstrukt verwenden.

Weitere Informationen zur AWS CloudFormation öffentlichen Registrierung finden Sie unter [Verwenden der AWS CloudFormation Registrierung](#) im AWS CloudFormation -Benutzerhandbuch.

Alle von veröffentlichten öffentlichen Erweiterungen AWS sind für alle Konten in allen Regionen verfügbar, ohne dass Sie etwas unternehmen müssen. Sie müssen jedoch jede Erweiterung eines Drittanbieters aktivieren, die Sie verwenden möchten, in jedem Konto und jeder Region, in der Sie sie verwenden möchten.

Note

Wenn Sie AWS CloudFormation mit Ressourcentypen von Drittanbietern verwenden, fallen Gebühren an. Die Gebühren basieren auf der Anzahl der Handler-Operationen, die Sie pro Monat ausführen, und der Dauer des Handler-Vorgangs. Vollständige Informationen finden Sie unter [-CloudFormation Preise](#).

Weitere Informationen zu öffentlichen Erweiterungen finden Sie unter [Verwenden von öffentlichen Erweiterungen in CloudFormation](#) im AWS CloudFormation -Benutzerhandbuch.

Themen


- [Aktivieren einer Drittanbieter-Ressource in Ihrem Konto und Ihrer Region](#)
- [Hinzufügen einer Ressource aus der AWS CloudFormation öffentlichen Registrierung zu Ihrer CDK-App](#)

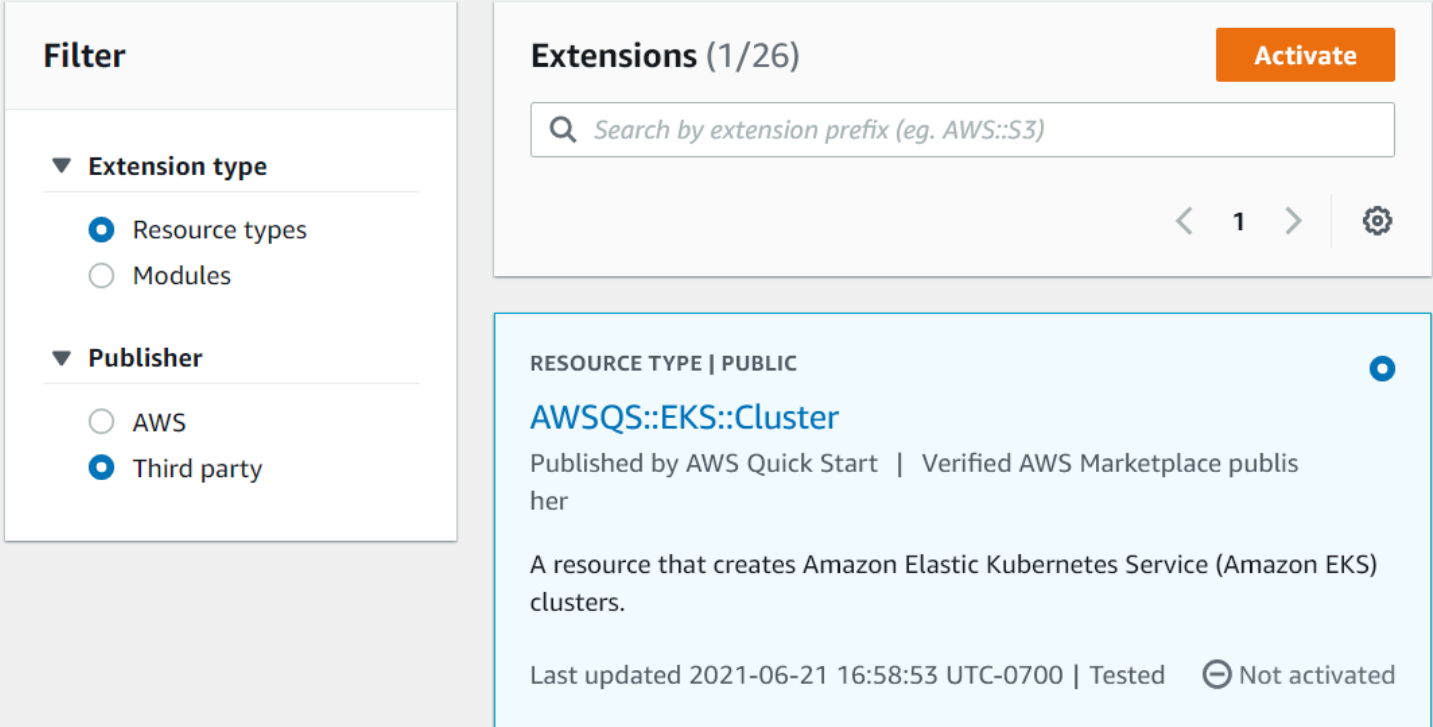
Aktivieren einer Drittanbieter-Ressource in Ihrem Konto und Ihrer Region

Erweiterungen, die von veröffentlicht AWS werden, erfordern keine Aktivierung. Sie sind immer in jedem Konto und jeder Region verfügbar. Sie können eine Erweiterung eines Drittanbieters über die AWS Management Console, die AWS Command Line Interface oder durch die Bereitstellung einer speziellen AWS CloudFormation Ressource aktivieren.

So aktivieren Sie eine Erweiterung eines Drittanbieters über die AWS Management Console oder sehen, welche Ressourcen verfügbar sind

Registry: Public extensions

The CloudFormation registry lets you manage the extensions that are available for use in your CloudFormation account. Public extensions are those publicly published in the registry for use by all CloudFormation users. This includes all extensions published by Amazon, as well as third-party extension publishers. Third-party public extensions must first be activated before they can be used in your account. [Learn more](#) 



Filter

▼ **Extension type**

- Resource types
- Modules

▼ **Publisher**

- AWS
- Third party

Extensions (1/26) **Activate**

🔍 Search by extension prefix (eg. AWS::S3)

< 1 > ⚙️

RESOURCE TYPE | PUBLIC

AWSQS::EKS::Cluster

Published by AWS Quick Start | Verified AWS Marketplace publisher

A resource that creates Amazon Elastic Kubernetes Service (Amazon EKS) clusters.

Last updated 2021-06-21 16:58:53 UTC-0700 | Tested ⊖ Not activated

1. Melden Sie sich bei dem AWS Konto an, in dem Sie die Erweiterung verwenden möchten, und wechseln Sie dann zu der Region, in der Sie sie verwenden möchten.
2. Navigieren Sie über das Menü Services zur - CloudFormation Konsole.
3. Wählen Sie in der Navigationsleiste Öffentliche Erweiterungen und aktivieren Sie dann das Optionsfeld Drittanbieter unter Publisher . Eine Liste der verfügbaren öffentlichen Erweiterungen von Drittanbietern wird angezeigt. (Sie können auch eine Liste der von veröffentlichten öffentlichen Erweiterungen AWS anzeigen AWS, müssen sie jedoch nicht aktivieren.)
4. Durchsuchen Sie die Liste und suchen Sie die Erweiterung, die Sie aktivieren möchten. Alternativ können Sie danach suchen und dann das Optionsfeld in der oberen rechten Ecke der Karte der Erweiterung aktivieren.
5. Wählen Sie oben in der Liste die Schaltfläche Aktivieren, um die ausgewählte Erweiterung zu aktivieren. Die Seite Aktivieren der Erweiterung wird angezeigt.

6. Auf der Seite Aktivieren können Sie den Standardnamen der Erweiterung überschreiben und eine Ausführungsrolle und Protokollierungskonfiguration angeben. Sie können auch auswählen, ob die Erweiterung automatisch aktualisiert werden soll, wenn eine neue Version veröffentlicht wird. Wenn Sie diese Optionen nach Ihren Wünschen festgelegt haben, wählen Sie unten auf der Seite die Option Erweiterung aktivieren aus.

So aktivieren Sie eine Erweiterung eines Drittanbieters mithilfe der AWS CLI

- Verwenden Sie den `activate-type`-Befehl. Ersetzen Sie den ARN des benutzerdefinierten Typs, den Sie verwenden möchten, sofern angegeben.

Im Folgenden wird ein Beispiel gezeigt:

```
aws cloudformation activate-type --public-type-arn public_extension_ARN --auto-update-activated
```

So aktivieren Sie eine Erweiterung eines Drittanbieters über CloudFormation oder CDK

- Stellen Sie eine Ressource vom Typ `AWS::CloudFormation::TypeActivation` und geben Sie die folgenden Eigenschaften an:
 - a. `TypeName` – Der Name des Typs, z. B. `AWSQS::EKS::Cluster`.
 - b. `MajorVersion` – Die Hauptversionsnummer der gewünschten Erweiterung. Lassen Sie weg, wenn Sie die neueste Version wünschen.
 - c. `AutoUpdate` – Gibt an, ob diese Erweiterung automatisch aktualisiert werden soll, wenn eine neue Nebenversion vom Herausgeber veröffentlicht wird. (Hauptversionsaktualisierungen erfordern eine explizite Änderung der `-MajorVersionEigenschaft`.)
 - d. `ExecutionRoleArn` – Der ARN der IAM-Rolle, unter der diese Erweiterung ausgeführt wird.
 - e. `LoggingConfig` – Die Protokollierungskonfiguration für die Erweiterung.

Die `TypeActivation` Ressource kann vom CDK mithilfe des [CfnResource](#) Konstrukts bereitgestellt werden. Dies wird für die tatsächlichen Erweiterungen im folgenden Abschnitt angezeigt.

Hinzufügen einer Ressource aus der AWS CloudFormation öffentlichen Registrierung zu Ihrer CDK-App

Verwenden Sie das [CfnResource](#)-Konstrukt, um eine Ressource aus der AWS CloudFormation öffentlichen Registrierung in Ihre Anwendung aufzunehmen. Dieses Konstrukt befindet sich im `aws-cdk-lib` Modul des CDK.

Angenommen, es gibt eine öffentliche Ressource mit dem Namen `MY::S5::UltimateBucket`, die Sie in Ihrer AWS CDK Anwendung verwenden möchten. Diese Ressource hat eine Eigenschaft: den Bucket-Namen. Die entsprechende `CfnResource`-Instanziierung sieht wie folgt aus.

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"
    }
});
```

Bereitstellen von AWS CDK Anwendungen

Stellen Sie AWS Cloud Development Kit (AWS CDK) Anwendungen bereit.

Themen

- [AWS CDK Richtlinienvvalidierung zur Synthetisierungszeit](#)
- [Kontinuierliche Integration und Bereitstellung \(CI/CD\) mithilfe von CDK Pipelines](#)

AWS CDK Richtlinienvvalidierung zur Synthetisierungszeit

Themen

- [Richtlinienvvalidierung zur Synthetisierungszeit](#)
- [Für Anwendungsentwickler](#)
- [Für Plugin-Autoren](#)

Richtlinienvvalidierung zur Synthetisierungszeit

Wenn Sie oder Ihre Organisation ein Richtlinienvvalidierungstool wie [AWS CloudFormation Guard](#) oder [OPA](#) verwenden, um Einschränkungen für Ihre AWS CloudFormation Vorlage zu definieren, können Sie diese AWS CDK zur Generierungszeit integrieren. Durch die Verwendung des entsprechenden Richtlinienvvalidierungs-Plugins können Sie die AWS CDK Anwendung veranlassen, die generierte AWS CloudFormation Vorlage unmittelbar nach der Generierung anhand Ihrer Richtlinien zu überprüfen. Wenn Verstöße vorliegen, schlägt die Synthetisierung fehl und ein Bericht wird in die Konsole gedruckt.

Die Validierung, die von AWS CDK zur Generierungszeit durchgeführt wird, validiert die Kontrollen an einem Punkt im Bereitstellungszyklus, kann sich jedoch nicht auf Aktionen auswirken, die außerhalb der Generierung auftreten. Beispiele hierfür sind Aktionen, die direkt in der Konsole oder über Service-APIs durchgeführt werden. Sie sind nicht widerstandsfähig gegenüber Änderungen von AWS CloudFormation Vorlagen nach der Synthetisierung. Ein anderer Mechanismus zur Validierung desselben Regelsatzes sollte unabhängig voneinander eingerichtet werden, z. B. [AWS CloudFormation Hooks](#) oder [AWS Config](#). Die Fähigkeit des , den Regelsatz während der Entwicklung AWS CDK auszuwerten, ist jedoch weiterhin nützlich, da dies die Erkennungsgeschwindigkeit und die Produktivität der Entwickler verbessert.

Ziel der AWS CDK Richtlinienvalidierung ist es, den Einrichtungsaufwand für die Entwicklung zu minimieren und ihn so einfach wie möglich zu machen.

Note

Diese Funktion gilt als experimentell, und sowohl die Plugin-API als auch das Format des Validierungsberichts können sich in Zukunft ändern.

Themen

- [Für Anwendungsentwickler](#)
- [Für Plugin-Autoren](#)

Für Anwendungsentwickler

Um ein oder mehrere Validierungs-Plugins in Ihrer Anwendung zu verwenden, verwenden Sie die `-policyValidationBeta1`Eigenschaft von `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: [...],
});
```

Unmittelbar nach der Synthetisierung werden alle auf diese Weise registrierten Plugins aufgerufen, um alle Vorlagen zu validieren, die in dem von Ihnen definierten Bereich generiert wurden. Insbesondere wenn Sie die Vorlagen im `App` Objekt registrieren, werden alle Vorlagen validiert.

Warning

Plugins können nicht nur die Cloud-Baugruppe ändern, sondern alles tun, was Ihre AWS CDK Anwendung kann. Sie können Daten aus dem Dateisystem lesen, auf das Netzwerk zugreifen

usw. Es liegt in Ihrer Verantwortung als Konsument eines Plugins zu überprüfen, ob es sicher ist.

AWS CloudFormation Guard -Plugin

Mit dem [CfnGuardValidator](#) Plugin können Sie verwenden, [AWS CloudFormation Guard](#) um Richtlinienvalidierungen durchzuführen. Das CfnGuardValidator Plugin verfügt über eine Reihe von integrierten [AWS Control Tower proaktiven Kontrollen](#). Der aktuelle Regelsatz finden Sie in der [Projektdokumentation](#). Wie in erwähnt [Richtlinienvalidierung zur Synthetisierungszeit](#), empfehlen wir Organisationen, eine autoritativere Validierungsmethode mit [AWS CloudFormation Hooks](#) einzurichten.

Für [AWS Control Tower](#) Kunden können dieselben proaktiven Kontrollen in Ihrer gesamten Organisation eingesetzt werden. Wenn Sie AWS Control Tower proaktive Kontrollen in Ihrer AWS Control Tower Umgebung aktivieren, können diese die Bereitstellung nicht konformer Ressourcen, die über bereitgestellt werden, stoppen AWS CloudFormation. Weitere Informationen zu verwalteten proaktiven Kontrollen und deren Funktionsweise finden Sie in der [AWS Control Tower - Dokumentation](#).

Diese AWS CDK gebündelten Kontrollen und verwalteten AWS Control Tower proaktiven Kontrollen werden am besten zusammen verwendet. In diesem Szenario können Sie dieses Validierungs-Plugin mit denselben proaktiven Kontrollen konfigurieren, die in Ihrer AWS Control Tower Cloud-Umgebung aktiv sind. Sie können sich dann schnell darauf verlassen, dass Ihre AWS CDK Anwendung die AWS Control Tower Kontrollen durch `cdk synth` lokales Ausführen übergibt.

Validierungsbericht

Wenn Sie die AWS CDK App synthetisieren, werden die Validator-Plugins aufgerufen und die Ergebnisse werden gedruckt. Ein Beispielbericht wird unten angezeigt.

```
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

Standardmäßig wird der Bericht in einem für Menschen lesbaren Format gedruckt. Wenn Sie einen Bericht im JSON-Format erstellen möchten, aktivieren Sie ihn über `@aws-cdk/core:validationReportJson` die CLI oder übergeben Sie ihn direkt an die Anwendung:

```
const app = new App({
  context: { '@aws-cdk/core:validationReportJson': true },
});
```

Alternativ können Sie dieses Kontext-Schlüssel-Wert-Paar mithilfe der `cdk.context.json` Dateien `cdk.json` oder in Ihrem Projektverzeichnis festlegen (siehe [Laufzeitkontext](#)).

Wenn Sie das JSON-Format wählen, AWS CDK druckt die den Richtlinienvalidierungsbericht in einer Datei namens `policy-validation-report.json` im Cloud-Assembly-Verzeichnis. Für das lesbare Standardformat wird der Bericht in die Standardausgabe gedruckt.

Für Plugin-Autoren

Plug-ins

Das AWS CDK Kern-Framework ist für die Registrierung und den Aufruf von Plugins und die anschließende Anzeige des formatierten Validierungsberichts verantwortlich. Die Verantwortung des Plugins besteht darin, als Übersetzungsebene zwischen dem AWS CDK Framework und dem Richtlinienvalidierungstool zu fungieren. Ein Plugin kann in jeder von unterstützten Sprache erstellt werden AWS CDK. Wenn Sie ein Plugin erstellen, das möglicherweise von mehreren Sprachen verwendet wird, wird empfohlen, das Plugin in zu erstellen, TypeScript damit Sie JSII verwenden können, um das Plugin in jeder AWS CDK Sprache zu veröffentlichen.

Erstellen von Plugins

Das Kommunikationsprotokoll zwischen dem AWS CDK Kernmodul und Ihrem Richtlinientool wird durch die `IPolicyValidationPluginBeta1` Schnittstelle definiert. Um ein neues Plugin zu erstellen, müssen Sie eine Klasse schreiben, die diese Schnittstelle implementiert. Es gibt zwei Dinge, die Sie implementieren müssen: den Plugin-Namen (durch Überschreiben der `name` Eigenschaft) und die `validate()` Methode.

Das Framework ruft auf `validate()` und übergibt ein `IValidationContextBeta1` Objekt. Der Speicherort der zu validierenden Vorlagen wird von `templatePaths` angegeben. Das Plugin sollte eine Instance von `returnValidationPluginReportBeta1`. Dieses Objekt stellt den Bericht dar, den der Benutzer am Ende der Generierung erhält.

```
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',
    }],
  }],
};
}
```

Beachten Sie, dass Plugins nichts an der Cloud-Baugruppe ändern dürfen. Jeder Versuch, dies zu tun, führt zu einem Generierungsfehler.

Wenn Ihr Plugin von einem externen Tool abhängt, denken Sie daran, dass einige Entwickler dieses Tool möglicherweise noch nicht auf ihren Workstations installiert haben. Um die Reibung zu minimieren, empfehlen wir dringend, dass Sie ein Installationskript zusammen mit Ihrem Plugin-Paket bereitstellen, um den gesamten Prozess zu automatisieren. Führen Sie dieses Skript noch besser als Teil der Installation Ihres Pakets aus. Mit können Sie es npmbeispielsweise dem `postinstall` [Skript](#) in der `-package.json` Datei hinzufügen.

Umgang mit Ausnahmen

Wenn Ihre Organisation über einen Mechanismus für den Umgang mit Ausnahmen verfügt, kann dieser als Teil des Validator-Plugins implementiert werden.

Ein Beispielszenario zur Veranschaulichung eines möglichen Befreiungsmechanismus:

- Eine Organisation hat die Regel, dass öffentliche Amazon S3-Buckets nicht zulässig sind, mit Ausnahme von in bestimmten Szenarien.
- Ein Entwickler erstellt einen Amazon S3-Bucket, der unter eines dieser Szenarien fällt, und fordert eine Befreiung an (z. B. ein Ticket erstellen).
- Sicherheitstools wissen, wie aus dem internen System gelesen wird, das Ausnahmen registriert

In diesem Szenario würde der Entwickler eine Ausnahme im internen System anfordern und dann eine Möglichkeit benötigen, diese Ausnahme zu „registrieren“. Wenn Sie zum Beispiel des Guard-Plugins hinzufügen, können Sie ein Plugin erstellen, das Ausnahmen behandelt, indem Sie die Verstöße herausfiltern, für die eine entsprechende Befreiung in einem internen Ticketing-System gilt.

Beispiele für Implementierungen finden Sie in den vorhandenen Plugins.

- [@cdklabs/cdk-validator-cfnguard](#)

Kontinuierliche Integration und Bereitstellung (CI/CD) mithilfe von CDK Pipelines

Verwenden Sie das Modul [CDK Pipelines](#) aus der AWS Construct Library, um die kontinuierliche Bereitstellung von AWS CDK Anwendungen zu konfigurieren. Wenn Sie den Quellcode Ihrer CDK-App in AWS CodeCommit, oder AWS CodeStarübertragenGitHub, kann CDK Pipelines Ihre neue Version automatisch erstellen, testen und bereitstellen.

CDK Pipelines aktualisieren sich selbst. Wenn Sie Anwendungsstufen oder Stacks hinzufügen, konfiguriert sich die Pipeline automatisch neu, um diese neuen Stufen oder Stacks bereitzustellen.

Note

CDK Pipelines unterstützt zwei APIs. Eine davon ist die ursprüngliche API, die in der CDK Pipelines Developer Preview zur Verfügung gestellt wurde. Die andere ist eine moderne API, die das Feedback von CDK-Kunden aus der Vorschauphase berücksichtigt. Die Beispiele in diesem Thema verwenden die moderne API. Einzelheiten zu den Unterschieden zwischen den beiden unterstützten APIs finden Sie unter [Original-API von CDK Pipelines](#) im GitHubaws-cdk-Repository.

Themen

- [Bootstrap für Ihre Umgebungen AWS](#)
- [Initialisieren Sie ein Projekt](#)
- [Definieren Sie eine Pipeline](#)
- [Phasen der Bewerbung](#)
- [Bereitstellungen testen](#)
- [Sicherheitshinweise](#)
- [Fehlerbehebung](#)

Bootstrap für Ihre Umgebungen AWS

Bevor Sie CDK Pipelines verwenden können, müssen Sie die AWS [Umgebung](#), in der Sie Ihre Stacks bereitstellen, booten.

Eine CDK-Pipeline umfasst mindestens zwei Umgebungen. In der ersten Umgebung wird die Pipeline bereitgestellt. In der zweiten Umgebung möchten Sie die Stacks oder Stages der Anwendung bereitstellen (Stages sind Gruppen verwandter Stacks). Diese Umgebungen können identisch sein, eine bewährte Methode besteht jedoch darin, die Stufen in verschiedenen Umgebungen voneinander zu isolieren.

Note

[the section called “Bootstrapping”](#) Weitere Informationen zu den Arten von Ressourcen, die durch Bootstrapping erstellt werden, und zur Anpassung des Bootstrap-Stacks finden Sie unter.

Für die kontinuierliche Bereitstellung mit CDK Pipelines muss Folgendes im CDK Toolkit-Stack enthalten sein:

- Ein Amazon Simple Storage Service (Amazon S3) -Bucket.
- Ein Amazon ECR-Repository.
- IAM-Rollen, um den verschiedenen Teilen einer Pipeline die erforderlichen Berechtigungen zu erteilen.

Das CDK Toolkit aktualisiert Ihren vorhandenen Bootstrap-Stack oder erstellt bei Bedarf einen neuen.

Um eine Umgebung zu booten, die eine AWS CDK Pipeline bereitstellen kann, rufen Sie `cdk bootstrap` wie im folgenden Beispiel gezeigt auf. Wenn Sie das AWS CDK Toolkit über den `npm` Befehl aufrufen, wird es bei Bedarf vorübergehend installiert. Es wird auch die Version des Toolkits verwendet, die im aktuellen Projekt installiert ist, falls eine existiert.

`--cloudformation-execution-policies` gibt den ARN einer Richtlinie an, unter der future CDK-Pipeline-Bereitstellungen ausgeführt werden. Die `AdministratorAccess` Standardrichtlinie stellt sicher, dass Ihre Pipeline jeden Ressourcentyp bereitstellen kann. AWS Wenn Sie diese Richtlinie verwenden, stellen Sie sicher, dass Sie dem gesamten Code und den Abhängigkeiten vertrauen, aus denen Ihre AWS CDK App besteht.

Die meisten Organisationen schreiben strengere Kontrollen darüber vor, welche Ressourcen automatisiert eingesetzt werden können. Erkundigen Sie sich bei der zuständigen Abteilung Ihres Unternehmens nach den Richtlinien, die für Ihre Pipeline gelten sollen.

Sie können die `--profile` Option weglassen, wenn Ihr AWS Standardprofil die erforderliche Authentifizierungskonfiguration und AWS-Region enthält.

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
```

Verwenden Sie stattdessen die folgenden Befehle, um zusätzliche Umgebungen zu starten, in denen AWS CDK Anwendungen über die Pipeline bereitgestellt werden. Die `--trust` Option gibt an, welches andere Konto über Berechtigungen zum Bereitstellen von AWS CDK Anwendungen in dieser Umgebung verfügen soll. Geben Sie für diese Option die AWS Konto-ID der Pipeline an.

Auch hier können Sie die `--profile` Option weglassen, wenn Ihr AWS Standardprofil die erforderliche Authentifizierungskonfiguration und AWS-Region enthält.

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
```

i Tip

Verwenden Sie Administratoranmeldedaten nur für das Bootstrap und die Bereitstellung der ersten Pipeline. Verwenden Sie anschließend die Pipeline selbst, nicht Ihren lokalen Computer, um Änderungen bereitzustellen.

Wenn Sie eine ältere Bootstrap-Umgebung aktualisieren, ist der vorherige Amazon S3 S3-Bucket verwaist, wenn der neue Bucket erstellt wird. Löschen Sie es manuell mithilfe der Amazon S3 S3-Konsole.

Schützen Sie Ihren Bootstrap-Stack vor dem Löschen

Wenn ein Bootstrap-Stack gelöscht wird, werden die AWS Ressourcen, die ursprünglich in der Umgebung zur Unterstützung von CDK-Bereitstellungen bereitgestellt wurden, ebenfalls gelöscht. Dies führt dazu, dass die Pipeline nicht mehr funktioniert. In diesem Fall gibt es keine allgemeine Lösung für die Wiederherstellung.

Nachdem Ihre Umgebung gebootet wurde, sollten Sie den Bootstrap-Stack der Umgebung nicht löschen und neu erstellen. Versuchen Sie stattdessen, den Bootstrap-Stack auf eine neue Version zu aktualisieren, indem Sie den Befehl erneut ausführen. `cdk bootstrap`

Zum Schutz vor versehentlichem Löschen Ihres Bootstrap-Stacks empfehlen wir Ihnen, die `--termination-protection` Option mit dem `cdk bootstrap` Befehl zum Aktivieren des Kündigungsschutzes bereitzustellen. Sie können den Kündigungsschutz für neue oder bestehende Bootstrap-Stacks aktivieren. Weitere Informationen zu dieser Option finden Sie unter [--termination-protection](#)

Nachdem Sie den Kündigungsschutz aktiviert haben, können Sie die AWS CLI CloudFormation OR-Konsole zur Überprüfung verwenden.

Um den Kündigungsschutz zu aktivieren

1. Führen Sie den folgenden Befehl aus, um den Kündigungsschutz für einen neuen oder vorhandenen Bootstrap-Stack zu aktivieren:

```
$ cdk bootstrap --termination-protection
```

2. Verwenden Sie die CloudFormation Konsole AWS CLI oder, um dies zu überprüfen. Im Folgenden finden Sie ein Beispiel unter Verwendung der AWS CLI. Wenn Sie Ihren Bootstrap-Stack-Namen geändert haben, `CDKToolkit` ersetzen Sie ihn durch Ihren Stack-Namen:

```
$ aws cloudformation describe-stacks --stack-name CDKToolkit --query  
  "Stacks[0].EnableTerminationProtection"  
true
```

Initialisieren Sie ein Projekt

Erstellen Sie ein neues, leeres GitHub Projekt und klonen Sie es auf Ihre Workstation im `my-pipeline` Verzeichnis. (Unsere Codebeispiele in diesem Thema verwenden GitHub. Sie können auch AWS CodeStar oder verwenden AWS CodeCommit.)

```
git clone GITHUB-CLONE-URL my-pipeline  
cd my-pipeline
```

Note

Sie können einen anderen Namen als den Namen `my-pipeline` für das Hauptverzeichnis Ihrer App verwenden. Wenn Sie dies jedoch tun, müssen Sie die Datei- und Klassennamen später in diesem Thema anpassen. Das liegt daran, dass das AWS CDK Toolkit einige Datei- und Klassennamen auf dem Namen des Hauptverzeichnisses basiert.

Initialisieren Sie das Projekt nach dem Klonen wie gewohnt.

TypeScript

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

JavaScript

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

Python

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

Nachdem die App erstellt wurde, geben Sie auch die folgenden beiden Befehle ein. Diese aktivieren die virtuelle Python-Umgebung der App und installieren die AWS CDK Kernabhängigkeiten.

```
$ source .venv/bin/activate # On Windows, run `.\venv\Scripts\activate` instead
$ python -m pip install -r requirements.txt
```

Java

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

Wenn Sie eine IDE verwenden, können Sie das Projekt jetzt öffnen oder importieren. Wählen Sie in Eclipse beispielsweise „Datei“ > „Importieren“ > „Maven“ > „Bestehende Maven-Projekte“. Stellen Sie sicher, dass die Projekteinstellungen so eingestellt sind, dass sie Java 8 (1.8) verwenden.

C#

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

Wenn Sie Visual Studio verwenden, öffnen Sie die Lösungsdatei im `src` Verzeichnis.

Go

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

Geben Sie nach der Erstellung der App auch den folgenden Befehl ein, um die AWS Construct Library-Module zu installieren, die die App benötigt.

```
$ go get
```

Important

Stellen Sie sicher, dass Sie Ihre `cdk.json` `cdk.context.json` Dateien der Quellcodeverwaltung übergeben. Die Kontextinformationen (wie Feature-Flags und zwischengespeicherte Werte, die aus Ihrem AWS Konto abgerufen wurden) sind Teil des Status Ihres Projekts. Die Werte können in einer anderen Umgebung unterschiedlich sein,

was zu unerwarteten Änderungen Ihrer Ergebnisse führen kann. Weitere Informationen finden Sie unter [the section called “Kontext”](#).

Definieren Sie eine Pipeline

Ihre CDK Pipelines Pipelines-Anwendung wird mindestens zwei Stacks enthalten: einen, der die Pipeline selbst darstellt, und einen oder mehrere Stacks, die die über sie bereitgestellte Anwendung darstellen. Stacks können auch in Stufen gruppiert werden, mit denen Sie Kopien von Infrastruktur-Stacks in verschiedenen Umgebungen bereitstellen können. Zunächst werden wir uns mit der Pipeline befassen und uns später mit der Anwendung befassen, die sie bereitstellen wird.

Das Konstrukt [CodePipeline](#) ist das Konstrukt, das eine CDK-Pipeline darstellt, die AWS CodePipeline als Deployment Engine verwendet wird. Wenn Sie `CodePipeline` in einem Stack instanziiieren, definieren Sie den Quellspeicherort für die Pipeline (z. B. ein GitHub Repository). Sie definieren auch die Befehle zum Erstellen der App.

Im Folgenden wird beispielsweise eine Pipeline definiert, deren Quelle in einem GitHub Repository gespeichert ist. Sie beinhaltet auch einen Build-Schritt für eine TypeScript CDK-Anwendung. Geben Sie die Informationen zu Ihrem GitHub Repo an den angegebenen Stellen ein.

Note

Standardmäßig authentifiziert sich die Pipeline GitHub mit einem persönlichen Zugriffstoken, das in Secrets Manager unter dem Namen `github-token` gespeichert ist.

Sie müssen auch die Instanziierung des Pipeline-Stacks aktualisieren, um das AWS Konto und die Region anzugeben.

TypeScript

In `lib/my-pipeline-stack.ts` (kann variieren, wenn Ihr Projektordner nicht benannt `my-pipeline` ist):

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { CodePipeline, CodePipelineSource, ShellStep } from 'aws-cdk-lib/pipelines';

export class MyPipelineStack extends cdk.Stack {
```



```

constructor(scope: Construct, id: string, props?: cdk.StackProps) {
  super(scope, id, props);

  const pipeline = new CodePipeline(this, 'Pipeline', {
    pipelineName: 'MyPipeline',
    synth: new ShellStep('Synth', {
      input: CodePipelineSource.gitHub('OWNER/REPO', 'main'),
      commands: ['npm ci', 'npm run build', 'npx cdk synth']
    })
  });
}
}
}

```

In `bin/my-pipeline.ts` (kann variieren, wenn Ihr Projektordner nicht benannt ist `my-pipeline`):

```

#!/usr/bin/env node
import * as cdk from 'aws-cdk-lib';
import { MyPipelineStack } from '../lib/my-pipeline-stack';

const app = new cdk.App();
new MyPipelineStack(app, 'MyPipelineStack', {
  env: {
    account: '111111111111',
    region: 'eu-west-1',
  }
});

app.synth();

```

JavaScript

In `lib/my-pipeline-stack.js` (kann variieren, wenn Ihr Projektordner nicht benannt ist `my-pipeline`):

```

const cdk = require('aws-cdk-lib');
const { CodePipeline, CodePipelineSource, ShellStep } = require('aws-cdk-lib/pipelines');

class MyPipelineStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);
  }
}

```

```

const pipeline = new CodePipeline(this, 'Pipeline', {
  pipelineName: 'MyPipeline',
  synth: new ShellStep('Synth', {
    input: CodePipelineSource.gitHub('OWNER/REPO', 'main'),
    commands: ['npm ci', 'npm run build', 'npx cdk synth']
  })
});
}
}

module.exports = { MyPipelineStack }

```

In `bin/my-pipeline.js` (kann variieren, wenn Ihr Projektordner nicht benannt ist `my-pipeline`):

```

#!/usr/bin/env node

const cdk = require('aws-cdk-lib');
const { MyPipelineStack } = require('../lib/my-pipeline-stack');

const app = new cdk.App();
new MyPipelineStack(app, 'MyPipelineStack', {
  env: {
    account: '111111111111',
    region: 'eu-west-1',
  }
});

app.synth();

```

Python

In `my-pipeline/my-pipeline-stack.py` (kann variieren, wenn Ihr Projektordner nicht benannt ist `my-pipeline`):

```

import aws_cdk as cdk
from constructs import Construct
from aws_cdk.pipelines import CodePipeline, CodePipelineSource, ShellStep

class MyPipelineStack(cdk.Stack):

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

```

```
pipeline = CodePipeline(self, "Pipeline",
                        pipeline_name="MyPipeline",
                        synth=ShellStep("Synth",
                                       input=CodePipelineSource.git_hub("OWNER/REPO", "main"),
                                       commands=["npm install -g aws-cdk",
                                               "python -m pip install -r requirements.txt",
                                               "cdk synth"])
                        )
```

In `app.py`:

```
#!/usr/bin/env python3
import aws_cdk as cdk
from my_pipeline.my_pipeline_stack import MyPipelineStack

app = cdk.App()
MyPipelineStack(app, "MyPipelineStack",
                env=cdk.Environment(account="111111111111", region="eu-west-1")
                )

app.synth()
```

Java

In `src/main/java/com/myorg/MyPipelineStack.java` (kann variieren, wenn Ihr Projektordner nicht benannt ist `my-pipeline`):

```
package com.myorg;

import java.util.Arrays;
import software.constructs.Construct;
import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.pipelines.CodePipeline;
import software.amazon.awscdk.pipelines.CodePipelineSource;
import software.amazon.awscdk.pipelines.ShellStep;

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

```

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

        CodePipeline pipeline = CodePipeline.Builder.create(this, "pipeline")
            .pipelineName("MyPipeline")
            .synth(ShellStep.Builder.create("Synth")
                .input(CodePipelineSource.gitHub("OWNER/REPO", "main"))
                .commands(Arrays.asList("npm install -g aws-cdk", "cdk synth"))
                .build())
            .build();
    }
}

```

In `src/main/java/com/myorg/MyPipelineApp.java` (kann variieren, wenn Ihr Projektordner nicht benannt ist `my-pipeline`):

```

package com.myorg;

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

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

        new MyPipelineStack(app, "PipelineStack", StackProps.builder()
            .env(Environment.builder()
                .account("111111111111")
                .region("eu-west-1")
                .build())
            .build());

        app.synth();
    }
}

```

C#

In `src/MyPipeline/MyPipelineStack.cs` (kann variieren, wenn Ihr Projektordner nicht benannt ist `my-pipeline`):

```

using Amazon.CDK;
using Amazon.CDK.Pipelines;

namespace MyPipeline
{
    public class MyPipelineStack : Stack
    {
        internal MyPipelineStack(Construct scope, string id, IStackProps props =
null) : base(scope, id, props)
        {
            var pipeline = new CodePipeline(this, "pipeline", new CodePipelineProps
{
                PipelineName = "MyPipeline",
                Synth = new ShellStep("Synth", new ShellStepProps
                {
                    Input = CodePipelineSource.GitHub("OWNER/REPO", "main"),
                    Commands = new string[] { "npm install -g aws-cdk", "cdk
synth" }
                })
            });
        }
    }
}

```

In `src/MyPipeline/Program.cs` (kann variieren, wenn Ihr Projektordner nicht benannt ist `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();
        }
    }
}

```

```
    }  
  }  
}
```

Go

```
package main  
  
import (  
    "github.com/aws/aws-cdk-go/awscdk/v2"  
    codebuild "github.com/aws/aws-cdk-go/awscdk/v2/awscodebuild"  
    ssm "github.com/aws/aws-cdk-go/awscdk/v2/awsssm"  
    pipeline "github.com/aws/aws-cdk-go/awscdk/v2/pipelines"  
    "github.com/aws/constructs-go/constructs/v10"  
    "github.com/aws/jsii-runtime-go"  
    "os"  
)  
  
// my CDK Stack with resources  
func NewCdkStack(scope constructs.Construct, id *string, props *awscdk.StackProps)  
    awscdk.Stack {  
    stack := awscdk.NewStack(scope, id, props)  
  
    // create an example ssm parameter  
    _ = ssm.NewStringParameter(stack, jsii.String("ssm-test-param"),  
        &ssm.StringParameterProps{  
            ParameterName: jsii.String("/testparam"),  
            Description:   jsii.String("ssm parameter for demo"),  
            StringValue:   jsii.String("my test param"),  
        })  
  
    return stack  
}  
  
// my CDK Application  
func NewCdkApplication(scope constructs.Construct, id *string, props  
    *awscdk.StageProps) awscdk.Stage {  
    stage := awscdk.NewStage(scope, id, props)  
  
    _ = NewCdkStack(stage, jsii.String("cdk-stack"), &awscdk.StackProps{Env:  
        props.Env})  
  
    return stage  
}
```

```
}

// my CDK Pipeline
func NewCdkPipeline(scope constructs.Construct, id *string, props
    *awscdk.StackProps) awscdk.Stack {
    stack := awscdk.NewStack(scope, id, props)

    // GitHub repo with owner and repository name
    githubRepo := pipeline.CodePipelineSource_GitHub(jsii.String("owner/repo"),
        jsii.String("main"), &pipeline.GitHubSourceOptions{
            Authentication: awscdk.SecretValue_SecretsManager(jsii.String("my-github-token")),
            nil,
        })

    // self mutating pipeline
    myPipeline := pipeline.NewCodePipeline(stack, jsii.String("cdkPipeline"),
        &pipeline.CodePipelineProps{
            PipelineName: jsii.String("CdkPipeline"),
            // self mutation true - pipeline changes itself before application deployment
            SelfMutation: jsii.Bool(true),
            CodeBuildDefaults: &pipeline.CodeBuildOptions{
                BuildEnvironment: &codebuild.BuildEnvironment{
                    // image version 6.0 recommended for newer go version
                    BuildImage: codebuild.LinuxBuildImage_FromCodeBuildImageId(jsii.String("aws/
codebuild/standard:6.0")),
                },
            },
            Synth: pipeline.NewCodeBuildStep(jsii.String("Synth"),
                &pipeline.CodeBuildStepProps{
                    Input: githubRepo,
                    Commands: &[*string{
                        jsii.String("npm install -g aws-cdk"),
                        jsii.String("cdk synth"),
                    },
                },
            ),
        })

    // deployment of actual CDK application
    myPipeline.AddStage(NewCdkApplication(stack, jsii.String("MyApplication"),
        &awscdk.StageProps{
            Env: targetAccountEnv(),
        }, &pipeline.AddStageOpts{
            Post: &[*pipeline.Step{
```

```
    pipeline.NewCodeBuildStep(jsii.String("Manual Steps"),
    &pipeline.CodeBuildStepProps{
        Commands: &[*string{
            jsii.String("echo \"My CDK App deployed, manual steps go here ... \"),
        },
    },
    },
    })

    return stack
}

// main app
func main() {
    defer jsii.Close()

    app := awscdk.NewApp(nil)

    // call CDK Pipeline
    NewCdkPipeline(app, jsii.String("CdkPipelineStack"), &awscdk.StackProps{
        Env: pipelineEnv(),
    })

    app.Synth(nil)
}

// env determines the AWS environment (account+region) in which our stack is to
// be deployed. For more information see: https://docs.aws.amazon.com/cdk/latest/guide/environments.html
func pipelineEnv() *awscdk.Environment {
    return &awscdk.Environment{
        Account: jsii.String(os.Getenv("CDK_DEFAULT_ACCOUNT")),
        Region: jsii.String(os.Getenv("CDK_DEFAULT_REGION")),
    }
}

func targetAccountEnv() *awscdk.Environment {
    return &awscdk.Environment{
        Account: jsii.String(os.Getenv("CDK_DEFAULT_ACCOUNT")),
        Region: jsii.String(os.Getenv("CDK_DEFAULT_REGION")),
    }
}
```


Sie müssen eine Pipeline einmal manuell bereitstellen. Danach hält sich die Pipeline aus dem Quellcode-Repository auf dem neuesten Stand. Stellen Sie also sicher, dass der Code im Repo der Code ist, den Sie bereitstellen möchten. Checken Sie Ihre Änderungen ein GitHub, klicken Sie auf und stellen Sie sie dann bereit:

```
git add --all
git commit -m "initial commit"
git push
cdk deploy
```

Tip

Nachdem Sie die erste Bereitstellung abgeschlossen haben, benötigt Ihr lokales AWS Konto keinen Administratorzugriff mehr. Dies liegt daran, dass alle Änderungen an Ihrer App über die Pipeline bereitgestellt werden. Sie müssen lediglich in der Lage sein, darauf zu drücken GitHub.

Phasen der Bewerbung

Um eine AWS Multi-Stack-Anwendung zu definieren, die der Pipeline auf einmal hinzugefügt werden kann, definieren Sie eine Unterklasse von [Stage](#) (Dies unterscheidet sich von CdkStage dem Modul CDK Pipelines.)

Die Phase enthält die Stapel, aus denen Ihre Anwendung besteht. Wenn zwischen den Stacks Abhängigkeiten bestehen, werden die Stacks automatisch in der richtigen Reihenfolge zur Pipeline hinzugefügt. Stacks, die nicht voneinander abhängen, werden parallel bereitgestellt. Sie können eine Abhängigkeitsbeziehung zwischen Stacks hinzufügen, indem Sie aufrufen. `stack1.addDependency(stack2)`

Stufen akzeptieren ein env Standardargument, das zur Standardumgebung für die darin enthaltenen Stacks wird. (Für Stacks kann immer noch eine eigene Umgebung angegeben werden.)

Eine Anwendung wird der Pipeline hinzugefügt, indem sie [addStage\(\)](#) mit Instanzen von [Stage](#) aufgerufen wird. Eine Phase kann mehrmals instanziiert und der Pipeline hinzugefügt werden, um verschiedene Phasen Ihrer DTAP- oder Multi-Region-Anwendungspipeline zu definieren.

Wir werden einen Stack erstellen, der eine einfache Lambda-Funktion enthält, und diesen Stack in einer Phase platzieren. Dann fügen wir die Phase der Pipeline hinzu, damit sie bereitgestellt werden kann.

TypeScript

Erstellen Sie die neue Datei `lib/my-pipeline-lambda-stack.ts`, die unseren Anwendungsstapel enthält, der eine Lambda-Funktion enthält.

```
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";')
    });
  }
}
```

Erstellen Sie die neue Datei `lib/my-pipeline-app-stage.ts` für unsere Bühne.

```
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');
  }
}
```

Bearbeiten `lib/my-pipeline-stack.ts`, um die Phase zu unserer Pipeline hinzuzufügen.

```

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

Erstellen Sie die neue Dateilib/my-pipeline-lambda-stack.js, die unseren Anwendungsstapel enthält, der eine Lambda-Funktion enthält.

```

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 }
```

Erstellen Sie die neue Datei `lib/my-pipeline-app-stage.js` für unsere Bühne.

```
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 };
```

Bearbeiten `lib/my-pipeline-stack.ts`, um die Phase zu unserer Pipeline hinzuzufügen.

```
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

Erstellen Sie die neue Datei `my_pipeline/my_pipeline_lambda_stack.py`, die unseren Anwendungsstapel enthält, der eine Lambda-Funktion enthält.

```
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';")
        )
```

Erstellen Sie die neue Datei `my_pipeline/my_pipeline_app_stage.py` für unsere Bühne.

```
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")
```

Bearbeiten `my_pipeline/my-pipeline-stack.py`, um die Phase zu unserer Pipeline hinzuzufügen.

```
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

Erstellen Sie die neue Datei `src/main/java/com.myorg/MyPipelineLambdaStack.java`, die unseren Anwendungsstapel enthält, der eine Lambda-Funktion enthält.

```

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();
    }
}
```

Erstellen Sie die neue Datei `src/main/java/com.myorg/MyPipelineAppStage.java` für unsere Bühne.

```
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");
    }
}
```

Bearbeitensrc/main/java/com.myorg/MyPipelineStack.java, um die Phase zu unserer Pipeline hinzuzufügen.

```
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#

Erstellen Sie die neue Datei `src/MyPipeline/MyPipelineLambdaStack.cs`, die unseren Anwendungsstapel enthält, der eine Lambda-Funktion enthält.

```

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';")
        });
    }
}
}
}

```

Erstellen Sie die neue Datei `src/MyPipeline/MyPipelineAppStage.cs` für unsere Bühne.

```

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");
        }
    }
}
}

```

Bearbeitensrc/MyPipeline/MyPipelineStack.cs, um die Phase zu unserer Pipeline hinzuzufügen.

```

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"
        }
      }));
    }
  }
}

```

Jede von hinzugefügte Anwendungsphase `addStage()` führt zum Hinzufügen einer entsprechenden Pipeline-Phase, die durch eine vom `addStage()` Aufruf zurückgegebene [StageDeployment](#) Instanz repräsentiert wird. Sie können der Phase Aktionen vor oder nach der Bereitstellung hinzufügen, indem Sie die Methode `addPre()` or `addPost()` aufrufen.

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"));
```

Sie können Stufen zu einer [Wave](#) hinzufügen, um sie parallel bereitzustellen, z. B. wenn Sie eine Phase für mehrere Konten oder Regionen bereitstellen.

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"
    }
}));

```

Bereitstellungen testen

Sie können einer CDK-Pipeline Schritte hinzufügen, um die von Ihnen durchgeführten Bereitstellungen zu validieren. Sie können beispielsweise die CDK-Pipeline-Bibliotheken verwenden [ShellStep](#), um Aufgaben wie die folgenden auszuführen:

- Es wird versucht, auf ein neu bereitgestelltes Amazon API Gateway zuzugreifen, das von einer Lambda-Funktion unterstützt wird
- Überprüfung einer Einstellung einer bereitgestellten Ressource durch Ausgabe eines Befehls AWS CLI

In seiner einfachsten Form sieht das Hinzufügen von Validierungsaktionen wie folgt aus:

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" }
}));
```

Viele AWS CloudFormation Bereitstellungen führen zur Generierung von Ressourcen mit unvorhersehbaren Namen. Aus diesem Grund bieten CDK Pipelines die Möglichkeit, AWS CloudFormation Ausgaben nach einer Bereitstellung zu lesen. Dadurch ist es möglich, (zum Beispiel) die generierte URL eines Load Balancers an eine Testaktion zu übergeben.

Um Ausgaben zu verwenden, machen Sie das `CfnOutput` Objekt verfügbar, an dem Sie interessiert sind. Übergeben Sie es dann in der `envFromCfnOutputs` Eigenschaft eines Schritts, um es als Umgebungsvariable innerhalb dieses Schritts verfügbar zu machen.

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" }
}));
```

Sie können einfache Validierungstests direkt in das schreiben `ShellStep`, aber dieser Ansatz wird unhandlich, wenn der Test mehr als ein paar Zeilen umfasst. Für komplexere Tests können Sie zusätzliche Dateien (z. B. vollständige Shell-Skripte oder Programme in anderen Sprachen) `ShellStep` über die `inputs` Eigenschaft hinzufügen. Bei den Eingaben kann es sich um jeden Schritt handeln, der eine Ausgabe hat, einschließlich einer Quelle (z. B. eines GitHub Repositorys) oder einer anderen `ShellStep`.

Das Importieren von Dateien aus dem Quell-Repository ist angemessen, wenn die Dateien direkt im Test verwendet werden können (z. B. wenn sie selbst ausführbar sind). In diesem Beispiel deklarieren wir unser GitHub Repo als `source` (anstatt es inline als Teil von zu instanziiieren). CodePipeline Dann übergeben wir diesen Dateisatz sowohl an die Pipeline als auch an den Validierungstest.

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()))
```

```

        .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" }
}));

```

Das Abrufen der zusätzlichen Dateien aus dem Synth-Schritt ist angemessen, wenn Ihre Tests kompiliert werden müssen, was im Rahmen der Synthese erfolgt.

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" }
}));
```

Sicherheitshinweise

Jede Form von Continuous Delivery birgt inhärente Sicherheitsrisiken. Im Rahmen des [Modells der AWS gemeinsamen Verantwortung](#) sind Sie für die Sicherheit Ihrer Informationen in der AWS Cloud verantwortlich. Die CDK-Pipelines-Bibliothek verschafft Ihnen einen Vorsprung, indem sie sichere Standardeinstellungen und bewährte Modellierungsmethoden integriert.

Es liegt jedoch in der Natur der Sache, dass eine Bibliothek, die ein hohes Maß an Zugriff benötigt, um ihren beabsichtigten Zweck zu erfüllen, keine vollständige Sicherheit gewährleisten kann. Es gibt viele Angriffsvektoren außerhalb AWS und Ihres Unternehmens.

Beachten Sie insbesondere Folgendes:

- Achten Sie auf die Software, auf die Sie angewiesen sind. Prüfen Sie alle Drittanbieter-Software, die Sie in Ihrer Pipeline ausführen, da sie die bereitgestellte Infrastruktur verändern kann.
- Verwenden Sie das Sperren von Abhängigkeiten, um versehentliche Upgrades zu verhindern. CDK Pipelines respektiert `package-lock.json` und stellt `yarn.lock` sicher, dass Ihre Abhängigkeiten die sind, die Sie erwarten.
- CDK Pipelines läuft auf Ressourcen, die in Ihrem eigenen Konto erstellt wurden, und die Konfiguration dieser Ressourcen wird von Entwicklern gesteuert, die Code über die Pipeline einreichen. Daher können CDK Pipelines allein nicht vor böswilligen Entwicklern schützen, die versuchen, Konformitätsprüfungen zu umgehen. Wenn Ihr Bedrohungsmodell Entwickler beinhaltet, die CDK-Code schreiben, sollten Sie externe Compliance-Mechanismen wie [AWS CloudFormation Hooks](#) (präventiv) oder [AWS Config](#) (reaktiv) einrichten, zu deren Deaktivierung die AWS CloudFormation Ausführungsrolle nicht berechtigt ist.
- Anmeldeinformationen für Produktionsumgebungen sollten kurzlebig sein. Nach dem Bootstrapping und der ersten Bereitstellung benötigen Entwickler überhaupt keine Kontoanmeldeinformationen mehr. Änderungen können über die Pipeline bereitgestellt werden. Reduzieren Sie die Wahrscheinlichkeit, dass Anmeldeinformationen durchsickern, indem Sie sie gar nicht erst benötigen.

Fehlerbehebung

Die folgenden Probleme treten häufig bei den ersten Schritten mit CDK Pipelines auf.

Pipeline: Interner Fehler

```
CREATE_FAILED | AWS::CodePipeline::Pipeline | Pipeline/Pipeline  
Internal Failure
```

Überprüfen Sie Ihr GitHub Zugriffstoken. Es fehlt möglicherweise oder verfügt nicht über die erforderlichen Berechtigungen für den Zugriff auf das Repository.

Schlüssel: Die Richtlinie enthält eine Anweisung mit einem oder mehreren ungültigen Prinzipalen

```
CREATE_FAILED | AWS::KMS::Key | Pipeline/Pipeline/ArtifactsBucketEncryptionKey  
Policy contains a statement with one or more invalid principals.
```

Eine der Zielumgebungen wurde nicht mit dem neuen Bootstrap-Stack gebootet. Stellen Sie sicher, dass alle Ihre Zielumgebungen bootstrapped sind.

Der Stack befindet sich im Status `ROLLBACK_COMPLETE` und kann nicht aktualisiert werden.

```
Stack STACK_NAME is in ROLLBACK_COMPLETE state and can not be updated. (Service:
AmazonCloudFormation; Status Code: 400; Error Code: ValidationError; Request
ID: ...)
```

Der Stack ist bei der vorherigen Bereitstellung fehlgeschlagen und befindet sich in einem Zustand, der nicht erneut versucht werden kann. Löschen Sie den Stack von der AWS CloudFormation Konsole und wiederholen Sie die Bereitstellung.

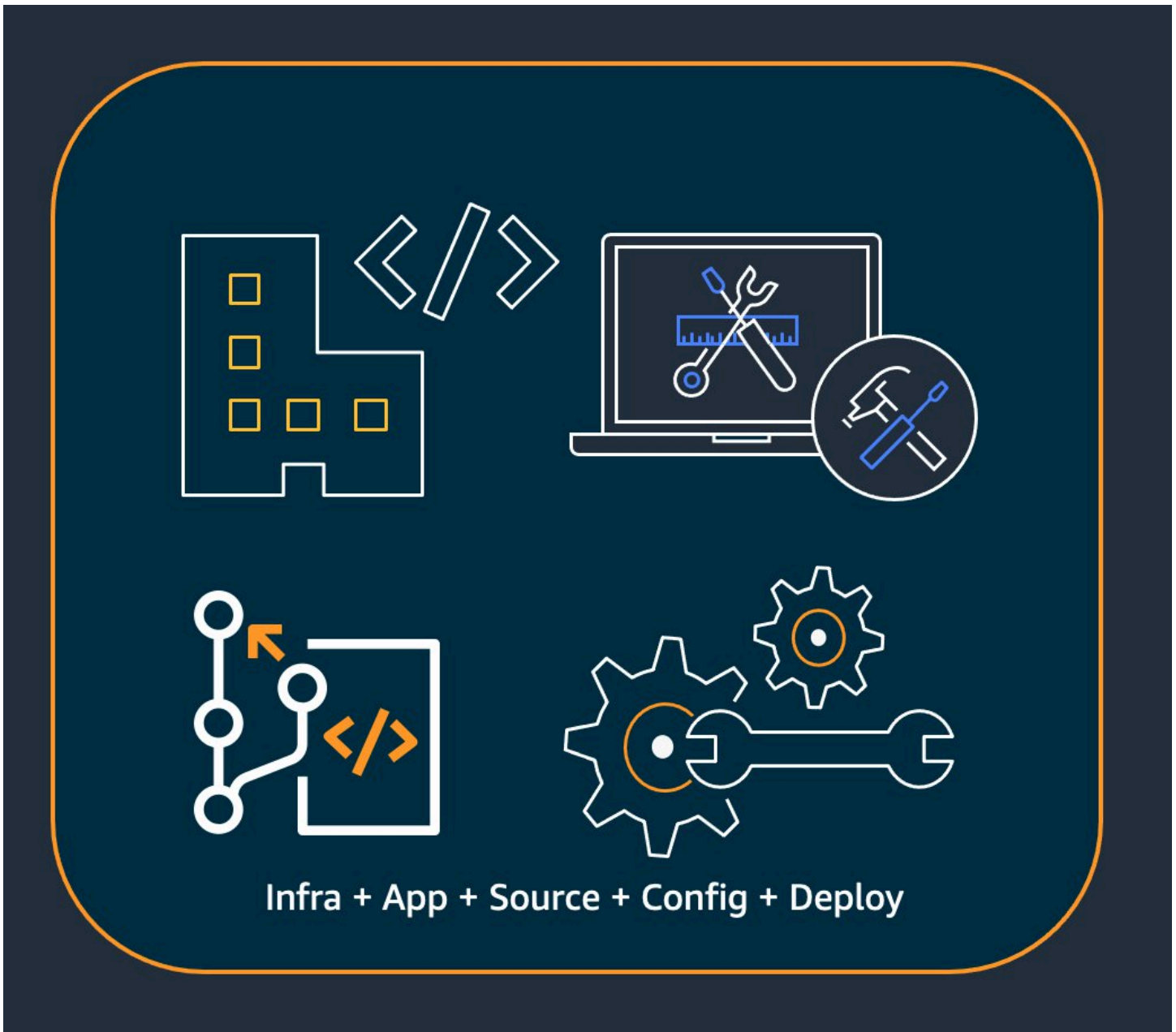
Bewährte Methoden für die Entwicklung und Bereitstellung einer Cloud-Infrastruktur mit der AWS CDK

Mit der können AWS CDK Entwickler oder Administratoren ihre Cloud-Infrastruktur mithilfe einer unterstützten Programmiersprache definieren. CDK-Anwendungen sollten in logische Einheiten wie API-, Datenbank- und Überwachungsressourcen organisiert werden und optional über eine Pipeline für automatisierte Bereitstellungen verfügen. Die logischen Einheiten sollten als Konstrukte implementiert werden, einschließlich der folgenden:

- Infrastruktur (z. B. Amazon S3-Buckets, Amazon-RDS-Datenbanken oder ein Amazon-VPC-Netzwerk)
- Laufzeitcode (z. B. AWS Lambda Funktionen)
- Konfigurationscode

Stacks definieren das Bereitstellungsmodell dieser logischen Einheiten. Eine detailliertere Einführung in die Konzepte hinter dem CDK finden Sie unter [Erste Schritte](#).

Die AWS CDK spiegelt die sorgfältige Berücksichtigung der Anforderungen unserer Kunden und internen Teams sowie der Fehlermuster wider, die sich häufig während der Bereitstellung und laufenden Wartung komplexer Cloud-Anwendungen ergeben. Wir haben festgestellt, dass Fehler häufig mit „out-of-band“-Änderungen an einer Anwendung zusammenhängen, die nicht vollständig getestet wurden, z. B. Konfigurationsänderungen. Daher haben wir die AWS CDK um ein Modell herum entwickelt, in dem Ihre gesamte Anwendung im Code definiert ist, nicht nur in der Geschäftslogik, sondern auch in der Infrastruktur und Konfiguration. Auf diese Weise können vorgeschlagene Änderungen sorgfältig überprüft, in Umgebungen, die der Produktion ähneln, in unterschiedlichem Umfang umfassend getestet und im Falle eines Fehlers vollständig rückgängig gemacht werden.



Zum Zeitpunkt der Bereitstellung AWS CDK synthetisiert die eine Cloud-Baugruppe, die Folgendes enthält:

- AWS CloudFormation -Vorlagen, die Ihre Infrastruktur in allen Zielumgebungen beschreiben
- Dateikomponenten, die Ihren Laufzeitcode und ihre unterstützenden Dateien enthalten

Mit dem CDK kann jeder Commit im Hauptversionsverwaltungszweig Ihrer Anwendung eine vollständige, konsistente, bereitstellbare Version Ihrer Anwendung darstellen. Ihre Anwendung kann dann automatisch bereitgestellt werden, wenn eine Änderung vorgenommen wird.

Der Rückstand hinter den AWS CDK führt zu unseren empfohlenen bewährten Methoden, die wir in vier große Kategorien unterteilt haben.

- [the section called “Bewährte Methoden für Organisationen”](#)
- [the section called “Bewährte Methoden für die Codierung”](#)
- [the section called “Bewährte Methoden für die Erstellung von”](#)
- [the section called “Bewährte Methoden für Anwendungen”](#)

 Tip

Berücksichtigen Sie auch [bewährte Methoden für AWS CloudFormation](#) und die einzelnen - AWS Services, die Sie verwenden, sofern zutreffend für CDK-definierte Infrastruktur.

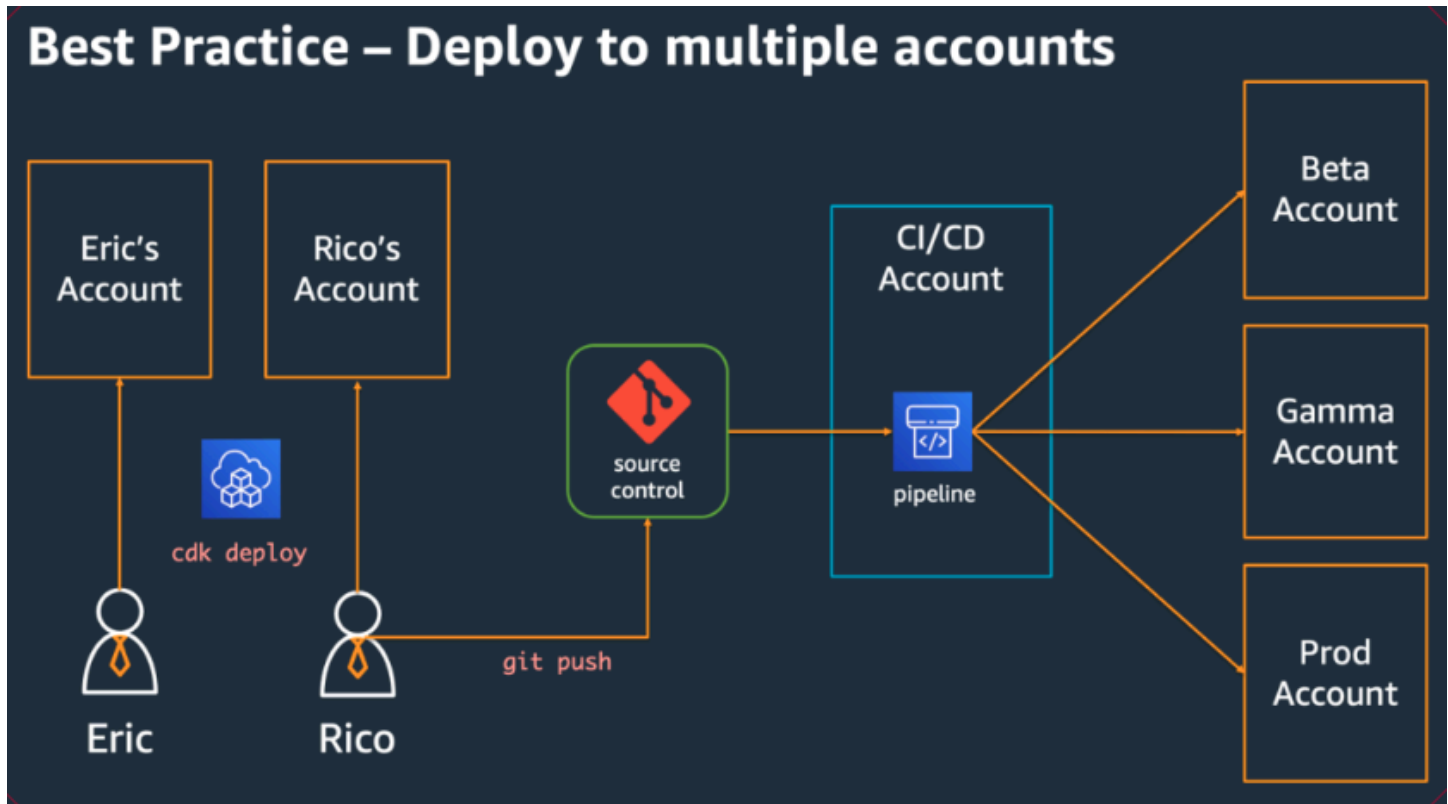
Bewährte Methoden für Organisationen

In den Anfangsphasen der AWS CDK Einführung ist es wichtig zu überlegen, wie Sie Ihre Organisation erfolgreich einrichten können. Es ist eine bewährte Methode, ein Team von Experten zu haben, das für die Schulung und Unterstützung des restlichen Unternehmens verantwortlich ist, wenn es das CDK übernimmt. Die Größe dieses Teams kann variieren, von einer oder zwei Personen in einem kleinen Unternehmen bis hin zu einem vollwertigen Cloud-Kompetenzzentrum (CCoE) in einem größeren Unternehmen. Dieses Team ist für die Festlegung von Standards und Richtlinien für die Cloud-Infrastruktur in Ihrem Unternehmen sowie für das Training und die Unterstützung von Entwicklern verantwortlich.

Das CCoE bietet möglicherweise Anleitungen dazu, welche Programmiersprachen für die Cloud-Infrastruktur verwendet werden sollten. Die Details variieren von Organisation zu Organisation, aber eine gute Richtlinie hilft sicherzustellen, dass Entwickler die Cloud-Infrastruktur des Unternehmens verstehen und verwalten können.

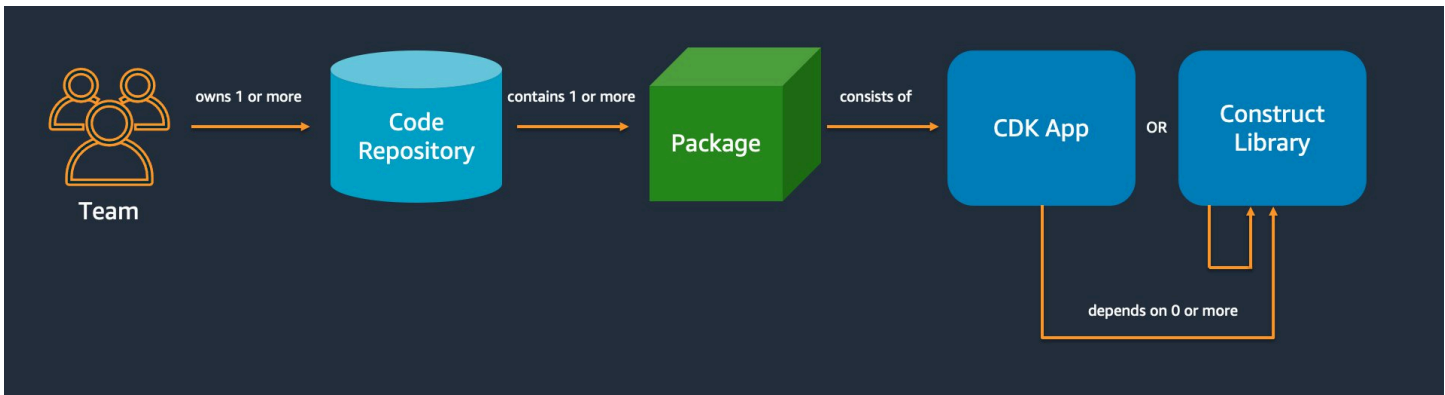
Das CCoE erstellt auch eine „Landing Zone“, die Ihre Organisationseinheiten innerhalb von definiert AWS. Eine Landing Zone ist eine vorkonfigurierte, sichere, skalierbare AWS Umgebung mit mehreren Konten, die auf bewährten Methoden basiert. Um die Services zu verknüpfen, aus denen Ihre Landing Zone besteht, können Sie verwenden [AWS Control Tower](#), das Ihr gesamtes System mit mehreren Konten von einer einzigen Benutzeroberfläche aus konfiguriert und verwaltet.

Entwicklungsteams sollten in der Lage sein, ihre eigenen Konten zum Testen und Bereitstellen neuer Ressourcen in diesen Konten nach Bedarf zu verwenden. Einzelne Entwickler können diese Ressourcen als Erweiterungen ihrer eigenen Entwicklungsarbeitsstation behandeln. Mithilfe von [CDK Pipelines](#) können die AWS CDK Anwendungen dann über ein CI/CD-Konto in Test-, Integrations- und Produktionsumgebungen (jeweils isoliert in einer eigenen AWS Region oder einem eigenen Konto) bereitgestellt werden. Dazu wird der Code der Entwickler mit dem kanonischen Repository Ihrer Organisation zusammengeführt.



Bewährte Methoden für die Codierung

In diesem Abschnitt werden bewährte Methoden für die Organisation Ihres AWS CDK Codes vorgestellt. Das folgende Diagramm zeigt die Beziehung zwischen einem Team und den Code-Repositories, Paketen, Anwendungen und Konstruktbibliotheken dieses Teams.



Starten Sie einfach und fügen Sie Komplexität nur hinzu, wenn Sie sie benötigen

Das Leitprinzip für die meisten unserer bewährten Methoden besteht darin, die Dinge so einfach wie möglich zu halten – aber nicht einfacher. Fügen Sie Komplexität nur hinzu, wenn Ihre Anforderungen eine kompliziertere Lösung vorgeben. Mit der können Sie Ihren Code nach Bedarf umgestalten AWS CDK, um neue Anforderungen zu unterstützen. Sie müssen nicht für alle möglichen Szenarien im Voraus entwerfen.

Abstimmung mit dem AWS Well-Architected Framework

Das [AWS Well-Architected](#) Framework definiert eine Komponente als Code, Konfiguration und AWS Ressourcen, die zusammen eine Anforderung erfüllen. Eine Komponente ist oft die Einheit des technischen Eigentums und ist von anderen Komponenten entkoppelt. Der Begriff Workload wird verwendet, um eine Reihe von Komponenten zu identifizieren, die zusammen einen Geschäftswert bieten. Ein Workload ist in der Regel der Detaillierungsgrad, über den Geschäfts- und Technologieleiter kommunizieren.

Eine AWS CDK Anwendung ist einer Komponente zugeordnet, wie sie vom AWS Well-Architected Framework. AWS CDK apps definiert ist, ein Mechanismus zur Kodifizierung und Bereitstellung bewährter Methoden für Well-Architected-Cloud-Anwendungen. Sie können Komponenten auch als wiederverwendbare Codebibliotheken über Artefakt-Repositorys wie erstellen und freigeben AWS CodeArtifact.

Jede Anwendung beginnt mit einem einzigen Paket in einem einzigen Repository

Ein einzelnes Paket ist der Eintrittspunkt Ihrer AWS CDK App. Hier definieren Sie, wie und wo die verschiedenen logischen Einheiten Ihrer Anwendung bereitgestellt werden sollen. Sie definieren auch die CI/CD-Pipeline, um die Anwendung bereitzustellen. Die Konstrukte der App definieren die logischen Einheiten Ihrer Lösung.

Verwenden Sie zusätzliche Pakete für Konstrukte, die Sie in mehr als einer Anwendung verwenden. (Gemeinsame Konstrukte sollten auch ihren eigenen Lebenszyklus und ihre eigene Teststrategie haben.) Abhängigkeiten zwischen Paketen im selben Repository werden von den Build-Tools Ihres Repo verwaltet.

Obwohl dies möglich ist, empfehlen wir nicht, mehrere Anwendungen in dasselbe Repository zu legen, insbesondere wenn automatisierte Bereitstellungs Pipelines verwendet werden. Dadurch wird der „Explosionsradius“ der Änderungen während der Bereitstellung erhöht. Wenn mehrere Anwendungen in einem Repository vorhanden sind, lösen Änderungen an einer Anwendung die Bereitstellung der anderen aus (auch wenn sich die anderen nicht geändert haben). Darüber hinaus verhindert eine Unterbrechung in einer Anwendung, dass die anderen Anwendungen bereitgestellt werden.

Verschieben von Code in Repositorys basierend auf dem Codelebenszyklus oder der Teameigentümerschaft

Wenn Pakete in mehreren Anwendungen verwendet werden, verschieben Sie sie in ein eigenes Repository. Auf diese Weise können die Pakete von Anwendungsentwicklungssystemen referenziert werden, die sie verwenden, und sie können unabhängig von den Anwendungslebenszyklen auch in Abständen aktualisiert werden. Es könnte jedoch zunächst sinnvoll sein, alle freigegebenen Konstrukte in einem Repository abzulegen.

Verschieben Sie Pakete außerdem in ein eigenes Repository, wenn verschiedene Teams an ihnen arbeiten. Dies trägt dazu bei, die Zugriffskontrolle durchzusetzen.

Um Pakete über Repository-Grenzen hinweg zu verwenden, benötigen Sie ein privates Paket-Repository – ähnlich wie NPM PyPi, oder Maven Central, aber intern in Ihrer Organisation. Sie benötigen auch einen Release-Prozess, der das Paket erstellt, testet und im privaten Paket-Repository veröffentlicht. [CodeArtifact](#) kann Pakete für die gängigsten Programmiersprachen hosten.

Abhängigkeiten von Paketen im Paket-Repository werden vom Paketmanager Ihrer Sprache verwaltet, z. B. NPM für - TypeScript oder - JavaScript Anwendungen. Ihr Paketmanager hilft sicherzustellen, dass Builds wiederholbar sind. Dazu werden die spezifischen Versionen jedes Pakets aufgezeichnet, von dem Ihre Anwendung abhängt. Außerdem können Sie diese Abhängigkeiten kontrolliert aktualisieren.

Freigegebene Pakete benötigen eine andere Teststrategie. Für eine einzelne Anwendung reicht es möglicherweise aus, die Anwendung in einer Testumgebung bereitzustellen und zu bestätigen, dass sie weiterhin funktioniert. Freigegebene Pakete müssen jedoch unabhängig von der verbrauchenden Anwendung getestet werden, als ob sie öffentlich veröffentlicht würden. (Ihre Organisation könnte sich dafür entscheiden, tatsächlich einige freigegebene Pakete für die Öffentlichkeit freizugeben.)

Beachten Sie, dass ein Konstrukt beliebig einfach oder komplex sein kann. Ein Bucket ist ein Konstrukt, `CameraShopWebsite` könnte aber auch ein Konstrukt sein.

Infrastruktur- und Laufzeitcode live im selben Paket

Zusätzlich zur Generierung von AWS CloudFormation Vorlagen für die Bereitstellung der Infrastruktur bündelt die AWS CDK auch Laufzeitressourcen wie Lambda-Funktionen und Docker-Images und stellt sie zusammen mit Ihrer Infrastruktur bereit. Dadurch ist es möglich, den Code, der Ihre Infrastruktur definiert, und den Code, der Ihre Laufzeitlogik implementiert, zu einem einzigen Konstrukt zu kombinieren. Dies ist eine bewährte Methode. Diese beiden Arten von Code müssen sich nicht in separaten Repositories oder sogar in separaten Paketen befinden.

Um die beiden Arten von Code gemeinsam weiterzuentwickeln, können Sie ein eigenständiges Konstrukt verwenden, das eine Funktionalität, einschließlich seiner Infrastruktur und Logik, vollständig beschreibt. Mit einem eigenständigen Konstrukt können Sie die beiden Arten von Code isoliert testen, den Code projektübergreifend freigeben und wiederverwenden und den gesamten Code synchron versionieren.

Bewährte Methoden für die Erstellung von

Dieser Abschnitt enthält bewährte Methoden für die Entwicklung von Konstrukten. Konstrukte sind wiederverwendbare, zusammensetzbare Module, die Ressourcen kapseln. Sie sind die Bausteine von AWS CDK Apps.

Modell mit Konstrukten, Bereitstellung mit Stacks

Stacks sind die Bereitstellungseinheit: alles in einem Stack wird zusammen bereitgestellt. Wenn Sie also die übergeordneten logischen Einheiten Ihrer Anwendung aus mehreren AWS Ressourcen erstellen, stellen Sie jede logische Einheit als [Construct](#), nicht als [Stack](#). Verwenden Sie Stacks nur, um zu beschreiben, wie Ihre Konstrukte für Ihre verschiedenen Bereitstellungsszenarien zusammengestellt und verbunden werden sollen.

Wenn es sich bei einer Ihrer logischen Einheiten beispielsweise um eine Website handelt, sollten die Konstrukte, aus denen sie besteht (z. B. ein Amazon S3-Bucket, API Gateway, Lambda-Funktionen oder Amazon-RDS-Tabellen), in einem einzigen übergeordneten Konstrukt zusammengesetzt werden. Dann sollte dieses Konstrukt zur Bereitstellung in einem oder mehreren Stacks instanziiert werden.

Durch die Verwendung von Konstrukten für die Erstellung und Stacks für die Bereitstellung verbessern Sie das Wiederverwendungsmöglichkeiten Ihrer Infrastruktur und geben Ihnen mehr Flexibilität bei der Bereitstellung.

Konfigurieren mit Eigenschaften und Methoden, nicht mit Umgebungsvariablen

Suchen nach Umgebungsvariablen innerhalb von Konstrukten und Stacks sind ein gängiges Anti-Muster. Sowohl Konstrukte als auch Stacks sollten ein Eigenschaftenobjekt akzeptieren, um eine vollständige Konfiguration vollständig im Code zu ermöglichen. Andernfalls führt dies zu einer Abhängigkeit von der Maschine, auf der der Code ausgeführt wird, wodurch noch mehr Konfigurationsinformationen erstellt werden, die Sie verfolgen und verwalten müssen.

Im Allgemeinen sollten Umgebungsvariablen-Lookups auf die oberste Ebene einer AWS CDK App beschränkt sein. Sie sollten auch verwendet werden, um Informationen zu übergeben, die für die Ausführung in einer Entwicklungsumgebung benötigt werden. Weitere Informationen finden Sie unter [the section called "Umgebungen"](#).

Komponententest Ihrer Infrastruktur

Um in allen Umgebungen konsistent eine vollständige Suite von Einheitentests zur Erstellungszeit durchzuführen, vermeiden Sie Netzwerk-Lookups während der Synthetisierung und modellieren Sie alle Ihre Produktionsphasen im Code. (Diese bewährten Methoden werden später behandelt.) Wenn ein einzelner Commit immer zu derselben generierten Vorlage führt, können Sie den von

Ihnen geschriebenen Einheitentests vertrauen, um zu bestätigen, dass die generierten Vorlagen wie erwartet aussehen. Weitere Informationen finden Sie unter [Konstrukte testen](#).

Ändern Sie die logische ID von zustandsbehafteten Ressourcen nicht

Das Ändern der logischen ID einer Ressource führt dazu, dass die Ressource bei der nächsten Bereitstellung durch eine neue ersetzt wird. Bei statusbehafteten Ressourcen wie Datenbanken und S3-Buckets oder einer persistenten Infrastruktur wie einer Amazon VPC ist dies selten, was Sie möchten. Seien Sie vorsichtig bei einem Faktorwechsel Ihres AWS CDK Codes, der dazu führen könnte, dass sich die ID ändert. Schreiben Sie Einheitentests, die bestätigen, dass die logischen IDs Ihrer statusbehafteten Ressourcen statisch bleiben. Die logische ID wird von der abgeleitet, die `id` Sie angeben, wenn Sie das Konstrukt instanziiieren, und von der Position des Konstrukts in der Konstruktstruktur. Weitere Informationen finden Sie unter [the section called "Logische IDs"](#).

Konstrukte reichen für Compliance nicht aus

Viele Unternehmenskunden schreiben ihre eigenen Wrapper für L2-Konstrukte (die „kuratierten“ Konstrukte, die einzelne AWS Ressourcen mit integrierten verzweigten Standardeinstellungen und bewährten Methoden darstellen). Diese Wrapper setzen bewährte Sicherheitsmethoden wie statische Verschlüsselung und spezifische IAM-Richtlinien durch. Sie können beispielsweise eine erstellen, `MyCompanyBucket` die Sie dann in Ihren Anwendungen anstelle des üblichen `AmazonS3Bucket` Konstrukts verwenden. Dieses Muster ist nützlich, um zu Beginn des Lebenszyklus der Softwareentwicklung Sicherheitsempfehlungen zu erhalten, aber verlassen Sie sich nicht darauf als einzige Möglichkeit der Durchsetzung.

Verwenden Sie stattdessen AWS Funktionen wie [Service-Kontrollrichtlinien](#) und [Berechtigungsgrenzen](#), um Ihre Sicherheitsvorkehrungen auf Organisationsebene durchzusetzen. Verwenden Sie - [the section called "Aspekte"](#) oder -Tools wie [CloudFormation Guard](#), um vor der Bereitstellung Aussagen über die Sicherheitseigenschaften von Infrastrukturelementen zu treffen. Verwenden Sie AWS CDK für das, was es am besten macht.

Beachten Sie schließlich, dass das Schreiben Ihrer eigenen „L2+“-Konstrukte Ihre Entwickler daran hindern kann, AWS CDK Pakete wie [AWS Lösungskonstrukte](#) oder Konstrukte von Drittanbietern aus dem Construct Hub zu nutzen. Diese Pakete basieren in der Regel auf AWS CDK Standardkonstrukten und können Ihre Wrapper-Konstrukte nicht verwenden.

Bewährte Methoden für Anwendungen

In diesem Abschnitt wird beschrieben, wie Sie Ihre AWS CDK Anwendungen schreiben und Konstrukte kombinieren, um zu definieren, wie Ihre AWS Ressourcen verbunden sind.

Treffen Sie Entscheidungen zur Generierungszeit

Obwohl es Ihnen AWS CloudFormation ermöglicht, Entscheidungen zur Bereitstellungszeit zu treffen (mit `Conditions`, und `Parameters`), und Ihnen einen gewissen Zugriff auf diese Mechanismen AWS CDK bietet, empfehlen wir, sie nicht zu verwenden. Die Arten von Werten, die Sie verwenden können, und die Arten von Operationen, die Sie für sie ausführen können, sind im Vergleich zu den verfügbaren Werten in einer Programmiersprache für allgemeine Zwecke begrenzt.

Versuchen Sie stattdessen, alle Entscheidungen, z. B. welches Konstrukt instanziiert werden soll, in Ihrer AWS CDK Anwendung unter Verwendung der `if` Anweisungen und anderen Funktionen Ihrer Programmiersprache zu treffen. Beispielsweise ist ein häufiger CDK-Ausdruck, der über eine Liste iteriert und ein Konstrukt mit Werten aus jedem Element in der Liste instanziiert, einfach nicht möglich, AWS CloudFormation Ausdrücke zu verwenden.

Behandeln Sie AWS CloudFormation als Implementierungsdetail, das die für robuste Cloud-Bereitstellungen AWS CDK verwendet, nicht als Sprachziel. Sie schreiben keine AWS CloudFormation Vorlagen in TypeScript oder Python, Sie schreiben CDK-Code, der zufällig CloudFormation für die Bereitstellung verwendet wird.

Verwenden Sie generierte Ressourcennamen, keine physischen Namen

Namen sind eine wertvolle Ressource. Jeder Name kann nur einmal verwendet werden. Wenn Sie also einen Tabellennamen oder Bucket-Namen in Ihrer Infrastruktur und Anwendung fest codieren, können Sie diesen Infrastrukturbereich nicht zweimal im selben Konto bereitstellen. (Der Name, über den wir hier sprechen, ist der Name, der beispielsweise durch die `bucketName` Eigenschaft auf einem Amazon S3-Bucket-Konstrukt angegeben wird.)

Schlimmer noch, Sie können keine Änderungen an der Ressource vornehmen, die eine Ersetzung erfordert. Wenn eine Eigenschaft nur bei der Ressourcenerstellung festgelegt werden kann, z. B. bei `KeySchema` einer Amazon-DynamoDB-Tabelle, ist diese Eigenschaft unveränderlich. Um diese Eigenschaft zu ändern, ist eine neue Ressource erforderlich. Die neue Ressource muss jedoch denselben Namen haben, um ein echter Ersatz zu sein. Es darf jedoch nicht denselben Namen haben, während die vorhandene Ressource diesen Namen noch verwendet.

Ein besserer Ansatz besteht darin, so wenige Namen wie möglich anzugeben. Wenn Sie Ressourcennamen weglassen, AWS CDK generiert die sie für Sie auf eine Weise, die keine Probleme verursacht. Angenommen, Sie haben eine Tabelle als Ressource. Anschließend können Sie den generierten Tabellennamen als Umgebungsvariable an Ihre AWS Lambda Funktion übergeben. In Ihrer AWS CDK Anwendung können Sie den Tabellennamen als `referenzierentable.tableName`. Alternativ können Sie beim Start eine Konfigurationsdatei auf Ihrer Amazon EC2-Instance generieren oder den tatsächlichen Tabellennamen in den AWS Systems Manager Parameter Store schreiben, damit Ihre Anwendung ihn von dort aus lesen kann.

Wenn es sich bei dem benötigten Stack um einen anderen AWS CDK Stack handelt, ist das noch einfacher. Angenommen, ein Stack definiert die Ressource und ein anderer Stack muss sie verwenden, gilt Folgendes:

- Wenn sich die beiden Stacks in derselben AWS CDK App befinden, übergeben Sie eine Referenz zwischen den beiden Stacks. Speichern Sie beispielsweise einen Verweis auf das Konstrukt der Ressource als Attribut des definierenden Stacks (`this.stack.uploadBucket = myBucket`). Übergeben Sie dann dieses Attribut an den Konstruktor des Stacks, der die Ressource benötigt.
- Wenn sich die beiden Stacks in verschiedenen AWS CDK Apps befinden, verwenden Sie eine statische `from` Methode, um eine extern definierte Ressource basierend auf ihrem ARN, Namen oder anderen Attributen zu verwenden. (Verwenden Sie beispielsweise `Table.fromArn()` für eine DynamoDB-Tabelle). Verwenden Sie das `CfnOutput` Konstrukt, um den ARN oder einen anderen erforderlichen Wert in der Ausgabe von `cdk deploy`, oder suchen Sie in der AWS Management Console. Alternativ kann die zweite App die CloudFormation von der ersten App generierte Vorlage lesen und diesen Wert aus dem `Outputs` Abschnitt abrufen.

Definieren von Entfernungsrichtlinien und Protokollaufbewahrung

Der AWS CDK versucht, Sie daran zu hindern, Daten zu verlieren, indem er standardmäßig Richtlinien verwendet, die alles enthalten, was Sie erstellen. Die standardmäßige Entfernungsrichtlinie für Ressourcen, die Daten enthalten (z. B. Amazon S3-Buckets und Datenbanktabellen), besteht beispielsweise darin, die Ressource nicht zu löschen, wenn sie aus dem Stack entfernt wird. Stattdessen ist die Ressource aus dem Stack verwaist. In ähnlicher Weise ist die Standardeinstellung des CDK, alle Protokolle dauerhaft beizubehalten. In Produktionsumgebungen können diese Standardwerte schnell zur Speicherung großer Datenmengen, die Sie nicht wirklich benötigen, und zu einer entsprechenden AWS Rechnung führen.

Überlegen Sie sorgfältig, was diese Richtlinien für jede Produktionsressource sein sollen, und geben Sie sie entsprechend an. Verwenden Sie [the section called “Aspekte”](#), um die Entfernungs- und Protokollierungsrichtlinien in Ihrem Stack zu validieren.

Trennen Sie Ihre Anwendung gemäß den Bereitstellungsanforderungen in mehrere Stacks

Es gibt keine feste und schnelle Regel für die Anzahl der Stacks, die Ihre Anwendung benötigt. In der Regel treffen Sie die Entscheidung zu Ihren Bereitstellungsmustern. Beachten Sie die folgenden Richtlinien:

- In der Regel ist es einfacher, so viele Ressourcen wie möglich im selben Stack aufzubewahren. Halten Sie sie daher zusammen, es sei denn, Sie wissen, dass Sie sie trennen möchten.
- Erwägen Sie, zustandsbehaftete Ressourcen (wie Datenbanken) in einem separaten Stack von zustandslosen Ressourcen aufzubewahren. Anschließend können Sie den Beendigungsschutz für den zustandsbehafteten Stack aktivieren. Auf diese Weise können Sie mehrere Kopien des zustandslosen Stacks frei löschen oder erstellen, ohne dass Daten verloren gehen könnten.
- Zustandsbehaftete Ressourcen sind empfindlicher bei der Umbenennung von Konstrukten – das Umbenennen führt zu einem Ressourcenaustausch. Verschachteln Sie daher keine zustandsbehafteten Ressourcen in Konstrukte, die wahrscheinlich verschoben oder umbenannt werden (es sei denn, der Status kann bei Verlust neu erstellt werden, wie ein Cache). Dies ist ein weiterer guter Grund, zustandsbehaftete Ressourcen in ihrem eigenen Stack abzulegen.

Bekennen Sie sich `cdk.context.json`, um nicht deterministisches Verhalten zu vermeiden

Determinismus ist der Schlüssel zu erfolgreichen AWS CDK Bereitstellungen. Eine AWS CDK App sollte bei jeder Bereitstellung in einer bestimmten Umgebung im Wesentlichen dasselbe Ergebnis haben.

Da Ihre AWS CDK App in einer Programmiersprache für allgemeine Zwecke geschrieben ist, kann sie beliebigen Code ausführen, beliebige Bibliotheken verwenden und beliebige Netzwerkaufrufe tätigen. Sie könnten beispielsweise ein AWS SDK verwenden, um einige Informationen aus Ihrem AWS Konto abzurufen, während Sie Ihre App synthetisieren. Beachten Sie, dass dies zu zusätzlichen Einrichtungsanforderungen für Anmeldeinformationen, einer erhöhten Latenz und einer Wahrscheinlichkeit eines Fehlers führt, wenn Sie `cdk synth` ausführen.

Ändern Sie niemals Ihr AWS Konto oder Ihre Ressourcen während der Generierung. Das Synthetisieren einer App sollte keine Nebenwirkungen haben. Änderungen an Ihrer Infrastruktur sollten erst in der Bereitstellungsphase erfolgen, nachdem die AWS CloudFormation Vorlage generiert wurde. Auf diese Weise AWS CloudFormation kann bei einem Problem die Änderung automatisch rückgängig machen. Um Änderungen vorzunehmen, die nicht einfach innerhalb des AWS CDK Frameworks vorgenommen werden können, verwenden Sie [benutzerdefinierte Ressourcen](#), um bei der Bereitstellung beliebigen Code auszuführen.

Selbst strikt schreibgeschützte Aufrufe sind nicht unbedingt sicher. Überlegen Sie, was passiert, wenn sich der von einem Netzwerkaufruf zurückgegebene Wert ändert. Welchen Teil Ihrer Infrastruktur wird sich dies auswirken? Was passiert mit bereits bereitgestellten Ressourcen? Im Folgenden finden Sie zwei Beispielsituationen, in denen eine plötzliche Änderung der Werte ein Problem verursachen kann.

- Wenn Sie eine Amazon VPC für alle verfügbaren Availability Zones in einer bestimmten Region bereitstellen und die Anzahl der AZs am Bereitstellungstag zwei beträgt, wird Ihr IP-Bereich halbiert. Wenn am nächsten Tag eine neue Availability Zone AWS startet, versucht die nächste Bereitstellung danach, Ihren IP-Bereich in Dritte aufzuteilen, sodass alle Subnetze neu erstellt werden müssen. Dies ist wahrscheinlich nicht möglich, da Ihre Amazon EC2-Instances noch ausgeführt werden und Sie dies manuell bereinigen müssen.
- Wenn Sie das neueste Amazon Linux-Computer-Image abfragen und eine Amazon EC2-Instance bereitstellen und am nächsten Tag ein neues Image veröffentlicht wird, übernimmt eine nachfolgende Bereitstellung das neue AMI und ersetzt alle Ihre Instances. Dies ist möglicherweise nicht das, was Sie erwartet haben.

Diese Situationen können unheimlich sein, da die AWS-seitige Änderung nach Monaten oder Jahren erfolgreicher Bereitstellungen auftreten kann. Plötzlich schlagen Ihre Bereitstellungen „ohne Grund“ fehl und Sie haben lange vergessen, was Sie getan haben und warum.

Glücklicherweise AWS CDK enthält das einen Mechanismus namens Kontextanbieter, um einen Snapshot nicht deterministischer Werte aufzuzeichnen. Auf diese Weise können zukünftige Synthetisierungsvorgänge genau dieselbe Vorlage erzeugen wie bei der ersten Bereitstellung. Die einzigen Änderungen in der neuen Vorlage sind die Änderungen, die Sie in Ihrem Code vorgenommen haben. Wenn Sie die `.fromLookup()` Methode eines Konstrukts verwenden, wird das Ergebnis des Aufrufs in `zwischenengespeichertcdk.context.json`. Sie sollten dies zusammen mit dem Rest Ihres Codes an die Versionskontrolle übergeben, um sicherzustellen, dass zukünftige Ausführungen Ihrer CDK-App denselben Wert verwenden. Das CDK Toolkit enthält Befehle zur

Verwaltung des Kontext-Caches, sodass Sie bei Bedarf bestimmte Einträge aktualisieren können. Weitere Informationen finden Sie unter [the section called “Kontext”](#).

Wenn Sie einen Wert (von AWS oder anderswo) benötigen, für den es keinen nativen CDK-Kontextanbieter gibt, empfehlen wir, ein separates Skript zu schreiben. Das Skript sollte den Wert abrufen und in eine Datei schreiben und diese Datei dann in Ihrer CDK-App lesen. Führen Sie das Skript nur aus, wenn Sie den gespeicherten Wert aktualisieren möchten, nicht im Rahmen Ihres regulären Build-Prozesses.

Lassen Sie die Rollen und Sicherheitsgruppen AWS CDK verwalten

Mit den `grant()` Convenience-Methoden der AWS-CDK-Konstruktbibliothek können Sie AWS Identity and Access Management Rollen erstellen, die Zugriff auf eine Ressource durch eine andere gewähren, indem Sie minimal begrenzte Berechtigungen verwenden. Betrachten Sie beispielsweise eine Zeile wie die folgende:

```
myBucket.grantRead(myLambda)
```

Diese einzelne Zeile fügt der Rolle der Lambda-Funktion (die auch für Sie erstellt wird) eine Richtlinie hinzu. Diese Rolle und ihre Richtlinien umfassen mehr als ein Dutzend Zeilen, CloudFormation die Sie nicht schreiben müssen. Die AWS CDK gewährt nur die Mindestberechtigungen, die die Funktion zum Lesen aus dem Bucket benötigt.

Wenn Entwickler immer vordefinierte Rollen verwenden müssen, die von einem Sicherheitsteam erstellt wurden, wird die AWS CDK Codierung viel komplizierter. Ihre Teams verlieren möglicherweise viel Flexibilität bei der Entwicklung ihrer Anwendungen. Eine bessere Alternative besteht darin, [Service-Kontrollrichtlinien](#) und [Berechtigungsgrenzen](#) zu verwenden, um sicherzustellen, dass Entwickler innerhalb des Integritätsschutzes bleiben.

Modellieren aller Produktionsphasen im Code

In herkömmlichen AWS CloudFormation Szenarien besteht Ihr Ziel darin, ein einzelnes Artefakt zu erzeugen, das parametrisiert ist, damit es in verschiedenen Zielumgebungen bereitgestellt werden kann, nachdem Konfigurationswerte angewendet wurden, die für diese Umgebungen spezifisch sind. Im CDK können und sollten Sie diese Konfiguration in Ihren Quellcode integrieren. Erstellen Sie einen Stack für Ihre Produktionsumgebung und einen separaten Stack für jede Ihrer anderen Phasen. Geben Sie dann die Konfigurationswerte für jeden Stack in den Code ein. Verwenden Sie -Services wie [Secrets Manager](#) und [Systems Manager](#) Parameter Store für sensible Werte, die Sie nicht bei der Quellsteuerung anmelden möchten, indem Sie die Namen oder ARNs dieser Ressourcen verwenden.

Wenn Sie Ihre Anwendung synthetisieren, enthält die im `cdk.out` Ordner erstellte Cloud-Baugruppe für jede Umgebung eine separate Vorlage. Ihr gesamter Build ist deterministisch. Es gibt keine out-of-band Änderungen an Ihrer Anwendung, und jeder bestimmte Commit führt immer zu genau derselben AWS CloudFormation Vorlage und denselben zugehörigen Komponenten. Dadurch werden Komponententests viel zuverlässiger.

Alles messen

Um das Ziel einer vollständigen kontinuierlichen Bereitstellung ohne menschliche Eingriffe zu erreichen, ist ein hohes Maß an Automatisierung erforderlich. Diese Automatisierung ist nur mit umfangreichen Überwachungsmengen möglich. Um alle Aspekte Ihrer bereitgestellten Ressourcen zu messen, erstellen Sie Metriken, Alarme und Dashboards. Halten Sie nicht an, um Dinge wie CPU-Auslastung und Festplattenspeicher zu messen. Zeichnen Sie auch Ihre Geschäftsmetriken auf und verwenden Sie diese Messungen, um Bereitstellungsentscheidungen wie Rollbacks zu automatisieren. Die meisten L2-Konstrukte in AWS CDK verfügen über praktische Methoden, die Ihnen beim Erstellen von Metriken helfen, z. B. die `metricUserErrors()` Methode in der Klasse [dynamodb.Table](#).

AWS CDKCLIBefehlsreferenz

Dieser Abschnitt enthält Befehlsreferenzinformationen für die AWS Cloud Development Kit (AWS CDK) Befehlszeilenschnittstelle (CLI). Das CDK CLI wird auch als CDK-Toolkit bezeichnet.

Verwendung

```
$ cdk <command> <arguments> <options>
```

Befehle

[acknowledge, ack](#)

Bestätigen Sie einen Hinweis anhand der Problemnummer und verhindern Sie, dass er erneut angezeigt wird.

[bootstrap](#)

Bereiten Sie eine AWS Umgebung für CDK-Bereitstellungen vor, indem Sie den CDK-Bootstrap-Stack (genannt `CDKToolkit`) in der Umgebung bereitstellen. AWS

[context](#)

Verwalten Sie zwischengespeicherte Kontextwerte für Ihre CDK-Anwendung.

[deploy](#)

Stellen Sie einen oder mehrere CDK-Stacks in Ihrer Umgebung bereit. AWS

[destroy](#)

Löschen Sie einen oder mehrere CDK-Stacks aus Ihrer Umgebung. AWS

[diff](#)

Führen Sie einen Vergleich durch, um die Infrastrukturänderungen zwischen den CDK-Stacks zu sehen.

[docs, doc](#)

Öffnen Sie die CDK-Dokumentation in Ihrem Browser.

[doctor](#)

Sehen Sie sich nützliche Informationen über Ihr lokales CDK-Projekt und Ihre Entwicklungsumgebung an und zeigen Sie sie an.

[import](#)

Verwenden Sie AWS CloudFormation Ressourcenimporte, um vorhandene AWS Ressourcen in einen CDK-Stack zu importieren.

[init](#)

Erstellen Sie ein neues CDK-Projekt aus einer Vorlage.

[list, ls](#)

Listet alle CDK-Stacks und ihre Abhängigkeiten von einer CDK-App auf.

[metadata](#)

Zeigt Metadaten an, die einem CDK-Stack zugeordnet sind.

[migrate](#)

Migrieren Sie AWS Ressourcen, AWS CloudFormation Stacks und AWS CloudFormation Vorlagen in ein neues CDK-Projekt.

[notices](#)

Zeigen Sie Hinweise für Ihre CDK-Anwendung an.

[synthesize, synth](#)

Synthetisieren Sie eine CDK-App, um eine Cloud-Assembly zu erstellen, einschließlich einer AWS CloudFormation Vorlage für jeden Stapel.

[watch](#)

Beobachten Sie ein lokales CDK-Projekt kontinuierlich auf Änderungen bei der Durchführung von Implementierungen und Hotswaps.

Globale Optionen

Die folgenden Optionen sind mit allen CLI CDK-Befehlen kompatibel.

`--app, -a` *STRING*

Geben Sie den Befehl zum Ausführen Ihres App- oder Cloud-Assembly-Verzeichnisses an.

Erforderlich: Ja

`--asset-metadata` *BOOLEAN*

Fügen Sie `aws:asset:*` AWS CloudFormation Metadaten für Ressourcen hinzu, die Ressourcen verwenden.

Required: No

Standardwert: `true`

`--build` *STRING*

Befehl zum Ausführen eines Pre-Synthese-Builds.

Required: No

`--ca-bundle-path` *STRING*

Pfad zu einem CA-Zertifikat, das bei der Validierung von HTTPS-Anfragen verwendet werden soll.

Wenn diese Option nicht angegeben ist, liest das CDK CLI aus der `AWS_CA_BUNDLE` Umgebungsvariablen.

Erforderlich: Ja

`--ci` *BOOLEAN*

Geben Sie an, dass CLI CDK-Befehle in einer CI-Umgebung (Continuous Integration) ausgeführt werden.

Diese Option ändert das Verhalten des CDK, CLI um es besser an automatisierte Abläufe anzupassen, die für CI-Pipelines typisch sind.

Wenn Sie diese Option angeben, werden Protokolle an `stderr` statt an `stdout` gesendet.

Required: No

Standardwert: `false`

`--context, -c` *ARRAY*

Fügen Sie kontextbezogene Zeichenkettenparameter als Schlüssel-Wert-Paare hinzu.

`--debug` *BOOLEAN*

Aktivieren Sie detaillierte Debugging-Informationen. Diese Option erzeugt eine ausführliche Ausgabe, die viel mehr Details darüber enthält, was das CDK hinter den Kulissen CLI tut.

Required: No

Standardwert: `false`

`--ec2creds, -i` *BOOLEAN*

Zwingen Sie das CDKCLI, zu versuchen, Amazon EC2 EC2-Instance-Anmeldeinformationen abzurufen.

Standardmäßig CLI errät das CDK den Amazon EC2 EC2-Instance-Status.

Required: No

Standardwert: `false`

`--help, -h` *BOOLEAN*

Zeigt Befehlsreferenzinformationen für das CDK an. CLI

Required: No

Standardwert: `false`

`--ignore-errors` *BOOLEAN*

Ignorieren Sie Synthesefehler, da diese wahrscheinlich zu einer ungültigen Ausgabe führen.

Required: No

Standardwert: `false`

`--json, -j` *BOOLEAN*

Verwenden Sie JSON anstelle von YAML für AWS CloudFormation Vorlagen, die in der Standardausgabe (`stdout`) gedruckt werden.

Required: No

Standardwert: `false`

`--lookups` *BOOLEAN*

Führen Sie Kontext-Lookups durch.

Die Synthese schlägt fehl, wenn dieser Wert gleich ist `false` und Kontext-Lookups durchgeführt werden müssen.

Required: No

Standardwert: `true`

`--no-color` *BOOLEAN*

Entfernen Sie Farbe und anderes Design aus der Konsolenausgabe.

Required: No

Standardwert: `false`

`--notices` *BOOLEAN*

Zeigen Sie relevante Hinweise an.

Required: No

Standardwert: `false`

`--output, -o` *STRING*

Geben Sie das Verzeichnis an, in das die synthetisierte Cloud-Assembly ausgegeben werden soll.

Erforderlich: Ja

Standardwert: `cdk.out`

`--path-metadata` *BOOLEAN*

Fügen Sie `aws::cdk::path` AWS CloudFormation Metadaten für jede Ressource hinzu.

Required: No

Standardwert: `true`

`--plugin, -p` *ARRAY*

Name oder Pfad eines node Pakets, das die CDK-Funktionen erweitert. Diese Option kann in einem einzigen Befehl mehrfach angegeben werden.

Sie können diese Option in der `cdk.json` Projektdatei oder `~/.cdk.json` auf Ihrem lokalen Entwicklungscomputer konfigurieren:

```
{
  // ...
  "plugin": [
    "module_1",
    "module_2"
  ],
  // ...
}
```

Required: No

`--profile` *STRING*

Geben Sie den Namen des AWS Profils an, das Ihre AWS Umgebungsinformationen enthält und mit dem CDK CLI verwendet werden soll.

Erforderlich: Ja

`--proxy` *STRING*

Verwenden Sie den angegebenen Proxy.

Wenn diese Option nicht angegeben ist, liest CLI das CDK aus der `HTTPS_PROXY` Umgebungsvariablen.

Erforderlich: Ja

Standardwert: Aus der `HTTPS_PROXY` Umgebungsvariablen lesen.

`--role-arn, -r` *STRING*

Der ARN der IAM-Rolle, den der CDK bei der Interaktion mit ihm CLI übernimmt. AWS CloudFormation

Required: No

`--staging` *BOOLEAN*

Kopieren Sie die Assets in das Ausgabeverzeichnis.

Geben Sie `false` an, dass das Kopieren von Assets in das Ausgabeverzeichnis verhindert werden soll. Dadurch kann beim AWS SAM CLI lokalen Debuggen auf die ursprünglichen Quelldateien verwiesen werden.

Required: No

Standardwert: `true`

`--strict` *BOOLEAN*

Konstruieren Sie keine Stapel, die Warnungen enthalten.

Required: No

Standardwert: `false`

`--trace` *BOOLEAN*

Druckt den Trace für Stack-Warnungen.

Required: No

Standardwert: `false`

`--verbose`, `-v` *COUNT*

Debug-Protokolle anzeigen. Sie können diese Option mehrfach angeben, um die Ausführlichkeit zu erhöhen.

Required: No

`--version` *BOOLEAN*

Zeigt die CLI CDK-Versionsnummer an.

Required: No

Standardwert: `false`

`--version-reporting` *BOOLEAN*

Nehmen Sie die `AWS::CDK::Metadata` Ressource in synthetisierte Vorlagen AWS CloudFormation auf.

Required: No

Standardwert: `true`

Bereitstellung und Konfiguration von Optionen

Sie können Optionen über Befehlszeilenargumente übergeben. Für die meisten Optionen können Sie sie in einer `cdk.json` Konfigurationsdatei konfigurieren. Wenn Sie mehrere Konfigurationsquellen verwenden, hat das CLI CDK die folgende Priorität:

1. Befehlszeilenwerte — Jede in der Befehlszeile angegebene Option überschreibt die in Dateien konfigurierten Optionen. `cdk.json`
2. Projektkonfigurationsdatei — Die `cdk.json` Datei im Verzeichnis Ihres CDK-Projekts.
3. Benutzerkonfigurationsdatei — Die `cdk.json` Datei, die sich `~/cdk.json` auf Ihrem lokalen Computer befindet.

Optionen an der Befehlszeile übergeben

Boolesche Werte übergeben

Für Optionen, die einen booleschen Wert akzeptieren, können Sie sie auf folgende Weise angeben:

- Verwendung `true` und `false` Werte — Geben Sie den booleschen Wert mit dem Befehl an. Im Folgenden wird ein Beispiel gezeigt:

```
$ cdk deploy --watch=true
$ cdk deploy --watch=false
```

- Geben Sie das Gegenstück zur Option an — Ändern Sie den Optionsnamen, indem Sie ihn hinzufügen `no`, um einen `false` Wert anzugeben. Im Folgenden wird ein Beispiel gezeigt:

```
$ cdk deploy --watch-no
```

```
$ cdk deploy --no-watch
```

- Bei Optionen, die standardmäßig auf `true` oder gesetzt sind `false`, müssen Sie die Option nicht angeben, es sei denn, Sie möchten die Standardeinstellung ändern.

cdk acknowledge

Bestätigen Sie eine Mitteilung anhand der Problemnummer und verhindern Sie, dass sie erneut angezeigt wird.

Dies ist nützlich, um Hinweise auszublenden, die adressiert wurden oder die nicht auf Sie zutreffen.

Bestätigungen werden auf CDK-Projektebene gespeichert. Wenn Sie einen Hinweis in einem CDK-Projekt bestätigen, wird er weiterhin in anderen Projekten angezeigt, bis er dort bestätigt wird.

Verwendung

```
$ cdk acknowledge <arguments> <options>
```

Argumente

Hinweis-ID

Die ID der Mitteilung.

Typ: Zeichenfolge

Required: No

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter [Globale Optionen](#).

`--help`, `-h` **BOOLEAN**

Zeigt Befehlsreferenzinformationen für den `cdk acknowledge` Befehl an.

Beispiele

Bestätigen und verbergen Sie einen Hinweis, der angezeigt wird, wenn Sie einen anderen CDK-Befehl CLI ausführen

```
$ cdk deploy
... # Normal output of the command

NOTICES

16603  Toggling off auto_delete_objects for Bucket empties the bucket

      Overview: If a stack is deployed with an S3 bucket with
                auto_delete_objects=True, and then re-deployed with
                auto_delete_objects=False, all the objects in the bucket
                will be deleted.

      Affected versions: <1.126.0.

      More information at: https://github.com/aws/aws-cdk/issues/16603

17061  Error when building EKS cluster with monocdk import

      Overview: When using monocdk/aws-eks to build a stack containing
                an EKS cluster, error is thrown about missing
                lambda-layer-node-proxy-agent/layer/package.json.

      Affected versions: >=1.126.0 <=1.130.0.

      More information at: https://github.com/aws/aws-cdk/issues/17061

$ cdk acknowledge 16603
```

cdk bootstrap

Bereiten Sie eine AWS Umgebung für CDK-Bereitstellungen vor, indem Sie den CDK-Bootstrap-Stack (genannt) in der Umgebung bereitstellen. CDKToolkit AWS

Der Bootstrap-Stack ist ein CloudFormation Stack, der einen Amazon S3 S3-Bucket und ein Amazon ECR-Repository in der AWS Umgebung bereitstellt. The AWS CDK CLI verwendet diese

Ressourcen, um synthetisierte Vorlagen und zugehörige Ressourcen während der Bereitstellung zu speichern.

Verwendung

```
$ cdk bootstrap <arguments> <options>
```

Argumente

AWS Umgebung

Die AWS Zielumgebung, in der der Bootstrap-Stack bereitgestellt werden soll, im folgenden Format: `aws://<account-id>/<region>`.

Beispiel: `aws://123456789012/us-east-1`

Dieses Argument kann in einem einzigen Befehl mehrmals angegeben werden, um den Bootstrap-Stack in mehreren Umgebungen bereitzustellen.

Standardmäßig bootet das CDK CLI alle Umgebungen, auf die in der CDK-App verwiesen wird, oder bestimmt eine Umgebung anhand von Standardquellen. Dies kann eine Umgebung sein, die mit der `--profile` Option angegeben wurde, aus Umgebungsvariablen oder Standardquellen.

AWS CLI

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter [Globale Optionen](#).

`--bootstrap-bucket-name`, `--toolkit-bucket-name`, `-b` *STRING*

Der Name des Amazon S3 S3-Buckets, der vom CDK CLI verwendet wird. Dieser Bucket wird erstellt und darf derzeit nicht existieren.

Geben Sie diese Option an, um den Standardnamen des Amazon S3 S3-Buckets zu überschreiben.

Wenn Sie diese Option verwenden, müssen Sie die Synthese möglicherweise anpassen. Weitere Informationen hierzu finden Sie unter [Passen Sie die CDK-Stack-Synthese an](#).

Standardwert: undefiniert

`--bootstrap-customer-key` *BOOLEAN*

Erstellen Sie einen Customer Master Key (CMK) für den Bootstrap-Bucket (es fallen Gebühren an, aber Sie können die Berechtigungen anpassen, nur modernes Bootstrapping).

Diese Option ist nicht kompatibel mit `--bootstrap-kms-key-id`

Standardwert: undefiniert

`--bootstrap-kms-key-id` *STRING*

Die AWS KMS Master-Key-ID, die für die SSE-KMS Verschlüsselung verwendet werden soll.

Stellen Sie diese Option bereit, um den AWS KMS Standardschlüssel zu überschreiben, der zum Verschlüsseln des Amazon S3 S3-Buckets verwendet wird.

Diese Option ist nicht kompatibel mit `--bootstrap-customer-key`.

Standardwert: undefiniert

`--cloudformation-execution-policies` *ARRAY*

Die verwalteten IAM-Policy-ARNs, die der Bereitstellungsrolle zugewiesen werden sollten, die AWS CloudFormation während der Bereitstellung Ihrer Stacks übernommen wurde.

Standardmäßig werden Stacks mithilfe der Richtlinie mit vollen Administratorrechten bereitgestellt. `AdministratorAccess`

Sie können diese Option in einem einzigen Befehl mehrfach angeben. Sie können auch mehrere ARNs als eine einzige Zeichenfolge angeben, wobei die einzelnen ARNs durch Kommas getrennt sind. Im Folgenden wird ein Beispiel gezeigt:

```
$ cdk bootstrap --cloudformation-execution-policies "arn:aws:iam::aws:policy/  
AWSLambda_FullAccess,arn:aws:iam::aws:policy/AWSCodeDeployFullAccess"
```

Um Bereitstellungsfehler zu vermeiden, stellen Sie sicher, dass die von Ihnen angegebenen Richtlinien für alle Bereitstellungen, die Sie in der Umgebung durchführen, für die das Bootstrapping ausgeführt wird, ausreichend sind.

Diese Option gilt nur für modernes Bootstrapping.

⚠ Important

Die moderne Bootstrap-Vorlage gewährt praktisch jedem AWS Konto in der Liste die in der `--cloudformation-execution-policies` Liste enthaltenen Berechtigungen. `--trust` Standardmäßig werden dadurch die Lese- und Schreibberechtigungen für alle Ressourcen im Bootstrap-Konto erweitert. Stellen Sie sicher, dass [Sie den Bootstrapping-Stack mit Richtlinien und vertrauenswürdigen Konten konfigurieren](#), mit denen Sie vertraut sind.

Standardwert: []

`--custom-permissions-boundary, -cpb` *STRING*

Geben Sie den Namen einer zu verwendenden Berechtigungsgrenze an.

Diese Option ist nicht kompatibel mit `--example-permissions-boundary`.

Standardwert: undefiniert

`--example-permissions-boundary, -epb` *BOOLEAN*

Verwenden Sie das Beispiel für die Berechtigungsgrenze, die von der AWS CDK bereitgestellt wird.

Diese Option ist nicht kompatibel mit `--custom-permissions-boundary`.

Die vom CDK bereitgestellte Grenzrichtlinie für Berechtigungen sollte als Beispiel betrachtet werden. Bearbeiten Sie den Inhalt und verweisen Sie auf die Beispielrichtlinie, wenn Sie die Funktion testen. Wandeln Sie sie in eine neue Richtlinie für tatsächliche Bereitstellungen um, falls noch keine vorhanden ist. Es geht darum, Abweichungen zu vermeiden. Höchstwahrscheinlich wird eine Rechtegrenze beibehalten und es gibt spezielle Konventionen, einschließlich der Benennung.

Weitere Informationen zur Konfiguration von Berechtigungen, einschließlich der Verwendung von Rechtegrenzen, finden Sie im [Security and Safety Dev Guide](#).

Standardwert: undefiniert

`--execute` *BOOLEAN*

Konfigurieren Sie, ob der Änderungssatz ausgeführt werden soll.

Standardwert: `true`

`--force, -f` *BOOLEAN*

Führen Sie immer Bootstrap durch, auch wenn dadurch die Bootstrap-Vorlagenversion herabgestuft würde.

Standardwert: `false`

`--help, -h` *BOOLEAN*

Zeigt die Befehlsreferenzinformationen für den Befehl an. `cdk bootstrap`

`--previous-parameters` *BOOLEAN*

Verwenden Sie vorherige Werte für vorhandene Parameter.

Sobald eine Bootstrap-Vorlage mit einer Reihe von Parametern bereitgestellt wurde, müssen Sie diese Option auf `false` setzen, um alle Parameter bei future Bereitstellungen zu ändern. Wenn `false`, müssen Sie alle zuvor angegebenen Parameter erneut angeben.

Standardwert: `true`

`--public-access-block-configuration` *BOOLEAN*

Blockieren Sie die öffentliche Zugriffskonfiguration für den Amazon S3 S3-Bucket, der vom CDK CLI erstellt und verwendet wird.

Standardwert: `true`

`--qualifier` *STRING*

Zeichenkettenwert, der für jeden Bootstrap-Stack eindeutig ist. Dieser Wert wird zur physischen ID der Ressourcen im Bootstrap-Stack hinzugefügt.

Durch die Bereitstellung eines Qualifizierers vermeiden Sie Konflikte bei Ressourcennamen, wenn Sie mehrere Bootstrap-Stacks in derselben Umgebung bereitstellen.

Wenn Sie den Qualifier ändern, muss Ihre CDK-App den geänderten Wert an den Stack-Synthesizer übergeben. Weitere Informationen finden Sie unter [Passen Sie die CDK-Stack-Synthese an](#).

Standardwert: `hnb659fds`. Dieser Wert hat keine Bedeutung.

`--show-template` *BOOLEAN*

Anstatt Bootstrapping durchzuführen, drucken Sie die aktuelle Bootstrap-Vorlage auf die Standardausgabe (`stdout`). Anschließend können Sie die Vorlage kopieren und nach Bedarf anpassen.

Standardwert: `false`

`--tags, -t` *ARRAY*

Tags, die dem Bootstrap-Stack hinzugefügt werden sollen, im Format von `KEY=VALUE`.

Standardwert: `[]`

`--template` *STRING*

Verwenden Sie die Vorlage aus der angegebenen Datei anstelle der integrierten.

`--termination-protection` *BOOLEAN*

Schalten Sie den AWS CloudFormation Terminierungsschutz auf dem Bootstrap-Stack um.

Wann ist `true` der Kündigungsschutz aktiviert. Dadurch wird verhindert, dass der Bootstrap-Stack versehentlich gelöscht wird.

Weitere Informationen zum Kündigungsschutz finden Sie im AWS CloudFormation Benutzerhandbuch unter [Einen Stack vor dem Löschen schützen](#).

Standardwert: undefiniert

`--toolkit-stack-name` *STRING*

Der Name des zu erstellenden Bootstrap-Stacks.

`cdk bootstrap` stellt standardmäßig einen Stack mit dem Namen `CDKToolkit` in der angegebenen AWS Umgebung bereit. Verwenden Sie diese Option, um einen anderen Namen für Ihren Bootstrap-Stack anzugeben.

Standardwert: `CDKToolkit`

Erforderlich: Ja

`--trust` *ARRAY*

Die AWS-Konto IDs, denen bei der Durchführung von Bereitstellungen in dieser Umgebung vertraut werden sollte.

Dem Konto, das das Bootstrapping durchführt, wird immer vertraut.

Diese Option erfordert, dass Sie auch Folgendes angeben. `--cloudformation-execution-policies`

Sie können diese Option in einem einzigen Befehl mehrfach angeben.

Diese Option gilt nur für modernes Bootstrapping.

Um vertrauenswürdige Konten zu einem vorhandenen Bootstrap-Stack hinzuzufügen, müssen Sie alle Konten angeben, denen Sie vertrauen möchten, einschließlich der Konten, die Sie möglicherweise zuvor angegeben haben. Wenn Sie nur neue Konten angeben, denen Sie vertrauen können, werden die zuvor vertrauenswürdigen Konten entfernt.

Im Folgenden finden Sie ein Beispiel, bei dem zwei Konten als vertrauenswürdige eingestuft werden:

```
$ cdk bootstrap aws://123456789012/us-west-2 --trust 234567890123 --
trust 987654321098 --cloudformation-execution-policies arn:aws:iam::aws:policy/
AdministratorAccess
# Bootstrapping environment aws://123456789012/us-west-2...
Trusted accounts for deployment: 234567890123, 987654321098
Trusted accounts for lookup: (none)
Execution policies: arn:aws:iam::aws:policy/AdministratorAccess
CDKToolkit: creating CloudFormation changeset...
# Environment aws://123456789012/us-west-2 bootstrapped.
```

Important

Die moderne Bootstrap-Vorlage gewährt praktisch jedem AWS Konto in der Liste die in der `--cloudformation-execution-policies` Liste enthaltenen Berechtigungen. `--trust` Standardmäßig werden dadurch die Lese- und Schreibberechtigungen für alle Ressourcen im Bootstrap-Konto erweitert. Stellen Sie sicher, dass [Sie den Bootstrapping-Stack mit Richtlinien und vertrauenswürdigen Konten konfigurieren](#), mit denen Sie vertraut sind.

Standardwert: []

`--trust-for-lookup` *ARRAY*

Die AWS-Konto IDs, denen beim Nachschlagen von Werten in dieser Umgebung vertraut werden sollte.

Verwenden Sie diese Option, um Konten die Erlaubnis zu erteilen, Stacks zu synthetisieren, die in der Umgebung bereitgestellt werden, ohne ihnen tatsächlich die Erlaubnis zu geben, diese Stacks direkt bereitzustellen.

Sie können diese Option in einem einzigen Befehl mehrfach angeben.

Diese Option gilt nur für modernes Bootstrapping.

Standardwert: []

Beispiele

Bootstrap für die im AWS Produktprofil angegebene Umgebung

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

Stellen Sie den Bootstrap-Stack in den Umgebungen foo und bar bereit

```
$ cdk bootstrap --app='node bin/main.js' foo bar
```

Exportieren Sie die Bootstrap-Vorlage, um sie anzupassen

Wenn Sie spezielle Anforderungen haben, die von der Bootstrap-Vorlage nicht erfüllt werden, können Sie sie an Ihre Bedürfnisse anpassen.

Sie können die Bootstrap-Vorlage exportieren, ändern und mithilfe dieser Methode bereitstellen. AWS CloudFormation Im Folgenden finden Sie ein Beispiel für den Export der vorhandenen Vorlage:

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

Sie können das CDK auch anweisen, eine benutzerdefinierte Vorlage CLI zu verwenden. Im Folgenden wird ein Beispiel gezeigt:

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


Bootstrap mit einer Berechtigungsgrenze. Entfernen Sie dann diese Berechtigungsgrenze

Um mit einer benutzerdefinierten Berechtigungsgrenze zu booten, führen wir Folgendes aus:

```
$ cdk bootstrap --custom-permissions-boundary my-permissions-boundary
```

Um die Berechtigungsgrenze zu entfernen, führen wir Folgendes aus:

```
$ cdk bootstrap --no-previous-parameters
```

Verwenden Sie einen Qualifier, um Ressourcen zu unterscheiden, die für eine Entwicklungsumgebung erstellt wurden

```
$ cdk bootstrap --qualifier dev2024
```

cdk context

Verwaltet zwischengespeicherte Kontextwerte für Ihre AWS CDK Anwendung.

Der Kontext stellt die Konfiguration und Umgebungsinformationen dar, die beeinflussen können, wie Ihre Stacks synthetisiert und bereitgestellt werden. Mit `cdk context` können folgende Arbeiten durchgeführt werden:

- Sehen Sie sich Ihre konfigurierten Kontextwerte an.
- Legen Sie Kontextwerte fest und verwalten Sie sie.
- Kontextwerte entfernen.

Verwendung

```
$ cdk context <options>
```

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter [Globale Optionen](#).

`--clear` *BOOLEAN*

Löscht den gesamten Kontext.

`--force`, `-f` *BOOLEAN*

Ignoriere den Fehler mit fehlendem Schlüssel.

Standardwert: `false`

`--help`, `-h` *BOOLEAN*

Zeigt Befehlsreferenzinformationen für den `cdk context` Befehl an.

`--reset`, `-e` *STRING*

Der Kontextschlüssel oder sein Index, der zurückgesetzt werden soll.

cdk deploy

Stellen Sie einen oder mehrere AWS CDK Stacks in Ihrer AWS Umgebung bereit.

Während der Bereitstellung gibt das CDK CLI Fortschrittsindikatoren aus, die denen ähneln, die von der AWS CloudFormation Konsole aus beobachtet werden können.

Wenn in der AWS Umgebung kein Bootstrapping durchgeführt wird, können nur Stacks ohne Ressourcen und mit synthetisierten Vorlagen unter 51.200 Byte erfolgreich bereitgestellt werden.

Verwendung

```
$ cdk deploy <arguments> <options>
```

Argumente

Logische ID des CDK-Stacks

Die logische ID des CDK-Stacks aus Ihrer App, die bereitgestellt werden soll.

Typ: Zeichenfolge

Required: No

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter.

[Globale Optionen](#)

`--all` *BOOLEAN*

Stellen Sie alle Stacks in Ihrer CDK-App bereit.

Standardwert: `false`

`--asset-parallelism` *BOOLEAN*

Geben Sie an, ob Assets parallel erstellt und veröffentlicht werden sollen.

`--asset-prebuild` *BOOLEAN*

Geben Sie an, ob alle Assets erstellt werden sollen, bevor der erste Stack bereitgestellt wird. Diese Option ist nützlich für fehlgeschlagene Docker Builds.

Standardwert: `true`

`--build-exclude, -E` *ARRAY*

Das Asset mit der angegebenen ID darf nicht neu erstellt werden.

Diese Option kann in einem einzigen Befehl mehrfach angegeben werden.

Standardwert: `[]`

`--change-set-name` *STRING*

Der Name des zu erstellenden AWS CloudFormation Änderungssatzes.

Diese Option ist nicht kompatibel mit `--method='direct'`.

`--concurrency` *NUMBER*

Stellen Sie mehrere Stacks parallel bereit und berücksichtigen Sie dabei die Abhängigkeiten zwischen den Stacks. Verwenden Sie diese Option, um Bereitstellungen zu beschleunigen. Sie müssen immer noch andere AWS CloudFormation AWS-Konto Ratenbegrenzungen berücksichtigen.

Geben Sie eine Zahl ein, um die maximale Anzahl gleichzeitiger Bereitstellungen anzugeben (sofern die Abhängigkeit dies zulässt).

Standardwert: 1

--exclusively, -e *BOOLEAN*

Stellen Sie nur angeforderte Stacks bereit und schließen Sie keine Abhängigkeiten ein.

--force, -f *BOOLEAN*

Wenn Sie einen vorhandenen Stack bereitstellen, um ihn zu aktualisieren, vergleicht das CDK CLI die Vorlage und die Tags des bereitgestellten Stacks mit dem Stack, der gerade bereitgestellt werden soll. Wenn keine Änderungen festgestellt werden, überspringt das CDK CLI die Bereitstellung.

Um dieses Verhalten zu überschreiben und Stacks immer bereitzustellen, auch wenn keine Änderungen erkannt werden, verwenden Sie diese Option.

Standardwert: false

--help, -h *BOOLEAN*

Zeigt Befehlsreferenzinformationen für den `cdk deploy` Befehl an.

--hotswap *BOOLEAN*

Hotswap-Bereitstellungen für eine schnellere Entwicklung. Mit dieser Option wird versucht, wenn möglich eine schnellere Hotswap-Bereitstellung durchzuführen. Wenn Sie beispielsweise den Code einer Lambda-Funktion in Ihrer CDK-App ändern, aktualisiert das CDK CLI die Ressource direkt über Service-APIs, anstatt eine Bereitstellung durchzuführen. CloudFormation

Wenn das CDK Änderungen CLI erkennt, die Hotswapping nicht unterstützen, werden diese Änderungen ignoriert und es wird eine Meldung angezeigt. Wenn Sie es vorziehen, eine vollständige CloudFormation Bereitstellung als Ausweidlösung durchzuführen, verwenden Sie stattdessen. `--hotswap-fallback`

Das CDK CLI verwendet Ihre aktuellen AWS Anmeldeinformationen, um die API-Aufrufe durchzuführen. Es übernimmt nicht die Rollen aus Ihrem Bootstrap-Stack, auch wenn das `@aws-cdk/core:newStyleStackSynthesis` Feature-Flag auf gesetzt ist. `true` Diese Rollen verfügen nicht über die erforderlichen Berechtigungen, um AWS Ressourcen direkt zu aktualisieren, ohne sie zu verwenden CloudFormation. Stellen Sie daher sicher, dass Ihre Anmeldeinformationen für dieselben AWS-Konto Stacks gelten, für die Sie Hotswap-Bereitstellungen durchführen, und dass sie über die erforderlichen IAM-Berechtigungen verfügen, um die Ressourcen zu aktualisieren.

Hotswapping wird derzeit für die folgenden Änderungen unterstützt:

- Code-Assets (einschließlich Docker Bilder und Inline-Code), Tag-Änderungen und Konfigurationsänderungen (nur Beschreibungs- und Umgebungsvariablen werden unterstützt) von Lambda-Funktionen.
- Lambda-Versionen und Alias-Änderungen.
- Definitionsänderungen von AWS Step Functions Zustandsmaschinen.
- Änderungen der Container-Assets der Amazon ECS-Services.
- Änderungen der Website-Ressourcen von Amazon S3 S3-Bucket-Bereitstellungen.
- Änderungen an Quelle und Umgebung von AWS CodeBuild Projekten.
- Änderungen an der VTL-Mapping-Vorlage für AWS AppSync Resolver und Funktionen.
- Schemaänderungen für APIs AWS AppSync GraphQL.

Die Verwendung bestimmter CloudFormation systemeigener Funktionen wird im Rahmen einer Hotswap-Bereitstellung unterstützt. Dazu zählen:

- Ref
- Fn::GetAtt— Nur teilweise unterstützt. Informationen zu den unterstützten Ressourcen und Attributen finden Sie in [dieser Implementierung](#).
- Fn::ImportValue
- Fn::Join
- Fn::Select
- Fn::Split
- Fn::Sub

Diese Option ist auch mit verschachtelten Stacks kompatibel.

Note

- Diese Option führt bewusst zu einer Drift bei den CloudFormation Stacks, um Bereitstellungen zu beschleunigen. Verwenden Sie sie aus diesem Grund nur für Entwicklungszwecke. Verwenden Sie diese Option nicht für Ihre Produktionsbereitstellungen.
- Diese Option gilt als experimentell und könnte sich in future grundlegend ändern.
- Die Standardwerte für bestimmte Parameter können je nach Hotswap-Parameter unterschiedlich sein. Beispielsweise wird für einen Amazon ECS-Service derzeit der

Mindestfehlergrad auf festgelegt. Überprüfen Sie die Quelle entsprechend, falls dies der Fall ist.

Standardwert: `false`

`--hotswap-fallback` *BOOLEAN*

Diese Option ist ähnlich wie `--hotswap`. Der Unterschied besteht darin, dass auf eine vollständige CloudFormation Bereitstellung zurückgegriffen `--hotswap-fallback` wird, wenn eine Änderung erkannt wird, die dies erfordert.

Weitere Informationen zu dieser Option finden Sie unter `--hotswap`.

Standardwert: `false`

`--ignore-no-stacks` *BOOLEAN*

Führen Sie eine Bereitstellung durch, auch wenn Ihre CDK-App keine Stacks enthält.

Diese Option ist im folgenden Szenario hilfreich: Möglicherweise haben Sie eine App mit mehreren Umgebungen, z. B. `dev` und `prod`. Zu Beginn der Entwicklung verfügt Ihre Prod-App möglicherweise über keine Ressourcen, oder die Ressourcen sind auskommentiert. Dies führt zu einem Bereitstellungsfehler mit einer Meldung, dass die App keine Stacks hat. Wird verwendet `--ignore-no-stacks`, um diesen Fehler zu umgehen.

Standardwert: `false`

`--logs` *BOOLEAN*

Zeigt das CloudWatch Amazon-Protokoll in der Standardausgabe (`stdout`) für alle Ereignisse aus allen Ressourcen in den ausgewählten Stacks an.

Diese Option ist nur kompatibel mit `--watch`.

Standardwert: `true`

`--method, -m` *STRING*

Konfigurieren Sie die Methode zur Durchführung einer Bereitstellung.

- `change-set`— Standardmethode. Das CDK CLI erstellt einen CloudFormation Änderungssatz mit den Änderungen, die bereitgestellt werden, und führt dann die Bereitstellung durch.

- `direct`— Erstellen Sie keinen Änderungssatz. Wenden Sie die Änderung stattdessen sofort an. Dies ist in der Regel schneller als das Erstellen eines Änderungssatzes, aber Sie verlieren die Fortschrittsinformationen.
- `prepare-change-set`— Erstellen Sie einen Änderungssatz, führen Sie aber keine Bereitstellung durch. Dies ist nützlich, wenn Sie über externe Tools verfügen, die den Änderungssatz überprüfen, oder wenn Sie über ein Genehmigungsverfahren für Änderungssätze verfügen.

Gültige Werte:`change-set,direct,prepare-change-set`

Standardwert: `change-set`

`--notification-arns` *ARRAY*

Die ARNs der Amazon SNS SNS-Themen, die über Ereignisse im Zusammenhang mit dem Stack CloudFormation benachrichtigen.

`--outputs-file, -o` *STRING*

Der Pfad, in den die Stack-Ausgaben von Bereitstellungen geschrieben werden.

Nach der Bereitstellung werden die Stack-Ausgaben im JSON-Format in die angegebene Ausgabedatei geschrieben.

Sie können diese Option in der `cdk.json` Projektdatei oder `~/.cdk.json` auf Ihrem lokalen Entwicklungscomputer konfigurieren:

```
{
  "app": "npx ts-node bin/myproject.ts",
  // ...
  "outputsFile": "outputs.json"
}
```

Wenn mehrere Stapel bereitgestellt werden, werden die Ausgaben in dieselbe Ausgabedatei geschrieben, die nach Schlüsseln geordnet ist, die den Stacknamen darstellen.

`--parameters` *ARRAY*

Übergeben Sie CloudFormation während der Bereitstellung zusätzliche Parameter an.

Diese Option akzeptiert ein Array im folgenden Format:`STACK:KEY=VALUE`.

- `STACK`— Der Name des Stacks, dem der Parameter zugeordnet werden soll.
- `KEY`— Der Name des Parameters aus Ihrem Stack.

- **VALUE**— Der Wert, der bei der Bereitstellung übergeben werden soll.

Wenn kein Stack-Name angegeben wird oder wenn * er als Stack-Name angegeben wird, werden Parameter auf alle Stacks angewendet, die bereitgestellt werden. Wenn ein Stack den Parameter nicht verwendet, schlägt die Bereitstellung fehl.

Parameter werden nicht auf verschachtelte Stacks übertragen. Verwenden Sie das Konstrukt, um Parameter an verschachtelte Stacks zu übergeben. [NestedStack](#)

Standardwert: `{}`

`--previous-parameters` **BOOLEAN**

Verwenden Sie vorherige Werte für bestehende Parameter.

Wenn diese Option auf gesetzt ist `false`, müssen Sie bei jeder Bereitstellung alle Parameter angeben.

Standardwert: `true`

`--progress` **STRING**

Konfigurieren Sie, wie das CDK den Bereitstellungsfortschritt CLI anzeigt.

- `bar`— Zeigt Ereignisse bei der Stack-Bereitstellung als Fortschrittsbalken an, zusammen mit den Ereignissen für die Ressource, die gerade bereitgestellt wird.
- `events`— Stellen Sie einen vollständigen Verlauf bereit, einschließlich aller CloudFormation Ereignisse.

Sie können diese Option auch in der `cdk.json` Projektdatei oder `~/.cdk.json` auf Ihrem lokalen Entwicklungscomputer konfigurieren:

```
{
  "progress": "events"
}
```

Zulässige Werte: `bar`, `events`

Standardwert: `bar`

`--require-approval` **STRING**

Geben Sie an, für welche sicherheitsrelevanten Änderungen eine manuelle Genehmigung erforderlich ist.

- `any-change` — Für jede Änderung am Stack ist eine manuelle Genehmigung erforderlich.
- `broadening`— Manuelle Genehmigung erforderlich, wenn Änderungen eine Erweiterung der Berechtigungen oder Sicherheitsgruppenregeln beinhalten.
- `never`— Eine Genehmigung ist nicht erforderlich.

Gültige Werte:`any-change`,`broadening`, `never`


Standardwert: `broadening`

`--rollback` ***BOOLEAN***

Wenn während der Bereitstellung eine Ressource nicht erstellt oder aktualisiert werden kann, wird die Bereitstellung auf den letzten stabilen Status zurückgesetzt, bevor der CDK CLI zurückkehrt. Alle bis zu diesem Zeitpunkt vorgenommenen Änderungen werden rückgängig gemacht. Ressourcen, die erstellt wurden, werden gelöscht und vorgenommene Aktualisierungen werden rückgängig gemacht.

Geben Sie `false` an, ob dieses Verhalten deaktiviert werden soll. Wenn eine Ressource nicht erstellt oder aktualisiert werden kann, behält das CDK CLI die bis zu diesem Zeitpunkt vorgenommenen Änderungen bei und kehrt zurück. Dies kann in Entwicklungsumgebungen hilfreich sein, in denen Sie schnell iterieren.

Für `--rollback=false` können Sie `--no-rollback` oder `-R` verwenden.

 Note

Wenn `false`, Bereitstellungen, die zu einem Austausch von Ressourcen führen, schlagen immer fehl. Sie können diesen Optionswert nur für Bereitstellungen verwenden, bei denen neue Ressourcen aktualisiert oder erstellt werden.

Standardwert: `true`

`--toolkit-stack-name` ***STRING***

Der Name des vorhandenen CDK Toolkit-Stacks.

Diese Option wird nur für CDK-Apps verwendet, die Legacy-Synthese verwenden.

`--watch` ***BOOLEAN***

Beobachten Sie CDK-Projektdateien kontinuierlich und stellen Sie die angegebenen Stacks automatisch bereit, wenn Änderungen erkannt werden.

Diese Option impliziert `--hot-swap` standardmäßig.

Diese Option hat einen äquivalenten CLI CDK-Befehl. Weitere Informationen finden Sie unter [cdk watch](#).

Beispiele

Stellen Sie den Stack mit dem Namen bereit `MyStackName`

```
$ cdk deploy MyStackName --app='node bin/main.js'
```

Stellen Sie mehrere Stacks in einer App bereit

Verwenden Sie `cdk list`, um Ihre Stacks aufzulisten:

```
$ cdk list
CdkHelloWorldStack
CdkStack2
CdkStack3
```

Verwenden Sie die Option, um alle Stacks bereitzustellen: `--all`

```
$ cdk deploy --all
```

Um auszuwählen, welche Stacks bereitgestellt werden sollen, geben Sie Stack-Namen als Argumente an:

```
$ cdk deploy CdkHelloWorldStack CdkStack3
```

Pipeline-Stacks bereitstellen

Wird verwendet `cdk list`, um Stack-Namen als Pfade anzuzeigen und zu zeigen, wo sie sich in der Pipeline-Hierarchie befinden:

```
$ cdk list
PipelineStack
PipelineStack/Prod
PipelineStack/Prod/MyService
```

Verwenden Sie die `--all` Option oder den Platzhalter`*`, um alle Stacks bereitzustellen. Wenn Sie eine Hierarchie von Stapeln haben, wie oben beschrieben, `--all` und `*` nur Stapel auf der obersten Ebene abgleichen möchten. Um alle Stapel in der Hierarchie abzugleichen, verwenden Sie `**`.

Sie können diese Muster kombinieren. Im Folgenden werden alle Stapel in der `Prod` Phase bereitgestellt:

```
$ cdk deploy PipelineStack/Prod/**
```

Übergeben Sie die Parameter bei der Bereitstellung

Definieren Sie Parameter in Ihrem CDK-Stack. Im Folgenden finden Sie ein Beispiel, das einen Parameter erstellt, der `TopicNameParam` nach einem Amazon SNS SNS-Thema benannt ist:

```
new sns.Topic(this, 'TopicParameter', {
  topicName: new cdk.CfnParameter(this, 'TopicNameParam').value.toString()
});
```

Um den Parameterwert von `anzugebenparameterized`, führen Sie den folgenden Befehl aus:

```
$ cdk deploy --parameters "MyStackName:TopicNameParam=parameterized"
```

Sie können Parameterwerte überschreiben, indem Sie die `--force` Option verwenden. Im Folgenden finden Sie ein Beispiel für das Überschreiben des Themennamens aus einer früheren Bereitstellung:

```
$ cdk deploy --parameters "MyStackName:TopicNameParam=parameterName" --force
```

Schreiben Sie die Stack-Ausgaben nach der Bereitstellung in eine Datei

Definieren Sie Ausgaben in Ihrer CDK-Stack-Datei. Im Folgenden finden Sie ein Beispiel, das eine Ausgabe für eine Funktion ARN erstellt:

```
const fn = new lambda.Function(this, "fn", {
  handler: "index.handler",
  code: lambda.Code.fromInline(`exports.handler = \`${handler.toString()}``),
  runtime: lambda.Runtime.NODEJS_LATEST
});

new cdk.CfnOutput(this, 'FunctionArn', {
```

```
value: fn.functionArn,  
});
```

Stellen Sie den Stack bereit und schreiben Sie die Ausgaben in `outputs.json`:

```
$ cdk deploy --outputs-file outputs.json
```

Das Folgende ist ein Beispiel für die Zeit `outputs.json` nach der Bereitstellung:

```
{  
  "MyStack": {  
    "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:MyStack-fn5FF616E3-  
G632ITHSP5HK"  
  }  
}
```

In diesem Beispiel `FunctionArn` entspricht der Schlüssel der logischen ID der `CfnOutput` Instanz.

Im Folgenden finden Sie ein Beispiel für die Zeit `outputs.json` nach der Bereitstellung, wenn mehrere Stacks bereitgestellt werden:

```
{  
  "MyStack": {  
    "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:MyStack-fn5FF616E3-  
G632ITHSP5HK"  
  },  
  "AnotherStack": {  
    "VPCId": "vpc-z0mg270fee16693f"  
  }  
}
```

Ändern Sie die Bereitstellungsmethode

Um die Bereitstellung ohne Verwendung von Änderungssätzen zu beschleunigen, verwenden Sie `--method='direct'`:

```
$ cdk deploy --method='direct'
```

Um einen Änderungssatz zu erstellen, ihn aber nicht bereitzustellen, verwenden Sie `--method='prepare-change-set'`. Standardmäßig wird ein Änderungssatz mit `cdk-deploy-`

change-set dem Namen erstellt. Wenn ein früherer Änderungssatz mit diesem Namen existiert, wird er überschrieben. Wenn keine Änderungen erkannt werden, wird trotzdem ein leerer Änderungssatz erstellt.

Sie können Ihrem Änderungssatz auch einen Namen geben. Im Folgenden wird ein Beispiel gezeigt:

```
$ cdk deploy --method='prepare-change-set' --change-set-name='MyChangeSetName'
```

cdk destroy

Löschen Sie einen oder mehrere AWS CDK Stapel aus Ihrer AWS Umgebung.

Wenn Sie einen Stack löschen, werden die Ressourcen im Stack zerstört, sofern sie nicht mit einem DeletionPolicy of Retain konfiguriert wurden.

Beim Löschen eines Stacks gibt dieser Befehl Fortschrittsinformationen aus, die dem `cdk deploy` Verhalten ähneln.

Verwendung

```
$ cdk destroy <arguments> <options>
```

Argumente

Logische ID des CDK-Stacks

Die logische ID des CDK-Stacks aus Ihrer App, der gelöscht werden soll.

Typ: Zeichenfolge

Required: No

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter

[Globale Optionen](#)

`--all` **BOOLEAN**

Zerstört alle verfügbaren Stapel.

Standardwert: `false`

`--exclusively`, `-e` *BOOLEAN*

Zerstört nur angeforderte Stacks und schließt keine Abhängigkeiten ein.

`--force`, `-f` *BOOLEAN*

Bitte Sie nicht um Bestätigung, bevor Sie die Stacks zerstören.

`--help`, `-h` *BOOLEAN*

Zeigt die Befehlsreferenzinformationen für den `cdk destroy` Befehl an.

Beispiele

Löscht einen Stapel mit dem Namen `MyStackName`

```
$ cdk destroy --app='node bin/main.js' MyStackName
```

cdk diff

Führen Sie einen Vergleich durch, um die Infrastrukturänderungen zwischen den AWS CDK Stacks zu sehen.

Dieser Befehl wird normalerweise verwendet, um Unterschiede zwischen dem aktuellen Status der Stacks in Ihrer lokalen CDK-App und den bereitgestellten Stacks zu vergleichen. Sie können jedoch auch einen bereitgestellten Stack mit einer beliebigen lokalen Vorlage vergleichen. AWS CloudFormation

Verwendung

```
$ cdk diff <arguments> <options>
```

Argumente

Logische ID des CDK-Stacks

Die logische ID des CDK-Stacks aus Ihrer App, um einen Vergleich durchzuführen.

Typ: Zeichenfolge

Required: No

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter.

[Globale Optionen](#)

`--change-set` *BOOLEAN*

Geben Sie an, ob ein Änderungssatz erstellt werden soll, um den Austausch von Ressourcen zu analysieren.

Wann erstellt das CDK einen AWS CloudFormation Änderungssatz `true`, in dem die genauen Änderungen angezeigt CLI werden, die an Ihrem Stack vorgenommen werden. Diese Ausgabe beinhaltet, ob Ressourcen aktualisiert oder ersetzt werden. Das CDK CLI verwendet die Bereitstellungsrolle anstelle der Suchrolle, um diese Aktion auszuführen.

Wenn `false` durch den Vergleich von Vorlagen ein schnellerer, aber weniger genauer Vergleich durchgeführt wird. CloudFormation Jede Änderung, die an Eigenschaften erkannt wird, für die eine Ressource ersetzt werden muss, wird als Ressourcenersatz angezeigt, auch wenn es sich um eine rein kosmetische Änderung handelt, z. B. das Ersetzen einer Ressourcenreferenz durch einen hartcodierten ARN.

Standardwert: `true`

`--context-lines` *NUMBER*

Anzahl der Kontextzeilen, die in das beliebige JSON-Diff-Rendering aufgenommen werden sollen.

Standardwert: `3`

`--exclusively, -e` *BOOLEAN*

Vergleicht nur angeforderte Stacks und schließt keine Abhängigkeiten ein.

`--fail` *BOOLEAN*

Schlägt fehl und beendet das Programm mit dem Code 1, ob Unterschiede erkannt wurden.

`--help, -h` *BOOLEAN*

Zeigt Befehlsreferenzinformationen für den `cdk diff` Befehl an.

`--processed` *BOOLEAN*

Geben Sie an, ob bereits verarbeitete CloudFormation Transformationen mit der Vorlage verglichen werden sollen.

Standardwert: `false`

`--quiet, -q` *BOOLEAN*

Drucken Sie den CDK-Stacknamen und die `cdk diff` Standardnachricht nicht aus, `stdout` wenn keine Änderungen erkannt wurden.

Standardwert: `false`

`--security-only` *BOOLEAN*

Nur Diff für erweiterte Sicherheitsänderungen.

Standardwert: `false`

`--strict` *BOOLEAN*

Ändern Sie `cdk diff` das Verhalten, sodass es genauer oder strenger ist. Wenn der Wert wahr ist, filtert das CDK CLI keine `AWS::CDK::Metadata` Ressourcen oder unlesbare Nicht-ASCII-Zeichen heraus.

Standardwert: `false`

`--template` *STRING*

Der Pfad zur CloudFormation Vorlage, mit der ein CDK-Stack verglichen werden soll.

Beispiele

Vergleich mit dem aktuell bereitgestellten Stack mit dem Namen `MyStackName`

```
$ cdk diff MyStackName --app='node bin/main.js'
```

Vergleich mit einer bestimmten CloudFormation Vorlage

```
$ cdk diff MyStackName --app='node bin/main.js' --template-path='./MyStackNameTemplate.yaml'
```


Vergleicht einen lokalen Stack mit seinem bereitgestellten Stack. Drucken Sie nicht auf Standardausgabe, wenn keine Änderungen erkannt wurden

```
$ cdk diff MyStackName --app='node bin/main.js' --quiet
```

cdk docs

Öffnen Sie die AWS CDK Dokumentation in Ihrem Browser.

Verwendung

```
$ cdk docs <options>
```

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter [Globale Optionen](#).

`--browser`, `-b` *STRING*

Der zum Öffnen des Browsers zu verwendende Befehl, der %u als Platzhalter für den Pfad der zu öffnenden Datei verwendet wird.

Standardwert: open %u

`--help`, `-h` *BOOLEAN*

Zeigt Befehlsreferenzinformationen für den `cdk docs` Befehl an.

Beispiele

Öffnen Sie die AWS CDK Dokumentation in Google Chrome

```
$ cdk docs --browser='chrome %u'
```

cdk doctor

Untersuchen und zeigen Sie nützliche Informationen über Ihr lokales AWS CDK Projekt und Ihre Entwicklungsumgebung an.

Diese Informationen können bei der Behebung von CDK-Problemen helfen und sollten bei der Einreichung von Fehlerberichten angegeben werden.

Verwendung

```
$ cdk doctor <options>
```

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter [Globale Optionen](#).

[Globale Optionen](#)

--help, -h *BOOLEAN*

Zeigt Befehlsreferenzinformationen für den `cdk doctor` Befehl an.

Beispiele

Einfaches Beispiel für den `cdk doctor` Befehl

```
$ cdk doctor  
## CDK Version: 1.0.0 (build e64993a)  
## AWS environment variables:  
- AWS_EC2_METADATA_DISABLED = 1  
- AWS_SDK_LOAD_CONFIG = 1
```

cdk import

Verwenden Sie [AWS CloudFormation Ressourcenimporte](#), um vorhandene AWS Ressourcen in einen CDK-Stack zu importieren.

Mit diesem Befehl können Sie vorhandene Ressourcen, die mit anderen Methoden erstellt wurden, verwenden und mit der AWS CDK Verwaltung beginnen.

Wenn Sie erwägen, Ressourcen in das CDK-Management zu verlagern, ist es manchmal akzeptabel, neue Ressourcen zu erstellen, z. B. mit IAM-Rollen, Lambda-Funktionen und Ereignisregeln. Bei anderen Ressourcen, wie z. B. statusbehafteten Ressourcen wie Amazon S3 S3-Buckets und DynamoDB-Tabellen, kann das Erstellen neuer Ressourcen Auswirkungen auf Ihren Service haben. Sie können sie verwenden `cdk import`, um vorhandene Ressourcen mit minimaler Unterbrechung

Ihrer Dienste zu importieren. Eine Liste der unterstützten AWS Ressourcen finden Sie im AWS CloudFormation Benutzerhandbuch unter [Unterstützung für Ressourcentypen](#).

Um eine vorhandene Ressource in einen CDK-Stack zu importieren

1. Führen Sie `cdk diff`, um sicherzustellen, dass Ihr CDK-Stack keine ausstehenden Änderungen enthält. Wenn Sie eine `cdk import` ausführen, sind die einzigen Änderungen, die bei einem Importvorgang zulässig sind, das Hinzufügen neuer Ressourcen, die importiert werden.
2. Fügen Sie Konstrukte für die Ressourcen hinzu, die Sie in Ihren Stack importieren möchten. Fügen Sie beispielsweise Folgendes für einen Amazon S3 S3-Bucket hinzu:

```
new s3.Bucket(this, 'ImportedS3Bucket', {});
```

Fügen Sie keine weiteren Änderungen hinzu. Sie müssen außerdem sicherstellen, dass der aktuelle Status der Ressource exakt modelliert wird. Achten Sie beim Bucket-Beispiel darauf, AWS KMS Schlüssel, Lebenszyklusrichtlinien und alles andere, was für den Bucket relevant ist, anzugeben. Andernfalls führen nachfolgende Aktualisierungsvorgänge möglicherweise nicht zu Ihren Erwartungen.

3. Führen Sie `cdk import`. Wenn die CDK-App mehrere Stacks enthält, übergeben Sie einen bestimmten Stacknamen als Argument.
4. Das CDK fordert CLI Sie auf, die tatsächlichen Namen der Ressourcen, die Sie importieren, einzugeben. Nachdem Sie diese Informationen eingegeben haben, beginnt der Import.
5. Bei `cdk import` erfolgreicher Meldung wird die Ressource vom CDK verwaltet. Alle nachfolgenden Änderungen an der Konstruktconfiguration werden auf der Ressource widergespiegelt.

Diese Funktion hat derzeit folgende Einschränkungen:

- Das Importieren von Ressourcen in verschachtelte Stacks ist nicht möglich.
- Es wird nicht überprüft, ob die von Ihnen angegebenen Eigenschaften für die importierte Ressource korrekt und vollständig sind. Versuchen Sie, nach dem Import einen Vorgang zur Erkennung von Abweichungen zu starten.
- Ressourcen, die von anderen Ressourcen abhängig sind, müssen alle zusammen oder einzeln in der richtigen Reihenfolge importiert werden. Andernfalls schlägt die CloudFormation Bereitstellung mit ungelösten Verweisen fehl.

- Dieser Befehl verwendet die Anmeldeinformationen für die Bereitstellungsrolle, die zum Lesen des verschlüsselten Staging-Buckets erforderlich sind. Dazu ist Version 12 der Bootstrap-Vorlage erforderlich, die die erforderlichen IAM-Berechtigungen für die Bereitstellungsrolle enthält.

Verwendung

```
$ cdk import <arguments> <options>
```

Argumente

Logische ID des CDK-Stacks

Die logische ID des CDK-Stacks aus Ihrer App, in den Ressourcen importiert werden sollen. Dieses Argument kann in einem einzigen Befehl mehrfach angegeben werden.

Typ: Zeichenfolge

Required: No

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter [Globale Optionen](#).

`--change-set-name` *STRING*

Der Name des zu CloudFormation erstellenden Änderungssatzes.

`--execute` *BOOLEAN*

Geben Sie an, ob der Änderungssatz ausgeführt werden soll.

Standardwert: `true`

`--force`, `-f` *BOOLEAN*

Standardmäßig CLI beendet das CDK den Prozess, wenn das Template-Diff Aktualisierungen oder Löschungen beinhaltet. Geben Sie `true` an, dass dieses Verhalten außer Kraft gesetzt und der Import immer fortgesetzt werden soll.

`--help, -h BOOLEAN`

Zeigt die Befehlsreferenzinformationen für den `cdk import` Befehl an.

`--record-resource-mapping, -r STRING`

Verwenden Sie diese Option, um eine Zuordnung vorhandener physischer Ressourcen zu den CDK-Ressourcen zu generieren, die importiert werden. Die Zuordnung wird in den von Ihnen angegebenen Dateipfad geschrieben. Es werden keine eigentlichen Importvorgänge ausgeführt.

`--resource-mapping, -m STRING`

Verwenden Sie diese Option, um eine Datei anzugeben, die Ihre Ressourcenzuweisung definiert. Das CDK verwendet CLI diese Datei, um physische Ressourcen Ressourcen für den Import zuzuordnen, anstatt Sie interaktiv danach zu fragen.

Diese Option kann von Skripten aus ausgeführt werden.

`--rollback BOOLEAN`

Setzen Sie den Stack bei einem Fehler in den stabilen Zustand zurück.

Um dies zu spezifizieren `false`, können Sie `--no-rollback` oder `verwenden-R`.

Geben Sie `false` an, ob die Iteration schneller erfolgen soll. Bereitstellungen, die Ressourcen ersetzen, schlagen immer fehl.

Standardwert: `true`

`--toolkit-stack-name STRING`

Der Name des CDK Toolkit-Stacks, der erstellt werden soll

cdk init

Erstellen Sie ein neues AWS CDK Projekt aus einer Vorlage.

Verwendung

```
$ cdk init <arguments> <options>
```

Argumente

Typ der Vorlage

Der CDK-Vorlagentyp, von dem aus ein neues CDK-Projekt initialisiert werden soll.

- `app`— Vorlage für eine CDK-Anwendung.
- `lib`— Vorlage für eine AWS Construct-Bibliothek.
- `sample-app`— CDK-Beispielanwendung, die einige Konstrukte enthält.

Gültige Werte: `app`, `lib` `sample-app`

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter [Globale Optionen](#).

`--generate-only` *BOOLEAN*

Geben Sie diese Option an, um Projektdateien zu generieren, ohne zusätzliche Operationen wie das Einrichten eines Git-Repositorys, das Installieren von Abhängigkeiten oder das Kompilieren des Projekts zu initiieren.

Standardwert: `false`

`--help`, `-h` *BOOLEAN*

Zeigt Befehlsreferenzinformationen für `an. cdk init command`

`--language`, `-l` *STRING*

Die Sprache, die für das neue Projekt verwendet werden soll. Diese Option kann in der `cdk.json` Konfigurationsdatei des Projekts oder `~/ .cdk.json` auf Ihrem lokalen Entwicklungscomputer konfiguriert werden.

Gültige Werte: `csharp`, `fsharp`, `go`, `java`, `javascript`, `python`, `typescript`

`--list` *BOOLEAN*

Listet die verfügbaren Vorlagentypen und Sprachen auf.

Beispiele

Listet die verfügbaren Vorlagentypen und Sprachen auf

```
$ cdk init --list
Available templates:
* app: Template for a CDK Application
  ## cdk init app --language=[csharp|fsharp|go|java|javascript|python|typescript]
* lib: Template for a CDK Construct Library
  ## cdk init lib --language=typescript
* sample-app: Example CDK Application with some constructs
  ## cdk init sample-app --language=[csharp|fsharp|go|java|javascript|python|
typescript]
```

Erstellen Sie eine neue CDK-App TypeScript aus der Bibliotheksvorlage

```
$ cdk init lib --language=typescript
```

cdk list

Listet alle AWS CDK Stacks und ihre Abhängigkeiten von einer CDK-App auf.

Verwendung

```
$ cdk list <arguments> <options>
```

Argumente

Logische ID des CDK-Stacks

Die logische ID des CDK-Stacks aus Ihrer App, für den dieser Befehl ausgeführt werden soll.

Typ: Zeichenfolge

Required: No

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter.

[Globale Optionen](#)

`--help, -h` *BOOLEAN*

Zeigt Befehlsreferenzinformationen für den `cdk list` Befehl an.

`--long, -l` *BOOLEAN*

Zeigt AWS Umgebungsinformationen für jeden Stapel an.

Standardwert: `false`

`--show-dependencies, -d` *BOOLEAN*

Zeigt Informationen zur Stapelabhängigkeit für jeden Stapel an.

Standardwert: `false`

Beispiele

Listet alle Stacks in der CDK-App 'node bin/main.js' auf

```
$ cdk list --app='node bin/main.js'
Foo
Bar
Baz
```

Listet alle Stacks auf, einschließlich der AWS Umgebungsdetails für jeden Stack

```
$ cdk list --app='node bin/main.js' --long
-
  name: Foo
  environment:
    name: 000000000000/bermuda-triangle-1
    account: '000000000000'
    region: bermuda-triangle-1
-
  name: Bar
  environment:
    name: 111111111111/bermuda-triangle-2
```



```
    account: '111111111111'  
    region: bermuda-triangle-2  
-  
  name: Baz  
  environment:  
    name: 333333333333/bermuda-triangle-3  
    account: '333333333333'  
    region: bermuda-triangle-3
```

cdk metadata

Zeigt Metadaten an, die einem CDK-Stack zugeordnet sind.

Verwendung

```
$ cdk metadata <arguments> <options>
```

Argumente

Logische ID des CDK-Stacks

Die logische ID des CDK-Stacks aus Ihrer App, für den Metadaten angezeigt werden sollen.

Typ: Zeichenfolge

Required: No

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter.

[Globale Optionen](#)

--help, -h **BOOLEAN**

Zeigt Befehlsreferenzinformationen für den cdk metadata Befehl an.

cdk migrate

Migrieren Sie bereitgestellte AWS Ressourcen, AWS CloudFormation Stacks und CloudFormation Vorlagen in ein neues AWS CDK Projekt.

Mit diesem Befehl wird eine neue CDK-App erstellt, die einen einzelnen Stapel enthält, der mit dem Wert benannt ist, den Sie angeben. `--stack-name` Sie können die Migrationsquelle mit `--from-scan`, `--from-stack`, oder `--from-path` konfigurieren.

Weitere Informationen zur Verwendung finden `cdk migrate` Sie unter [Migrieren Sie bestehende Ressourcen und AWS CloudFormation Vorlagen auf die AWS CDK](#).

Note

Der `cdk migrate` Befehl ist experimentell und könnte in future grundlegende Änderungen haben.

Verwendung

```
$ cdk migrate <options>
```

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter [Globale Optionen](#).

Erforderliche Optionen

`--stack-name` *STRING*

Der Name des AWS CloudFormation Stacks, der nach der Migration in der CDK-App erstellt wird.

Erforderlich: Ja

Bedingte Optionen

`--from-path` *PATH*

Der Pfad zur AWS CloudFormation Vorlage, die migriert werden soll. Geben Sie diese Option an, um eine lokale Vorlage anzugeben.

Erforderlich: Bedingt. Erforderlich, wenn von einer lokalen AWS CloudFormation Vorlage migriert wird.

--from-scan *STRING*

Wenn Sie bereitgestellte Ressourcen aus einer AWS Umgebung migrieren, verwenden Sie diese Option, um anzugeben, ob ein neuer Scan gestartet werden soll oder ob der letzte erfolgreiche Scan verwendet werden AWS CDK CLI soll.

Erforderlich: Bedingt. Erforderlich bei der Migration von bereitgestellten AWS Ressourcen.

Zulässige Werte: `most-recent`, `new`

--from-stack *BOOLEAN*

Stellen Sie diese Option bereit, um von einem bereitgestellten AWS CloudFormation Stack zu migrieren. Wird verwendet `--stack-name`, um den Namen des bereitgestellten AWS CloudFormation Stacks anzugeben.

Erforderlich: Bedingt. Erforderlich, wenn von einem bereitgestellten AWS CloudFormation Stack migriert wird.

Optionale Optionen

--account *STRING*

Das Konto, von dem die AWS CloudFormation Stack-Vorlage abgerufen werden soll.

Required: No

Standard: Das AWS CDK CLI ruft Kontoinformationen aus Standardquellen ab.

--compress *BOOLEAN*

Stellen Sie diese Option bereit, um das generierte CDK-Projekt in eine ZIP Datei zu komprimieren.

Required: No

--filter *ARRAY*

Wird verwendet, wenn bereitgestellte Ressourcen von einem AWS Konto migriert werden und. AWS-Region Diese Option gibt einen Filter an, mit dem bestimmt wird, welche bereitgestellten Ressourcen migriert werden sollen.

Diese Option akzeptiert ein Array von Schlüssel-Wert-Paaren, wobei `key` für den Filtertyp und `value` für den zu filternden Wert steht.

Die folgenden Schlüssel werden akzeptiert:

- `resource-identifizier`— Ein Bezeichner für die Ressource. Der Wert kann die logische oder physische ID der Ressource sein. z. B. `resource-identifizier="ClusterName"`.
- `resource-type-prefix`— Das Präfix AWS CloudFormation für den Ressourcentyp. Geben Sie beispielsweise `resource-type-prefix="AWS::DynamoDB::"` an, dass alle Amazon DynamoDB-Ressourcen gefiltert werden sollen.
- `tag-key`— Der Schlüssel eines Ressourcen-Tags. z. B. `tag-key="myTagKey"`.
- `tag-value`— Der Wert eines Ressourcen-Tags. z. B. `tag-value="myTagValue"`.

Stellen Sie mehrere Schlüssel-Wert-Paare für die AND bedingte Logik bereit. Das folgende Beispiel filtert nach jeder DynamoDB-Ressource, die mit `myTagKey` dem Tag-Schlüssel gekennzeichnet ist: `--filter resource-type-prefix="AWS::DynamoDB::", tag-key="myTagKey"`

Geben Sie die `--filter` Option für OR bedingte Logik mehrmals in einem einzigen Befehl an. Das folgende Beispiel filtert nach jeder Ressource, die eine DynamoDB-Ressource ist oder mit der `myTagKey` als Tag-Schlüssel gekennzeichnet ist: `--filter resource-type-prefix="AWS::DynamoDB::" --filter tag-key="myTagKey"`

Required: No

`--help, -h` *BOOLEAN*

Zeigt Befehlsreferenzinformationen für den `cdk migrate` Befehl an.

`--language` *STRING*

Die Programmiersprache, die für das CDK-Projekt verwendet werden soll, das während der Migration erstellt wurde.

Required: No

Gültige Werte: `typescript,python,,javacsharp,go`.

Standardwert: `typescript`

`--output-path` *PATH*

Der Ausgabepfad für das migrierte CDK-Projekt.

Required: No

Standard: Standardmäßig verwendet der AWS CDK CLI Ihr aktuelles Arbeitsverzeichnis.

`--region` *STRING*

Das AWS-Region, aus dem die AWS CloudFormation Stack-Vorlage abgerufen werden soll.

Required: No

Standard: Der AWS CDK CLI ruft AWS-Region Informationen aus Standardquellen ab.

Beispiele

Einfaches Beispiel für die Migration von einem Stack CloudFormation

Migrieren Sie von einem bereitgestellten CloudFormation Stack in einer bestimmten AWS Umgebung mithilfe von `--from-stack`. Geben Sie `--stack-name` einen Namen für Ihren neuen CDK-Stack an. Im Folgenden finden Sie ein Beispiel für die Migration `myCloudFormationStack` zu einer neuen CDK-App, die Folgendes verwendet: TypeScript

```
$ cdk migrate --language typescript --from-stack --stack-name 'myCloudFormationStack'
```

Einfaches Beispiel für die Migration von einer lokalen Vorlage CloudFormation

Migrieren Sie von einer lokalen JSON- oder CloudFormation YAML-Vorlage mit `--from-path`. Geben Sie `--stack-name` an, ob Sie Ihrem neuen CDK-Stack einen Namen geben möchten. Im Folgenden finden Sie ein Beispiel, das eine neue CDK-App erstellt TypeScript, die einen `myCloudFormationStack` Stack aus einer lokalen `template.json` Datei enthält:

```
$ cdk migrate --stack-name "myCloudFormationStack" --language typescript --from-path "./template.json"
```

Einfaches Beispiel für die Migration von bereitgestellten Ressourcen AWS

Migrieren Sie bereitgestellte AWS Ressourcen aus einer bestimmten AWS Umgebung, die keinem CloudFormation Stack zugeordnet sind, mithilfe von `--from-scan`. Das CDK CLI verwendet den IaC generator Dienst, um nach Ressourcen zu suchen und eine Vorlage zu generieren. Anschließend CLI verweist das CDK auf die Vorlage, um die neue CDK-App zu erstellen. Im Folgenden finden Sie ein Beispiel, das eine neue CDK-App TypeScript mit einem neuen `myCloudFormationStack` Stack erstellt, der migrierte Ressourcen enthält: AWS

```
$ cdk migrate --language typescript --from-scan --stack-name "myCloudFormationStack"
```

cdk notices

Zeigt Hinweise für Ihre CDK-Anwendung an.

Hinweise können wichtige Meldungen zu Sicherheitslücken, Regressionen und der Verwendung nicht unterstützter Versionen enthalten.

Dieser Befehl zeigt relevante Hinweise an, unabhängig davon, ob sie bestätigt wurden oder nicht. Relevante Hinweise können standardmäßig auch nach jedem Befehl erscheinen.

Sie können Hinweise auf folgende Weise unterdrücken:

- Über Befehlsoptionen. Im Folgenden wird ein Beispiel gezeigt:

```
$ cdk deploy --no-notices
```

- Unterdrücken Sie alle Hinweise auf unbestimmte Zeit durch den Kontext in der `cdk.json` Projektdatei:

```
{
  "notices": false,
  "context": {
    // ...
  }
}
```

- Bestätigen Sie jeden Hinweis mit dem `cdk acknowledge` Befehl.

Verwendung

```
$ cdk notices <options>
```

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter [Globale Optionen](#).

`--help`, `-h` *BOOLEAN*

Zeigt Befehlsreferenzinformationen für den `cdk notices` Befehl an.

Beispiele

Beispiel für einen Standardhinweis, der nach der Ausführung des Befehls `cdk deploy` angezeigt wird

```
$ cdk deploy
```

```
... # Normal output of the command
```

```
NOTICES
```

```
16603  Toggling off auto_delete_objects for Bucket empties the bucket
```

```
Overview: If a stack is deployed with an S3 bucket with
           auto_delete_objects=True, and then re-deployed with
           auto_delete_objects=False, all the objects in the bucket
           will be deleted.
```

```
Affected versions: <1.126.0.
```

```
More information at: https://github.com/aws/aws-cdk/issues/16603
```

```
17061  Error when building EKS cluster with monocdk import
```

```
Overview: When using monocdk/aws-eks to build a stack containing
           an EKS cluster, error is thrown about missing
           lambda-layer-node-proxy-agent/layer/package.json.
```

```
Affected versions: >=1.126.0 <=1.130.0.
```

```
More information at: https://github.com/aws/aws-cdk/issues/17061
```

```
If you don't want to see an notice anymore, use "cdk acknowledge ID". For example, "cdk
acknowledge 16603"
```

Einfaches Beispiel für die Ausführung des Befehls `cdk notices`

```
$ cdk notices
```

```
NOTICES
```

16603 Toggling off `auto_delete_objects` for Bucket empties the bucket

Overview: if a stack is deployed with an S3 bucket with `auto_delete_objects=True`, and then re-deployed with `auto_delete_objects=False`, all the objects in the bucket will be deleted.

Affected versions: framework: `<=2.15.0 >=2.10.0`

More information at: <https://github.com/aws/aws-cdk/issues/16603>

If you don't want to see a notice anymore, use `"cdk acknowledge <id>"`. For example, `"cdk acknowledge 16603"`

cdk synthesize

Synthetisieren Sie eine CDK-App, um eine Cloud-Assembly zu erstellen, einschließlich einer AWS CloudFormation Vorlage für jeden Stapel.

Cloud-Assemblies sind Dateien, die alles enthalten, was Sie für die Bereitstellung Ihrer App in Ihrer AWS Umgebung benötigen. Es enthält beispielsweise eine CloudFormation Vorlage für jeden Stapel in Ihrer App und eine Kopie der Dateiressourcen oder Docker Bilder, auf die Sie in Ihrer App verweisen.

Wenn Ihre App einen einzelnen Stapel enthält oder wenn ein einzelner Stapel als Argument angegeben wird, wird die CloudFormation Vorlage auch in der Standardausgabe (`stdout`) im YAML-Format angezeigt.

Wenn Ihre App mehrere Stapel enthält, `cdk synth` wird die Cloud-Assembly zu synthetisiert.
`cdk.out`

Verwendung

```
$ cdk synthesize <arguments> <options>
```


Argumente

Logische ID des CDK-Stacks

Die logische ID des CDK-Stacks aus Ihrer App, die synthetisiert werden soll.

Typ: Zeichenfolge

Required: No

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter.

[Globale Optionen](#)

`--exclusively, -e BOOLEAN`

Synthetisiert nur angeforderte Stacks, schließt keine Abhängigkeiten ein.

`--help, -h BOOLEAN`

Zeigt Befehlsreferenzinformationen für den `cdk synth` Befehl an.

`--quiet, -q BOOLEAN`

Geben Sie die CloudFormation Vorlage nicht an `stdout`.

Diese Option kann in der `cdk.json` Datei des CDK-Projekts konfiguriert werden. Im Folgenden wird ein Beispiel gezeigt:

```
{
  "quiet": true
}
```

Standardwert: `false`

`--validation BOOLEAN`

Validieren Sie die generierten CloudFormation Vorlagen nach der Synthese, indem Sie zusätzliche Prüfungen durchführen.

Sie können diese Option auch über das `validateOnSynth` Attribut oder die `CDK_VALIDATION` Umgebungsvariable konfigurieren.

Standardwert: `true`

Beispiele

Synthetisieren Sie die Cloud-Assembly für einen CDK-Stack mit logischer ID `MyStackName` und geben Sie die CloudFormation Vorlage auf `stdout` aus

```
$ cdk synth MyStackName
```

Synthetisieren Sie die Cloud-Assembly für alle Stacks in einer CDK-App und speichern Sie sie in `cdk.out`

```
$ cdk synth
```

Synthetisieren Sie die Cloud-Assembly für `MyStackName`, schließen Sie aber keine Abhängigkeiten ein

```
$ cdk synth MyStackName --exclusively
```

Synthetisieren Sie die Cloud-Assembly für `MyStackName`, geben Sie die CloudFormation Vorlage jedoch nicht auf `stdout` aus

```
$ cdk synth MyStackName --quiet
```

cdk watch

Überwachen Sie ein lokales AWS CDK Projekt kontinuierlich auf Änderungen bei der Durchführung von Implementierungen und Hotswaps.

Dieser Befehl ähnelt dem Befehl `cdk deploy`, außer dass er kontinuierliche Bereitstellungen und Hotswaps mit einem einzigen Befehl ausführen kann.

Dieser Befehl ist eine Abkürzung für `cdk deploy --watch`

Um eine `cdk watch` Sitzung zu beenden, unterbrechen Sie den Vorgang, indem Sie auf `Ctrl+C`.

Welche Dateien beobachtet werden, hängt von der "watch" Einstellung in Ihrer `cdk.json` Datei ab. Es hat zwei Unterschlüssel, "include" und "exclude", das akzeptiert eine einzelne Zeichenfolge oder ein Array von Zeichenketten. Jeder Eintrag wird als Pfad relativ zum Speicherort der `cdk.json` Datei interpretiert. Beides `*` und `**` werden akzeptiert.

Wenn Sie mit dem `cdk init` Befehl ein Projekt erstellen, ist das folgende Standardverhalten `cdk watch` in der `cdk.json` Datei Ihres Projekts konfiguriert:

- "include" ist auf gesetzt `**/*`, was alle Dateien und Verzeichnisse im Stammverzeichnis des Projekts einschließt.
- "exclude" ist optional, mit Ausnahme von Dateien und Ordnern, die bereits standardmäßig ignoriert wurden. Dies besteht aus Dateien und Verzeichnissen, die mit `.`, dem CDK-Ausgabeverzeichnis und dem `node_modules` Verzeichnis beginnen.

Die zu konfigurierende Mindesteinstellung `watch` ist `"watch": {}`.

Wenn entweder Ihr CDK-Code oder Ihr Anwendungscode vor der Bereitstellung einen Build-Schritt erfordert, `cdk watch` funktioniert er mit dem "build" Schlüssel in der `cdk.json` Datei.

Note

Dieser Befehl gilt als experimentell und könnte in future grundlegende Änderungen haben.

Die gleichen Einschränkungen von `cdk deploy --hotswap` gelten für `cdk watch`. Weitere Informationen finden Sie unter [cdk deploy --hotswap](#).

Verwendung

```
$ cdk watch <arguments> <options>
```

Argumente

Logische ID des CDK-Stacks

Die logische ID des CDK-Stacks aus Ihrer App, die Sie sich ansehen möchten.

Typ: Zeichenfolge

Required: No

Optionen

Eine Liste der globalen Optionen, die mit allen CLI CDK-Befehlen funktionieren, finden Sie unter.

[Globale Optionen](#)

`--build-exclude, -E ARRAY`

Das Asset mit der angegebenen ID darf nicht neu erstellt werden.

Diese Option kann in einem einzigen Befehl mehrfach angegeben werden.

Standardwert: []

`--change-set-name STRING`

Der Name des zu erstellenden CloudFormation Änderungssatzes.

`--concurrency NUMBER`

Stellen Sie mehrere Stacks parallel bereit und tauschen Sie sie per Hotswap aus, wobei die Abhängigkeiten zwischen den Stacks berücksichtigt werden. Verwenden Sie diese Option, um Bereitstellungen zu beschleunigen. Sie müssen immer noch andere CloudFormation AWS-Konto Ratenbegrenzungen berücksichtigen.

Geben Sie eine Zahl ein, um die maximale Anzahl gleichzeitiger Bereitstellungen anzugeben (sofern die Abhängigkeit dies zulässt).

Standardwert: 1

`--exclusively, -e BOOLEAN`

Stellen Sie nur angeforderte Stacks bereit und schließen Sie keine Abhängigkeiten ein.

`--force, -f BOOLEAN`

Stellen Sie immer Stacks bereit, auch wenn die Vorlagen identisch sind.

Standardwert: false

`--help, -h BOOLEAN`

Zeigt Befehlsreferenzinformationen für den `cdk watch` Befehl an.

--hotswap *BOOLEAN*

`cdk watch` Verwendet standardmäßig Hotswap-Bereitstellungen, wenn möglich, um Ihre Ressourcen zu aktualisieren. Das CDK CLI versucht, eine Hotswap-Bereitstellung durchzuführen, und greift nicht auf eine vollständige Bereitstellung zurück, wenn dies nicht erfolgreich ist. CloudFormation Alle erkannten Änderungen, die nicht über einen Hotswap aktualisiert werden können, werden ignoriert.

Standardwert: `true`

--hotswap-fallback *BOOLEAN*

`cdk watch` Versucht standardmäßig, Hotswap-Bereitstellungen durchzuführen, und ignoriert Änderungen, die Bereitstellungen erfordern. CloudFormation Stellen Sie sicher `--hotswap-fallback`, dass Sie auf eine vollständige Bereitstellung zurückgreifen und eine vollständige CloudFormation Bereitstellung durchführen können, falls die Hotswap-Bereitstellung nicht erfolgreich ist.

--logs *BOOLEAN*

`cdk watch` Überwacht standardmäßig alle CloudWatch Protokollgruppen in Ihrer Anwendung und streamt die Protokollereignisse lokal an `stdout`

Standardwert: `true`

--progress *STRING*

Konfigurieren Sie, wie das CDK den Bereitstellungsfortschritt CLI anzeigt.

- `bar`— Zeigt Ereignisse bei der Stack-Bereitstellung als Fortschrittsbalken an, zusammen mit den Ereignissen für die Ressource, die gerade bereitgestellt wird.
- `events`— Stellen Sie einen vollständigen Verlauf bereit, einschließlich aller CloudFormation Ereignisse.

Sie können diese Option auch in der `cdk.json` Projektdatei oder `~/.cdk.json` auf Ihrem lokalen Entwicklungscomputer konfigurieren:

```
{
  "progress": "events"
}
```

Zulässige Werte: `bar`, `events`

Standardwert: `bar`

`--rollback` *BOOLEAN*

Wenn während der Bereitstellung eine Ressource nicht erstellt oder aktualisiert werden kann, wird die Bereitstellung auf den letzten stabilen Status zurückgesetzt, bevor das CDK CLI zurückkehrt. Alle bis zu diesem Zeitpunkt vorgenommenen Änderungen werden rückgängig gemacht. Ressourcen, die erstellt wurden, werden gelöscht und vorgenommene Aktualisierungen werden rückgängig gemacht.

Verwenden Sie `--no-rollback` oder `-R`, um dieses Verhalten zu deaktivieren. Wenn eine Ressource nicht erstellt oder aktualisiert werden kann, behält das CDK CLI die bis zu diesem Zeitpunkt vorgenommenen Änderungen bei und kehrt zurück. Dies kann in Entwicklungsumgebungen hilfreich sein, in denen Sie schnell iterieren.

Note

Wann werden Bereitstellungen `false`, die zu einem Austausch von Ressourcen führen, immer fehlschlagen. Sie können diesen Wert nur für Bereitstellungen verwenden, die neue Ressourcen aktualisieren oder erstellen.

Standardwert: `true`

`--toolkit-stack-name` *STRING*

Der Name des vorhandenen CDK Toolkit-Stacks.

Diese Option wird nur für CDK-Apps verwendet, die Legacy-Synthese verwenden.

Beispiele

Beobachten Sie einen CDK-Stack mit logischer ID `DevelopmentStack` auf Änderungen

```
$ cdk watch DevelopmentStack
Detected change to 'lambda-code/index.js' (type: change). Triggering 'cdk deploy'
DevelopmentStack: deploying...

# DevelopmentStack
```

Konfigurieren Sie in einer `cdk.json`-Datei, welche Elemente aufgenommen und von der Überwachung auf Änderungen ausgeschlossen werden sollen

```
{
  "app": "mvn -e -q compile exec:java",
  "watch": {
    "include": "src/main/**",
    "exclude": "target/*"
  }
}
```

Erstellen Sie Java vor der Bereitstellung ein CDK-Projekt, indem Sie die Datei `cdk.json` konfigurieren

```
{
  "app": "mvn -e -q exec:java",
  "build": "mvn package",
  "watch": {
    "include": "src/main/**",
    "exclude": "target/*"
  }
}
```

AWS CDK Referenz

Dieser Abschnitt enthält Referenzinformationen für den AWS Cloud Development Kit (AWS CDK).

Themen

- [API-Referenz](#)
- [AWS CDK Versioning](#)

API-Referenz

Die [API-Referenz](#) enthält Informationen über die AWS Construct Library und andere APIs, die von der bereitgestellt werden AWS Cloud Development Kit (AWS CDK). Die meisten der AWS Konstruktbibliothek sind in einem einzigen Paket mit dem TypeScript Namen enthalten: `aws-cdk-lib`. Der tatsächliche Paketname variiert je nach Sprache. Für jede unterstützte Programmiersprache werden separate Versionen der API-Referenz bereitgestellt.

Die CDK-API-Referenz ist in Submodule organisiert. Es gibt ein oder mehrere Submodule für jede AWS-Service.

Jedes Submodul hat eine Übersicht, die Informationen zur Verwendung seiner APIs enthält. Die [S3](#)-Übersicht zeigt beispielsweise, wie die Standardverschlüsselung für einen Amazon Simple Storage Service (Amazon S3)-Bucket festgelegt wird.

AWS CDK Versioning

Dieses Thema enthält Referenzinformationen darüber, wie das Versioning AWS Cloud Development Kit (AWS CDK) verarbeitet.

Versionsnummern bestehen aus drei numerischen Versionsteilen: Hauptversion.Nebenversion.Patch und halten sich strikt an das [semantische Versioning](#)-Modell. Das bedeutet, dass grundlegende Änderungen an stabilen APIs auf Hauptversionen beschränkt sind.

Neben- und Patch-Versionen sind abwärtskompatibel. Der in einer früheren Version mit derselben Hauptversion geschriebene Code kann auf eine neuere Version innerhalb derselben Hauptversion aktualisiert werden. Es wird auch weiterhin erstellt und ausgeführt, was dieselbe Ausgabe erzeugt.

Themen

- [AWS CDKCLI Kompatibilität](#)
- [AWS Versioning der Konstruktbibliothek](#)
- [Stabilität der Sprachbindung](#)

AWS CDKCLI Kompatibilität

Die AWS CDK CLI ist immer mit Konstruktbibliotheken einer semantisch niedrigeren oder gleichen Versionsnummer kompatibel. Daher ist es immer sicher, das AWS CDK CLI innerhalb derselben Hauptversion zu aktualisieren.

Die AWS CDK CLI ist nicht immer mit Konstruktbibliotheken einer semantisch höheren Version kompatibel. Kompatibilität hängt davon ab, ob dieselbe Cloud-Assembly-Schemaversion von den beiden Komponenten eingesetzt wird. Das AWS CDK Framework generiert während der Synthetisierung eine Cloud-Baugruppe und die AWS CDK CLI verbraucht sie für die Bereitstellung. Das Schema, das das Format der Cloud-Baugruppe definiert, wird streng angegeben und versioniert.

AWS -Konstruktbibliotheken, die eine bestimmte Cloud-Assembly-Schemaversion verwenden, sind mit AWS CDK CLI Versionen kompatibel, die diese Schemaversion oder höher verwenden. Dies kann Releases des AWS CDK CLI enthalten, die älter als eine bestimmte Konstruktbibliotheksversion sind.

Wenn die für die Konstruktbibliothek erforderliche Cloud-Assembly-Version nicht mit der von unterstützten Version kompatibel ist AWS CDK CLI, erhalten Sie eine Fehlermeldung wie die folgende:

```
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.
```

Um diesen Fehler zu beheben, aktualisieren Sie auf AWS CDK CLI eine Version, die mit der erforderlichen Cloud-Assembly-Version kompatibel ist, oder auf die neueste verfügbare Version. Die Alternative (Herunterstufen der Konstruktbibliotheksmodule, die Ihre App verwendet) wird im Allgemeinen nicht empfohlen.

Note

Weitere Informationen zum Cloud-Assembly-Schema finden Sie unter [Cloud-Assembly-Versioning](#).

AWS Versioning der Konstruktbibliothek

Die Module in der AWS Construct Library durchlaufen verschiedene Phasen, während sie vom Konzept bis zur ausgereiften API entwickelt werden. Verschiedene Stufen bieten in nachfolgenden Versionen von unterschiedliche API-Stabilität AWS CDK.

APIs in der AWS CDK Hauptbibliothek, `aws-cdk-lib`, sind stabil und die Bibliothek ist vollständig semantisch versioniert. Dieses Paket enthält AWS CloudFormation (L1)-Konstrukte für alle - AWS Services und alle stabilen höherrangigen (L2- und L3) Module. (Er enthält auch die CDK-Kernklassen wie `App` und `Stack`). APIs werden erst mit der nächsten Hauptversion des CDK aus diesem Paket entfernt (obwohl sie möglicherweise veraltet sind). Keine einzelne API wird jemals grundlegende Änderungen haben. Wenn eine grundlegende Änderung erforderlich ist, wird eine völlig neue API hinzugefügt.

Neue APIs, die sich in der Entwicklung für einen Service befinden, der bereits in integriert ist, `aws-cdk-lib` werden mit einem `BetaN` Suffix identifiziert, wobei bei 1 N beginnt und mit jeder grundlegenden Änderung der neuen API erhöht wird. `BetaN` APIs werden nie entfernt, sondern sind nur veraltet, sodass Ihre vorhandene App weiterhin mit neueren Versionen von `aws-cdk-lib` funktioniert. Wenn die API als stabil eingestuft wird, wird eine neue API ohne das `BetaN` Suffix hinzugefügt.

Wenn High-Level-APIs (L2 oder L3) für einen - AWS Service entwickelt werden, der zuvor nur über L1-APIs verfügte, werden diese APIs zunächst in einem separaten Paket verteilt. Der Name eines solchen Pakets hat ein „Alpha“-Suffix und seine Version entspricht der ersten Version, mit der `aws-cdk-lib` es kompatibel ist, mit einer `-alpha` Unterversion. Wenn das Modul die beabsichtigten Anwendungsfälle unterstützt, werden seine APIs zu `aws-cdk-lib` hinzugefügt.

Stabilität der Sprachbindung

Im Laufe der Zeit fügen wir dem möglicherweise Unterstützung AWS CDK für zusätzliche Programmiersprachen hinzu. Obwohl die in allen Sprachen beschriebene API identisch ist, variiert die Art und Weise, wie die API ausgedrückt wird, je nach Sprache und kann sich ändern, wenn sich die Sprachunterstützung weiterentwickeln. Aus diesem Grund gelten Sprachbindungen für einen bestimmten Zeitraum als experimentell, bis sie als bereit für den Produktionseinsatz gelten.

Language	Stability
TypeScript	Stable
JavaScript	Stable

Language	Stability
Python	Stable
Java	Stable
C#/.NET	Stable
Go	Stable

AWS CDK Anleitungen

Dieser Abschnitt enthält Tutorials für die AWS Cloud Development Kit (AWS CDK).

Themen

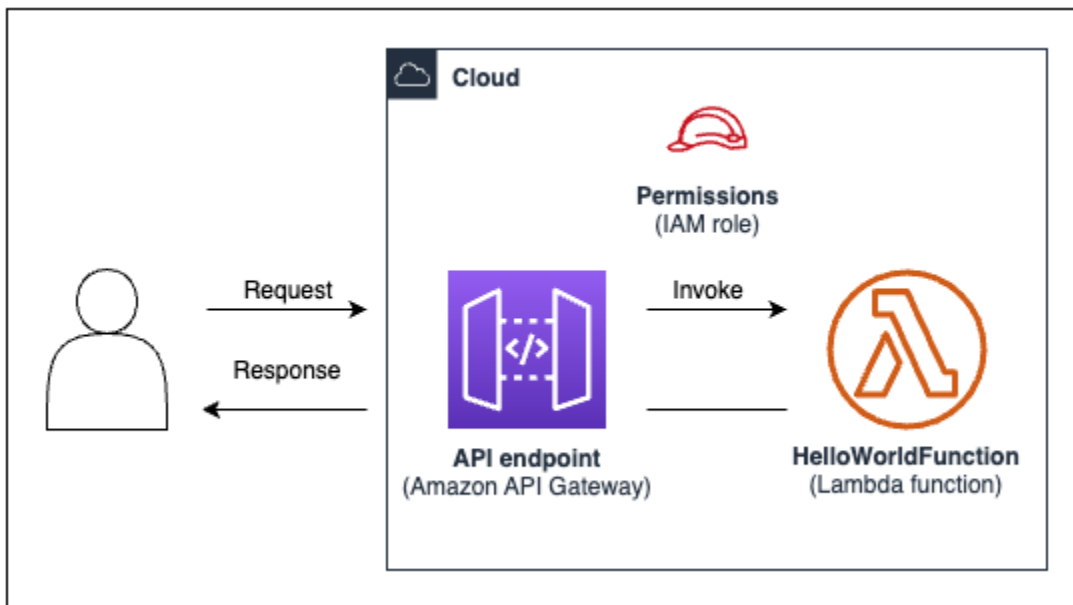
- [Eine serverlose Hello World-Anwendung erstellen](#)
- [Eine App mit mehreren Stacks erstellen](#)

Eine serverlose Hello World-Anwendung erstellen

In diesem Tutorial verwenden Sie die, um eine einfache serverlose Hello World Anwendung AWS Cloud Development Kit (AWS CDK) zu erstellen, die ein grundlegendes API-Backend implementiert, das aus den folgenden Komponenten besteht:

- Amazon API Gateway REST API — Stellt einen HTTP-Endpunkt bereit, der verwendet wird, um Ihre Funktion über eine HTTP GET Anfrage aufzurufen.
- AWS Lambda Funktion — Funktion, die eine Hello World! Nachricht zurückgibt, wenn sie mit dem HTTP Endpunkt aufgerufen wird.
- Integrationen und Berechtigungen — Konfigurationsdetails und Berechtigungen für Ihre Ressourcen, miteinander zu interagieren und Aktionen auszuführen, wie z. B. das Schreiben von Protokollen an Amazon CloudWatch.

Das folgende Diagramm zeigt die Komponenten dieser Anwendung:



In diesem Tutorial werden Sie Ihre Anwendung in den folgenden Schritten erstellen und mit ihr interagieren:

1. Erstellen Sie ein AWS CDK Projekt.
2. Definieren Sie eine Lambda-Funktion und eine API-Gateway-REST-API mithilfe von L2-Konstrukten aus der AWS Construct-Bibliothek.
3. Stellen Sie Ihre Anwendung auf dem bereit. AWS Cloud
4. Interagieren Sie mit Ihrer Anwendung in der AWS Cloud.
5. Löschen Sie die Beispielanwendung aus dem AWS Cloud.

Themen

- [Voraussetzungen](#)
- [Schritt 1: Erstellen Sie ein CDK-Projekt](#)
- [Schritt 2: Erstellen Sie Ihre Lambda-Funktion](#)
- [Schritt 3: Definieren Sie Ihre Konstrukte](#)
- [Schritt 4: Bereiten Sie Ihre Anwendung für die Bereitstellung vor](#)
- [Schritt 5: Ihre Anwendung bereitstellen](#)
- [Schritt 6: Interagieren Sie mit Ihrer Anwendung](#)
- [Schritt 7: Löschen Sie Ihre Anwendung](#)
- [Fehlerbehebung](#)

Voraussetzungen

Bevor Sie mit diesem Tutorial beginnen, müssen Sie folgende Aufgaben ausführen:

- Erstellen Sie ein AWS-Konto und lassen Sie das AWS Command Line Interface (AWS CLI) installieren und konfigurieren.
- Installieren Node.js und npm.
- Installieren Sie das CDK Toolkit global unter Verwendung von `npm install -g aws-cdk`

Weitere Informationen finden Sie unter [Erste Schritte mit der AWS CDK](#).

Wir empfehlen außerdem ein grundlegendes Verständnis der folgenden Themen:

- [Was ist der AWS CDK?](#) für eine grundlegende Einführung in die AWS CDK.
- [Lernen Sie die AWS CDK Kernkonzepte kennen](#) für einen Überblick über die Kernkonzepte der AWS CDK.

Schritt 1: Erstellen Sie ein CDK-Projekt

In diesem Schritt erstellen Sie mit dem Befehl ein neues CDK-Projekt. AWS CDK CLI `cdk init`

Um ein CDK-Projekt zu erstellen

1. Erstellen Sie von einem Startverzeichnis Ihrer Wahl aus ein Projektverzeichnis mit dem Namen `cdk-hello-world` auf Ihrem Computer und navigieren Sie zu diesem:

```
$ mkdir cdk-hello-world && cd cdk-hello-world
```

2. Verwenden Sie den `cdk init` Befehl, um ein neues Projekt in Ihrer bevorzugten Programmiersprache zu erstellen:

TypeScript

```
$ cdk init --language typescript
```

AWS CDK Bibliotheken installieren:

```
$ npm install aws-cdk-lib constructs
```

JavaScript

```
$ cdk init --language javascript
```

AWS CDK Bibliotheken installieren:

```
$ npm install aws-cdk-lib constructs
```

Python

```
$ cdk init --language python
```

Aktivieren Sie die virtuelle Umgebung:

```
$ source .venv/bin/activate # On Windows, run '.\venv\Scripts\activate' instead
```

AWS CDK Bibliotheken und Projektabhängigkeiten installieren:

```
(.venv)$ python3 -m pip install -r requirements.txt
```

Java

```
$ cdk init --language java
```

AWS CDK Bibliotheken und Projektabhängigkeiten installieren:

```
$ mvn package
```

C#

```
$ cdk init --language csharp
```

AWS CDK Bibliotheken und Projektabhängigkeiten installieren:

```
$ dotnet restore src
```

Go

```
$ cdk init --language go
```

Installieren Sie Projektabhängigkeiten:

```
$ go get github.com/aws/aws-cdk-go/awscdk/v2
$ go get github.com/aws/aws-cdk-go/awscdk/v2/awslambda
$ go get github.com/aws/aws-cdk-go/awscdk/v2/awsapigateway
$ go mod tidy
```

Das CDK CLI erstellt ein Projekt mit der folgenden Struktur:

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
### go.sum
```

Das CDK erstellt CLI automatisch eine CDK-App, die einen einzelnen Stack enthält. Die CDK-App-Instanz wird aus der Klasse erstellt. [App](#) Das Folgende ist ein Teil Ihrer CDK-Anwendungsdatei:

TypeScript

Befindet sich in `bin/cdk-hello-world.ts`:

```
#!/usr/bin/env node
import 'source-map-support/register';
import * as cdk from 'aws-cdk-lib';
import { CdkHelloWorldStack } from '../lib/cdk-hello-world-stack';
```

```
const app = new cdk.App();
new CdkHelloWorldStack(app, 'CdkHelloWorldStack', {
});
```

JavaScript

Befindet sich in `bin/cdk-hello-world.js`:

```
#!/usr/bin/env node
const cdk = require('aws-cdk-lib');
const { CdkHelloWorldStack } = require('../lib/cdk-hello-world-stack');
const app = new cdk.App();
new CdkHelloWorldStack(app, 'CdkHelloWorldStack', {
});
```

Python

Befindet sich in `app.py`:

```
#!/usr/bin/env python3
import os
import aws_cdk as cdk
from cdk_hello_world.cdk_hello_world_stack import CdkHelloWorldStack

app = cdk.App()
CdkHelloWorldStack(app, "CdkHelloWorldStack",)
app.synth()
```

Java

Befindet sich in `src/main/java/.../CdkHelloWorldApp.java`:

```
package com.myorg;

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

import java.util.Arrays;

public class JavaApp {
```

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

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

    app.synth();
}
}
```

C#

Befindet sich in `src/CdkHelloWorld/Program.cs`:

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

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

Go

Befindet sich in `incdk-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
}
```

Schritt 2: Erstellen Sie Ihre Lambda-Funktion

Erstellen Sie in Ihrem CDK-Projekt ein `lambda` Verzeichnis, das eine neue `hello.js` Datei enthält. Im Folgenden wird ein Beispiel gezeigt:

TypeScript

Führen Sie im Stammverzeichnis Ihres Projekts Folgendes aus:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Folgendes sollte jetzt zu Ihrem CDK-Projekt hinzugefügt werden:

```
cdk-hello-world
### lambda
### hello.js
```

JavaScript

Führen Sie im Stammverzeichnis Ihres Projekts Folgendes aus:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Folgendes sollte jetzt zu Ihrem CDK-Projekt hinzugefügt werden:

```
cdk-hello-world
### lambda
    ### hello.js
```

Python

Führen Sie im Stammverzeichnis Ihres Projekts Folgendes aus:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Folgendes sollte jetzt zu Ihrem CDK-Projekt hinzugefügt werden:

```
cdk-hello-world
### lambda
    ### hello.js
```

Java

Führen Sie im Stammverzeichnis Ihres Projekts Folgendes aus:

```
$ mkdir -p src/main/resources/lambda
$ cd src/main/resources/lambda
$ touch hello.js
```

Folgendes sollte jetzt zu Ihrem CDK-Projekt hinzugefügt werden:

```
cdk-hello-world
### src
    ### main
        ###resources
            ###lambda
                ###hello.js
```

C#

Führen Sie im Stammverzeichnis Ihres Projekts Folgendes aus:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Folgendes sollte jetzt zu Ihrem CDK-Projekt hinzugefügt werden:

```
cdk-hello-world
### lambda
    ### hello.js
```

Go

Führen Sie im Stammverzeichnis Ihres Projekts Folgendes aus:

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

Folgendes sollte jetzt zu Ihrem CDK-Projekt hinzugefügt werden:

```
cdk-hello-world
### lambda
    ### hello.js
```

Note

Um dieses Tutorial einfach zu halten, verwenden wir eine JavaScript Lambda-Funktion für alle CDK-Programmiersprachen.

Definieren Sie Ihre Lambda-Funktion, indem Sie der neu erstellten Datei Folgendes hinzufügen:

```
exports.handler = async (event) => {
  return {
    statusCode: 200,
    headers: { "Content-Type": "text/plain" },
    body: JSON.stringify({ message: "Hello, World!" }),
  };
};
```

```
};
```

Schritt 3: Definieren Sie Ihre Konstrukte

In diesem Schritt definieren Sie Ihre Lambda- und API-Gateway-Ressourcen mithilfe von AWS CDK L2-Konstrukten.

Öffnen Sie die Projektdatei, die Ihren CDK-Stack definiert. Sie werden diese Datei ändern, um Ihre Konstrukte zu definieren. Das Folgende ist ein Beispiel für Ihre Start-Stack-Datei:

TypeScript

Befindet sich in `lib/cdk-hello-world-stack.ts`:

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

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

    // Your constructs will go here
  }
}
```

JavaScript

Befindet sich in `lib/cdk-hello-world-stack.js`:

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

class CdkHelloWorldStack extends Stack {

  constructor(scope, id, props) {
    super(scope, id, props);

    // Your constructs will go here
  }
}
```



```
}  
  
module.exports = { CdkHelloWorldStack }
```

Python

Befindet sich in `incdk_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

Befindet sich in `insrc/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#

Befindet sich in `insrc/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

Befindet sich `incdk-hello-world.go`:

```
package main

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

type CdkHelloWorldStackProps struct {
    awscdk.StackProps
}

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

    // Your constructs will go here

    return stack
}
```

```
func main() {  
  
    // ...  
  
}  
  
func env() *awscdk.Environment {  
  
    return nil  
  
}
```

In dieser Datei AWS CDK macht der Folgendes:

- Ihre CDK-Stack-Instanz wird aus der Klasse instanziiert. [Stack](#)
- Die [Constructs](#) Basisklasse wird importiert und als Bereich oder übergeordnetes Objekt Ihrer Stack-Instance bereitgestellt.

Definieren Sie Ihre Lambda-Funktionsressource

Um Ihre Lambda-Funktionsressource zu definieren, importieren und verwenden Sie das [aws-lambda](#) L2-Konstrukt aus der AWS Construct-Bibliothek.

Ändern Sie Ihre Stack-Datei wie folgt:

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

Wir importieren das `aws_lambda` Modul `_lambda`, weil `lambda` es einen eingebauten Bezeichner enthält. 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"
    // Import S3 assets construct
    "github.com/aws/aws-cdk-go/awscdk/v2/awss3assets"
)

```

```
// ...
)

// ...

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"),
&awss3assets.AssetOptions{}), // Points to the lambda directory
    Handler: jsii.String("hello.handler"), // Points to the 'hello' file in the
lambda directory
    })

    return stack
}

// ...
```

Hier erstellen Sie eine Lambda-Funktionsressource und definieren die folgenden Eigenschaften:

- `runtime`— Die Umgebung, in der die Funktion ausgeführt wird. Hier verwenden wir Node.js Version 20.x.
- `code`— Der Pfad zum Funktionscode auf Ihrem lokalen Computer.
- `handler`— Der Name der spezifischen Datei, die Ihren Funktionscode enthält.

Definieren Sie Ihre REST API API-Gateway-Ressource

Um Ihre REST API API-Gateway-Ressource zu definieren, importieren und verwenden Sie das [aws-apigateway](#) L2-Konstrukt aus der AWS Construct-Bibliothek.

Ändern Sie Ihre Stack-Datei wie folgt:

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 Gateway 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"),
&awsapigateway.ResourceOptions{})
    helloResource.AddMethod(jsii.String("GET"),
awsapigateway.NewLambdaIntegration(helloWorldFunction,
&awsapigateway.LambdaIntegrationOptions{}), &awsapigateway.MethodOptions{})

    return stack
}

// ...
```

Hier erstellen Sie eine REST API API-Gateway-Ressource zusammen mit den folgenden Ressourcen:

- Eine Integration zwischen der REST API und Ihrer Lambda-Funktion, sodass die API Ihre Funktion aufrufen kann. Dies beinhaltet die Erstellung einer Lambda-Berechtigungsressource.
- Eine neue Ressource oder ein neuer Pfad mit dem Namen `hello`, der dem Stamm des API-Endpunkts hinzugefügt wird. Dadurch wird ein neuer Endpunkt erstellt, der Ihre Basis erweitert / `helloURL`.
- Eine GET-Methode für die `hello` Ressource. Wenn eine GET-Anfrage an den `/hello` Endpunkt gesendet wird, wird die Lambda-Funktion aufgerufen und ihre Antwort zurückgegeben.

Schritt 4: Bereiten Sie Ihre Anwendung für die Bereitstellung vor

In diesem Schritt bereiten Sie Ihre Anwendung für die Bereitstellung vor, indem Sie, falls erforderlich, eine grundlegende Validierung mit dem AWS CDK CLI `cdk synth` Befehl erstellen und durchführen.

Falls erforderlich, erstellen Sie Ihre Anwendung:

TypeScript

Führen Sie im Stammverzeichnis Ihres Projekts Folgendes aus:

```
$ npm run build
```

JavaScript

Ein Gebäude ist nicht erforderlich.

Python

Ein Gebäude ist nicht erforderlich.

Java

Führen Sie im Stammverzeichnis Ihres Projekts Folgendes aus:

```
$ mvn package
```

C#

Führen Sie im Stammverzeichnis Ihres Projekts Folgendes aus:

```
$ dotnet build src
```

Go

Ein Gebäude ist nicht erforderlich.

Führen Sie `cdk synth`, um eine AWS CloudFormation Vorlage aus Ihrem CDK-Code zu synthetisieren. Durch die Verwendung von L2-Konstrukten werden viele der Konfigurationsdetails, die benötigt werden AWS CloudFormation, um die Interaktion zwischen Ihrer Lambda-Funktion zu erleichtern, für Sie bereitgestellt. REST API AWS CDK

Führen Sie im Stammverzeichnis Ihres Projekts Folgendes aus:

```
$ cdk synth
```

Note

Wenn Sie eine Fehlermeldung wie die folgende erhalten, überprüfen Sie, ob Sie sich im `cdk-hello-world` Verzeichnis befinden, und versuchen Sie es erneut:

```
--app is required either in command-line, in cdk.json or in ~/.cdk.json
```

Bei Erfolg AWS CDK CLI wird die AWS CloudFormation Vorlage in der Befehlszeile im YAML Format ausgegeben. Eine JSON formatierte Vorlage wird ebenfalls im `cdk.out` Verzeichnis gespeichert.

Im Folgenden finden Sie ein Beispiel für die Ausgabe der AWS CloudFormation Vorlage:

AWS CloudFormation Vorlage

```
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: HelloWorldApiDeploymentStageprod*unique-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.
```

Mithilfe von L2-Konstrukten definieren Sie einige Eigenschaften, um Ihre Ressourcen zu konfigurieren, und verwenden Hilfsmethoden, um sie miteinander zu integrieren. Das AWS CDK konfiguriert den Großteil Ihrer AWS CloudFormation Ressourcen und Eigenschaften, die für die Bereitstellung Ihrer Anwendung erforderlich sind.

Schritt 5: Ihre Anwendung bereitstellen

In diesem Schritt verwenden Sie den AWS CDK CLI `cdk deploy` Befehl, um Ihre Anwendung bereitzustellen. Der AWS CDK arbeitet mit dem AWS CloudFormation Dienst zusammen, um Ihre Ressourcen bereitzustellen.

Important

Sie müssen vor der Bereitstellung ein einmaliges Bootstrapping Ihrer AWS Umgebung durchführen. Anweisungen finden Sie unter [Bootstrap your environment](#).

Führen Sie im Stammverzeichnis Ihres Projekts den folgenden Befehl aus. Bestätigen Sie Änderungen, wenn Sie dazu aufgefordert werden:

```
$ cdk deploy
```

```
# Synthesis time: 2.44s  
  
...  
  
Do you wish to deploy these changes (y/n)? y
```

Wenn die Bereitstellung abgeschlossen ist, AWS CDK CLI wird Ihre Endpunkt-URL ausgegeben. Kopieren Sie diese URL für den nächsten Schritt. Im Folgenden wird ein Beispiel gezeigt:

```
...  
# 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/<i>unique-identifier  
...
```

Schritt 6: Interagieren Sie mit Ihrer Anwendung

In diesem Schritt initiieren Sie eine GET-Anfrage an Ihren API-Endpunkt und erhalten Ihre Lambda-Funktionsantwort.

Suchen Sie Ihre Endpunkt-URL aus dem vorherigen Schritt und fügen Sie den `/hello` Pfad hinzu. Senden Sie dann über Ihren Browser oder die Befehlszeile eine GET-Anfrage an Ihren Endpunkt. Im Folgenden wird ein Beispiel gezeigt:

```
$ curl https://<api-id>.execute-api.<region>.amazonaws.com/prod/hello  
{"message":"Hello World!"}%
```

Herzlichen Glückwunsch! Sie haben Ihre Anwendung erfolgreich mit dem erstellt, bereitgestellt und mit ihr interagiert! AWS CDK

Schritt 7: Löschen Sie Ihre Anwendung

In diesem Schritt verwenden Sie die, AWS CDK CLI um Ihre Bewerbung aus dem zu löschen AWS Cloud.

Um Ihre Anwendung zu löschen, führen Sie den Befehl `cdk destroy`. Wenn Sie dazu aufgefordert werden, bestätigen Sie Ihre Anfrage zum Löschen der Anwendung:

```
$ cdk destroy
Are you sure you want to delete: CdkHelloWorldStack (y/n)? y
CdkHelloWorldStack: destroying... [1/1]
...
# CdkHelloWorldStack: destroyed
```

Fehlerbehebung

Fehler: {„Nachricht“: „Interner Serverfehler“}%

Beim Aufrufen der bereitgestellten Lambda-Funktion erhalten Sie diesen Fehler. Dieser Fehler kann aus mehreren Gründen auftreten.

Um weitere Fehler zu beheben

Verwenden Sie die AWS CLI , um Ihre Lambda-Funktion aufzurufen.

1. Ändern Sie Ihre Stack-Datei, um den Ausgabewert Ihres bereitgestellten Lambda-Funktionsnamens zu erfassen. Im Folgenden wird ein Beispiel gezeigt:

```
...

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. Stellen Sie Ihre Anwendung erneut bereit. Das AWS CDK CLI gibt den Wert Ihres bereitgestellten Lambda-Funktionsnamens aus:

```
$ cdk deploy

# Synthesis time: 0.29s
...
# CdkHelloWorldStack

# Deployment time: 20.36s

Outputs:
...
CdkHelloWorldStack.HelloWorldFunctionName = CdkHelloWorldStack-
HelloWorldFunctionunique-identifier
...
```

3. Verwenden Sie die AWS CLI , um Ihre Lambda-Funktion in aufzurufen AWS Cloud und die Antwort in eine Textdatei auszugeben:

```
$ aws lambda invoke --function-name CdkHelloWorldStack-HelloWorldFunctionunique-identifier output.txt
```

4. Überprüfen Sie `output.txt`, um Ihre Ergebnisse zu sehen.

Mögliche Ursache: Die API-Gateway-Ressource ist in Ihrer Stack-Datei falsch definiert.

Wenn eine erfolgreiche Lambda-Funktionsantwort `output.txt` angezeigt wird, liegt das Problem möglicherweise daran, wie Sie Ihre API-Gateway-REST-API definiert haben. Das AWS CLI ruft Ihr Lambda direkt auf, nicht über Ihren Endpunkt. Überprüfen Sie Ihren Code, um sicherzustellen, dass er mit diesem Tutorial übereinstimmt. Stellen Sie es dann erneut bereit.

Mögliche Ursache: Die Lambda-Ressource ist in Ihrer Stack-Datei falsch definiert.

Wenn ein Fehler `output.txt` zurückgegeben wird, liegt das Problem möglicherweise daran, wie Sie Ihre Lambda-Funktion definiert haben. Überprüfen Sie Ihren Code, um sicherzustellen, dass er mit diesem Tutorial übereinstimmt. Stellen Sie es dann erneut bereit.

Eine App mit mehreren Stacks erstellen

Sie können eine AWS Cloud Development Kit (AWS CDK) Anwendung erstellen, die mehrere [Stacks](#) enthält. Wenn Sie die AWS CDK App bereitstellen, wird jeder Stapel zu einer eigenen AWS

CloudFormation Vorlage. Mit dem AWS CDK CLI `cdk deploy` Befehl können Sie auch jeden Stack einzeln synthetisieren und bereitstellen.

Dieses Tutorial behandelt Folgendes:

- Wie man die Stack Klasse erweitert, um neue Eigenschaften oder Argumente zu akzeptieren.
- Wie man Eigenschaften verwendet, um festzustellen, welche Ressourcen der Stack enthält und wie sie konfiguriert sind.
- Wie instanziiert man mehrere Stacks aus dieser Klasse.

Das Beispiel in diesem Thema verwendet eine boolesche Eigenschaft mit dem Namen `encryptBucket` (Python:`encrypt_bucket`). Es gibt an, ob ein Amazon S3 S3-Bucket verschlüsselt werden soll. Wenn ja, ermöglicht der Stack die Verschlüsselung mithilfe eines Schlüssels, der von AWS Key Management Service (AWS KMS) verwaltet wird. Die App erstellt zwei Instanzen dieses Stacks, eine mit Verschlüsselung und eine ohne.

Themen

- [Bevor Sie beginnen](#)
- [Optionalen Parameter hinzufügen](#)
- [Definieren Sie die Stack-Klasse](#)
- [Erstellen Sie zwei Stack-Instanzen](#)
- [Synthetisieren Sie den Stack und stellen Sie ihn bereit](#)
- [Bereinigen](#)

Bevor Sie beginnen

Installieren Sie zunächst Node.js und die AWS CDK Befehlszeilentools, falls Sie dies noch nicht getan haben. Details dazu finden Sie unter [Erste Schritte mit der AWS CDK](#).

Erstellen Sie als Nächstes ein AWS CDK Projekt, indem Sie die folgenden Befehle in der Befehlszeile eingeben.

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 # On Windows, run '.\venv\Scripts\activate' instead
pip install -r requirements.txt
```

Java

```
mkdir multistack
cd multistack
cdk init --language=java
```

Sie können das resultierende Maven-Projekt in Ihre Java-IDE importieren.

C#

```
mkdir multistack
cd multistack
cdk init --language=csharp
```

Sie können die Datei `src/Pipeline.sln` in Visual Studio öffnen.

Optionalen Parameter hinzufügen

Das `props` Argument des `Stack` Konstruktors erfüllt die Schnittstelle `StackProps`. In diesem Beispiel möchten wir, dass der `Stack` eine zusätzliche Eigenschaft akzeptiert, die uns mitteilt, ob der Amazon S3 S3-Bucket verschlüsselt werden soll. Wir sollten eine Schnittstelle oder Klasse erstellen, die die Eigenschaft enthält. Auf diese Weise kann der Compiler sicherstellen, dass die Eigenschaft

einen booleschen Wert hat, und aktiviert die automatische Vervollständigung für diese Eigenschaft in Ihrer IDE.

Öffnen Sie also die angegebene Quelldatei in Ihrer IDE oder Ihrem Editor und fügen Sie die neue Schnittstelle, Klasse oder das neue Argument hinzu. Der Code sollte nach den Änderungen so aussehen. Die Zeilen, die wir hinzugefügt haben, sind fett gedruckt.

TypeScript

Datei: `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

Datei: `lib/multistack-stack.js`

JavaScript hat keine Schnittstellenfunktion; wir müssen keinen Code hinzufügen.

```
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

Datei: multistack/multistack_stack.py

Python hat keine Schnittstellenfunktion, daher erweitern wir unseren Stack so, dass er die neue Eigenschaft akzeptiert, indem wir ein Schlüsselwortargument hinzufügen.

```
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

Datei: src/main/java/com/myorg/MultistackStack.java

Es ist komplizierter, als wir wirklich wollen, einen Requisitentyp in Java zu erweitern. Schreiben Sie stattdessen den Konstruktor des Stacks so, dass er einen optionalen booleschen Parameter akzeptiert. Da props es sich um ein optionales Argument handelt, schreiben wir einen zusätzlichen Konstruktor, mit dem Sie ihn überspringen können. Standardmäßig ist es `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#

Datei: 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
        }
    }
}

```

Die neue Eigenschaft ist optional. Wenn `encryptBucket` (Python:`encrypt_bucket`) nicht vorhanden ist `undefined`, ist sein Wert oder das lokale Äquivalent. Der Bucket wird standardmäßig unverschlüsselt sein.

Definieren Sie die Stack-Klasse

Definieren wir nun unsere Stack-Klasse mithilfe unserer neuen Eigenschaft. Lass den Code wie folgt aussehen. Der Code, den Sie hinzufügen oder ändern müssen, ist fett gedruckt.

TypeScript

Datei: `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

Datei: `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

Datei: 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

Datei: 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#

Datei: 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
        });
    }
}
}
}
}

```

Erstellen Sie zwei Stack-Instanzen

Jetzt fügen wir den Code hinzu, um zwei separate Stapel zu instanziiieren. Nach wie vor müssen Sie die fett gedruckten Codezeilen hinzufügen. Löschen Sie die bestehende `MultistackStack` Definition.

TypeScript

Datei: `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

Datei: `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

Datei: ./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

Datei: 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#

Datei: 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();
    }
}
```

Dieser Code verwendet die neue `encryptBucket` (Python:`encrypt_bucket`) -Eigenschaft der `MultistackStack` Klasse, um Folgendes zu instanzieren:

- Ein Stapel mit einem verschlüsselten Amazon S3 S3-Bucket in der `us-east-1` AWS Region.
- Ein Stapel mit einem unverschlüsselten Amazon S3 S3-Bucket in der `us-west-1` AWS Region.

Synthetisieren Sie den Stack und stellen Sie ihn bereit

Jetzt können Sie Stacks über die App bereitstellen. Synthetisieren Sie zunächst eine AWS CloudFormation Vorlage für `MyEastCdkStack` — den Stack in. `us-east-1` Dies ist der Stack mit dem verschlüsselten S3-Bucket.

```
$ cdk synth MyEastCdkStack
```

Geben Sie einen der folgenden Befehle ein, um diesen Stack für Ihr AWS Konto bereitzustellen.

Der erste Befehl verwendet Ihr AWS Standardprofil, um die Anmeldeinformationen für die Bereitstellung des Stacks abzurufen. Der zweite verwendet ein Profil, das Sie angeben.

Ersetzen Sie `PROFILE_NAME` durch den Namen eines AWS CLI Profils, das die entsprechenden Anmeldeinformationen für die Bereitstellung in der `us-east-1` AWS Region enthält.

```
cdk deploy MyEastCdkStack
```

```
cdk deploy MyEastCdkStack --profile=PROFILE_NAME
```

Bereinigen

Um Gebühren für die von Ihnen bereitgestellten Ressourcen zu vermeiden, löschen Sie den Stack mit dem folgenden Befehl.

```
cdk destroy MyEastCdkStack
```

Der Löschvorgang schlägt fehl, wenn im Bucket des Stacks etwas gespeichert ist. Dies sollte nicht der Fall sein, wenn Sie nur die Anweisungen in diesem Thema befolgt haben. Wenn Sie jedoch etwas in den Bucket gelegt haben, müssen Sie den Inhalt des Buckets löschen, bevor Sie den Stapel zerstören. (Löschen Sie nicht den Bucket selbst.) Verwenden Sie das AWS Management Console oder das AWS CLI , um den Bucket-Inhalt zu löschen.

Beispiele

Dieses Thema enthält die folgenden Beispiele:

- [Eine serverlose Hello World-Anwendung erstellen](#)Erstellt eine serverlose Anwendung mit Lambda, API Gateway und Amazon S3.
- [Erstellen eines AWS Fargate-Dienstes mit dem AWS CDK](#)Erstellt anhand eines Images einen Amazon ECS Fargate-Service. DockerHub

Erstellen eines AWS Fargate-Dienstes mit dem AWS CDK

In diesem Beispiel erfahren Sie anhand eines Images auf Amazon ECR, wie Sie einen AWS Fargate-Service erstellen, der auf einem Amazon Elastic Container Service (Amazon ECS) -Cluster ausgeführt wird, dem ein mit dem Internet verbundener Application Load Balancer vorangestellt ist.

Amazon ECS ist ein hoch skalierbarer, schneller Container-Management-Service, der das Ausführen, Beenden und Verwalten von Docker-Containern in einem Cluster vereinfacht. Sie können Ihren Cluster auf einer serverlosen Infrastruktur hosten, die von Amazon ECS verwaltet wird, indem Sie Ihre Dienste oder Aufgaben mit dem Starttyp Fargate starten. Für mehr Kontrolle können Sie Ihre Aufgaben auf einem Cluster von Amazon Elastic Compute Cloud (Amazon EC2) -Instances hosten, die Sie mithilfe des Amazon EC2-Starttyps verwalten.

Dieses Tutorial zeigt Ihnen, wie Sie einige Dienste mit dem Fargate-Starttyp starten. Wenn Sie den verwendet haben, AWS Management Console um einen Fargate-Dienst zu erstellen, wissen Sie, dass Sie viele Schritte ausführen müssen, um diese Aufgabe zu erledigen. AWS enthält mehrere Tutorials und Dokumentationsthemen, die Sie durch die Erstellung eines Fargate-Dienstes führen, darunter:

- [So stellen Sie Docker-Container bereit - AWS](#)
- [Einrichtung mit Amazon ECS](#)
- [Erste Schritte mit Amazon ECS mithilfe von Fargate](#)

In diesem Beispiel wird ein ähnlicher Fargate-Dienst im AWS CDK Code erstellt.

Das in diesem Tutorial verwendete Amazon ECS-Konstrukt hilft Ihnen bei der Nutzung von AWS Services, indem es die folgenden Vorteile bietet:

- Konfiguriert automatisch einen Load Balancer.
- Öffnet automatisch eine Sicherheitsgruppe für Load Balancer. Dadurch können Load Balancer mit Instances kommunizieren, ohne dass Sie explizit eine Sicherheitsgruppe erstellen müssen.
- Ordnet automatisch die Abhängigkeit zwischen dem Service und dem Load Balancer an, der einer Zielgruppe zugeordnet ist. Dabei wird die korrekte Reihenfolge bei der Erstellung des Listeners AWS CDK erzwungen, bevor eine Instanz erstellt wird.
- Konfiguriert automatisch Benutzerdaten für automatisch skalierende Gruppen. Dadurch wird die richtige Konfiguration für die Zuordnung eines Clusters zu AMIs erstellt.
- Validiert Parameterkombinationen frühzeitig. Dadurch werden AWS CloudFormation Probleme früher aufgedeckt, sodass Sie Zeit bei der Bereitstellung sparen. Je nach Aufgabe ist es beispielsweise leicht, die Speichereinstellungen falsch zu konfigurieren. Bisher trat kein Fehler auf, bis Sie Ihre App bereitgestellt hatten. Aber jetzt AWS CDK kann sie eine Fehlkonfiguration erkennen und einen Fehler ausgeben, wenn Sie Ihre App synthetisieren.
- Fügt automatisch Berechtigungen für Amazon Elastic Container Registry (Amazon ECR) hinzu, wenn Sie ein Bild von Amazon ECR verwenden.
- Skaliert automatisch. Das AWS CDK stellt eine Methode bereit, mit der Sie Instances automatisch skalieren können, wenn Sie einen Amazon EC2 EC2-Cluster verwenden. Dies geschieht automatisch, wenn Sie eine Instanz in einem Fargate-Cluster verwenden.

Darüber hinaus AWS CDK verhindert das, dass eine Instanz gelöscht wird, wenn die automatische Skalierung versucht, eine Instanz zu beenden, aber entweder eine Aufgabe auf dieser Instanz läuft oder geplant ist.

Bisher mussten Sie eine Lambda-Funktion erstellen, um diese Funktionalität nutzen zu können.

- Bietet Asset-Support, sodass Sie in einem Schritt eine Quelle von Ihrem Computer in Amazon ECS bereitstellen können. Bisher mussten Sie zur Verwendung einer Anwendungsquelle mehrere manuelle Schritte ausführen, z. B. das Hochladen auf Amazon ECR und das Erstellen eines Docker-Images.

Einzelheiten finden Sie unter [ECS](#).

Important

Die `ApplicationLoadBalancedFargateService` Konstrukte, die wir verwenden werden, umfassen zahlreiche AWS Komponenten, von denen einige mit nicht unerheblichen Kosten verbunden sind, wenn sie in Ihrem AWS Konto bereitgestellt werden, auch wenn Sie sie

nicht verwenden. Stellen Sie sicher, dass Sie nach Abschluss dieses Beispiels clean up (cdk destroy) verwenden.

Das Verzeichnis erstellen und das initialisieren AWS CDK

Beginnen wir damit, ein Verzeichnis für den AWS CDK Code zu erstellen und dann eine AWS CDK App in diesem Verzeichnis zu erstellen.

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 # On Windows, run '.\venv\Scripts\activate' instead
pip install -r requirements.txt
```

Java

```
mkdir MyEcsConstruct
cd MyEcsConstruct
cdk init --language java
```

Sie können das Maven-Projekt jetzt in Ihre IDE importieren.

C#

```
mkdir MyEcsConstruct
```



```
cd MyEcsConstruct
cdk init --language csharp
```

Sie können es jetzt `src/MyEcsConstruct.sln` in Visual Studio öffnen.

Führen Sie die App aus und vergewissern Sie sich, dass sie einen leeren Stapel erstellt.

```
cdk synth
```

Einen Fargate-Dienst erstellen

Es gibt zwei verschiedene Möglichkeiten, Ihre Container-Aufgaben mit Amazon ECS auszuführen:

- Verwenden Sie den `Fargate` Starttyp, bei dem Amazon ECS die physischen Maschinen, auf denen Ihre Container laufen, für Sie verwaltet.
- Verwenden Sie den `EC2` Starttyp, bei dem Sie die Verwaltung übernehmen, z. B. die Angabe der automatischen Skalierung.

In diesem Beispiel erstellen wir einen Fargate-Dienst, der auf einem ECS-Cluster ausgeführt wird, dem ein mit dem Internet verbundener Application Load Balancer vorangestellt ist.

Fügen Sie der angegebenen Datei die folgenden Importe des AWS Construct Library-Moduls hinzu.

TypeScript

Datei: `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

Datei: `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

Datei: `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

Datei: `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#

Datei: `src/MyEcsConstruct/MyEcsConstructStack.cs`

```
using Amazon.CDK.AWS.EC2;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECS.Patterns;
```

Ersetzen Sie den Kommentar am Ende des Konstruktors durch den folgenden Code.

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
    }
);
```

Speichern Sie es und stellen Sie sicher, dass es ausgeführt wird und einen Stack erstellt.

```
cdk synth
```

Der Stapel besteht aus Hunderten von Zeilen, daher werden wir ihn hier nicht zeigen. Der Stack sollte eine Standardinstanz, ein privates Subnetz und ein öffentliches Subnetz für die drei Availability Zones sowie eine Sicherheitsgruppe enthalten.

Stellen Sie den Stack bereit.

```
cdk deploy
```

AWS CloudFormation zeigt Informationen zu den Dutzenden von Schritten an, die bei der Bereitstellung Ihrer App erforderlich sind.

So einfach ist es, einen von Fargate betriebenen Amazon ECS-Service zum Ausführen eines Docker-Images zu erstellen.

Bereinigen

Um unerwartete AWS Gebühren zu vermeiden, vernichten Sie Ihren AWS CDK Stack, nachdem Sie mit dieser Übung fertig sind.

```
cdk destroy
```

AWS CDK Beispiele

Weitere Beispiele für AWS CDK Stacks und Apps in Ihrer bevorzugten unterstützten Programmiersprache finden Sie im [AWS CDK Beispiel-Repository](#) unter GitHub.

AWS CDK Werkzeuge

Dieser Abschnitt enthält Informationen zu den unten aufgeführten AWS CDK Tools.

Themen

- [AWS CDK Toolkit \(cdkBefehl\)](#)
- [AWS Toolkit for Visual Studio Code](#)
- [AWS SAM -Integration](#)

AWS CDK Toolkit (**cdkBefehl**)

Das AWS CDK Toolkit, der CLI-Befehl `cdk`, ist das wichtigste Tool für die Interaktion mit Ihrer AWS CDK App. Es führt Ihre App aus, fragt das von Ihnen definierte Anwendungsmodell ab und erstellt und stellt die von der AWS CloudFormation generierten Vorlagen bereit. AWS CDK Es bietet auch andere Funktionen, die für die Erstellung und Arbeit mit Projekten nützlich sind. AWS CDK Dieses Thema enthält Informationen zu häufigen Anwendungsfällen des CDK Toolkit.

Das AWS CDK Toolkit wird mit dem Node Package Manager installiert. In den meisten Fällen empfehlen wir, es global zu installieren.

```
npm install -g aws-cdk           # install latest version
npm install -g aws-cdk@X.YY.Z   # install specific version
```

Tip

Wenn Sie regelmäßig mit mehreren Versionen von arbeiten, sollten Sie erwägen AWS CDK, eine passende Version des AWS CDK Toolkits in einzelnen CDK-Projekten zu installieren. Um dies zu tun, lassen Sie den Befehl `-g` aus. `npm install` Verwenden Sie es dann `npmx aws-cdk`, um es aufzurufen. Dadurch wird die lokale Version ausgeführt, falls eine existiert, und falls nicht, wird auf eine globale Version zurückgegriffen.

Toolkit-Befehle

Alle CDK Toolkit-Befehle beginnen mit `cdk`, gefolgt von einem Unterbefehl (`list`, `synthesizedeploy`, usw.). Einige Unterbefehle haben eine kürzere Version (`ls`, usw.) `synth`, die gleichwertig ist. Optionen und Argumente folgen dem Unterbefehl in beliebiger Reihenfolge.

Eine Beschreibung aller Unterbefehle, Optionen und Argumente finden Sie unter [AWS CDKCLI Befehlsreferenz](#)

Optionen und ihre Werte angeben

Befehlszeilenoptionen beginnen mit zwei Bindestrichen (`--`). Einige häufig verwendete Optionen haben Synonyme aus einem Buchstaben, die mit einem einzelnen Bindestrich beginnen (z. B. `--app` hat ein Synonym `-a`). Die Reihenfolge der Optionen in einem AWS CDK Toolkit-Befehl ist nicht wichtig.

Alle Optionen akzeptieren einen Wert, der dem Optionsnamen folgen muss. Der Wert kann durch Leerzeichen oder ein Gleichheitszeichen `=` vom Namen getrennt werden. Die folgenden beiden Optionen sind gleichwertig.

```
--toolkit-stack-name MyBootstrapStack
--toolkit-stack-name=MyBootstrapStack
```

Einige Optionen sind Flags (Boolesche Werte). Sie können `true` oder `false` als ihren Wert angeben. Wenn Sie keinen Wert angeben, wird der Wert als `true` angenommen. Sie können dem Optionsnamen auch implizit ein Präfix voranstellen `false`. `no-`

```
# sets staging flag to true
--staging
--staging=true
--staging true

# sets staging flag to false
--no-staging
--staging=false
--staging false
```

Einige Optionen, nämlich `--context`, `--parameters`, `--plugin`, und `--tags`, `--trust`, können mehrfach angegeben werden, um mehrere Werte anzugeben. Es wird darauf hingewiesen, dass sie in der CDK Toolkit-Hilfe `[array]` eingegeben werden müssen. Beispielsweise:

```
cdk bootstrap --tags costCenter=0123 --tags responsibleParty=jdoe
```

Integrierte Hilfe

Das AWS CDK Toolkit hat eine integrierte Hilfe. Allgemeine Hilfe zu dem Programm und eine Liste der bereitgestellten Unterbefehle finden Sie, indem Sie Folgendes eingeben:

```
cdk --help
```

Um beispielsweise Hilfe für einen bestimmten Unterbefehl zu erhalten `deploy`, geben Sie ihn vor dem `--help` Flag an.

```
cdk deploy --help
```

Problem `cdk version` beim Anzeigen der Version des AWS CDK Toolkits. Geben Sie diese Informationen an, wenn Sie Support anfordern.

Versionsberichterstattung

Um einen Einblick in die AWS CDK Verwendung von zu gewinnen, werden die von AWS CDK Anwendungen verwendeten Konstrukte mithilfe einer Ressource gesammelt und gemeldet, die als `AWS::CDK::Metadata` identifiziert wurde. Diese Ressource wird zu AWS CloudFormation Vorlagen hinzugefügt und kann leicht überprüft werden. Diese Informationen können auch verwendet werden, AWS um Stacks anhand eines Konstrukts mit bekannten Sicherheits- oder Zuverlässigkeitsproblemen zu identifizieren. Es kann auch verwendet werden, um ihre Benutzer mit wichtigen Informationen zu kontaktieren.

Note

Vor Version 1.93.0 AWS CDK meldeten sie die Namen und Versionen der Module, die während der Synthese geladen wurden, und nicht die im Stack verwendeten Konstrukte.

Standardmäßig AWS CDK meldet der die Verwendung von Konstrukten in den folgenden NPM-Modulen, die im Stack verwendet werden:

- AWS CDK Kernmodul
- AWS Konstruieren Sie Bibliotheksmodule

- AWS Modul Solutions Constructs
- AWS Modul „Render Farm Deployment Kit“

Die `AWS::CDK::Metadata` Ressource sieht in etwa wie folgt aus.

```
CDKMetadata:
  Type: "AWS::CDK::Metadata"
  Properties:
    Analytics:
      "v2:deflate64:H4sIAND9SGAAAzXKS5AMBAA0L1b2PdzBYnEAdio3RglgLY60zQi7u6TWL/
XKmNULxeQS0KwaPTBqrNhwEWU3hGHICzK0dWwfAxoL/Fd8mvk+QkS/0X6BdjnCdgm00QKwz
+AqQLDt2Y3YMnLYWwAAAA="
```

Die `Analytics` Eigenschaft ist eine gzip-komprimierte, Base64-kodierte, mit Präfix kodierte Liste der Konstrukte im Stapel.

Verwenden Sie eine der folgenden Methoden, um die Versionsberichterstattung zu deaktivieren:

- Verwenden Sie den `cdk` Befehl mit dem `--no-version-reporting` Argument, um sich von einem einzelnen Befehl abzumelden.

```
cdk --no-version-reporting synth
```

Denken Sie daran, dass das AWS CDK Toolkit vor der Bereitstellung neue Vorlagen synthetisiert. Sie sollten daher auch weitere Befehle `--no-version-reporting` hinzufügen `cdk deploy`.

- In oder `versionReporting` auf „Falsch“ setzen. `./cdk.json ~/cdk.json` Dies wird deaktiviert, es sei denn, Sie entscheiden sich dafür, indem Sie einen einzelnen Befehl angeben `--version-reporting`.

```
{
  "app": "...",
  "versionReporting": false
}
```

Authentifizierung mit AWS

Je nach Umgebung und verfügbarem Zugriff gibt es verschiedene Möglichkeiten, den programmatischen AWS Zugriff auf AWS Ressourcen zu konfigurieren.

Informationen zur Auswahl Ihrer Authentifizierungsmethode und deren Konfiguration für das CDK Toolkit finden Sie unter [Authentifizierung und Zugriff](#) im Referenzhandbuch für AWS SDKs und Tools.

Der empfohlene Ansatz für neue Benutzer, die sich lokal entwickeln und von ihrem Arbeitgeber keine Authentifizierungsmethode erhalten, ist die Einrichtung AWS IAM Identity Center. Diese Methode beinhaltet die Installation von AWS CLI um die Konfiguration zu vereinfachen und sich regelmäßig beim AWS Zugangportal anzumelden. Wenn Sie sich für diese Methode entscheiden, sollte Ihre Umgebung die folgenden Elemente enthalten, nachdem Sie das Verfahren zur [IAM Identity Center-Authentifizierung](#) im Referenzhandbuch für AWS SDKs und Tools abgeschlossen haben:

- Die AWS CLI, mit der Sie eine AWS Access-Portal-Sitzung starten, bevor Sie Ihre Anwendung ausführen.
- Eine [gemeinsam genutzte AWSconfig Datei](#) mit einem [default] Profil mit einer Reihe von Konfigurationswerten, auf die von der aus verwiesen werden kann AWS CDK. Den Speicherort dieser Datei finden Sie unter [Speicherort der freigegebenen Dateien](#) im Referenzhandbuch für AWS SDKs und Tools.
- Die gemeinsam genutzte config Datei legt die [region](#) Einstellung fest. Dies legt die Standardeinstellung fest, AWS-Region die das AWS CDK und das CDK Toolkit für AWS Anfragen verwenden.
- Das CDK Toolkit verwendet die [SSO-Token-Provider-Konfiguration](#) des Profils, um Anmeldeinformationen abzurufen, bevor Anfragen an gesendet werden. AWS Der `sso_role_name` Wert, bei dem es sich um eine IAM-Rolle handelt, die mit einem IAM Identity Center-Berechtigungssatz verbunden ist, sollte den Zugriff auf die in Ihrer AWS-Services Anwendung verwendeten Rollen ermöglichen.

Die folgende config Beispieldatei zeigt ein Standardprofil, das mit der Konfiguration des SSO-Token-Anbieters eingerichtet wurde. Die `sso_session` Einstellung des Profils bezieht sich auf den benannten [sso-sessionAbschnitt](#). Der `sso-session` Abschnitt enthält Einstellungen zum Initiieren einer AWS Access-Portal-Sitzung.

```
[default]
sso_session = my-ss0
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-ss0]
```

```
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
sso_registration_scopes = sso:account:access
```

Starten Sie eine AWS Access-Portal-Sitzung

Vor dem Zugriff benötigen Sie eine aktive AWS Access-Portal-Sitzung AWS-Services, damit das CDK Toolkit die IAM Identity Center-Authentifizierung zur Auflösung von Anmeldeinformationen verwenden kann. Abhängig von Ihrer konfigurierten Sitzungsdauer läuft Ihr Zugriff irgendwann ab und im CDK Toolkit wird ein Authentifizierungsfehler auftreten. Führen Sie den folgenden Befehl im aus AWS CLI , um sich beim AWS Zugriffsportal anzumelden.

```
aws sso login
```

Wenn Ihre SSO-Token-Provider-Konfiguration ein benanntes Profil anstelle des Standardprofils verwendet, lautet der Befehl `aws sso login --profile NAME`. Geben Sie dieses Profil auch an, wenn Sie `cdk` Befehle mit der `--profile` Option oder der `AWS_PROFILE` Umgebungsvariablen ausgeben.

Führen Sie den folgenden AWS CLI Befehl aus, um zu testen, ob Sie bereits eine aktive Sitzung haben.

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

In der Antwort auf diesen Befehl sollten das in der freigegebenen `config`-Datei konfigurierte IAM-Identity-Center-Konto und der Berechtigungssatz angegeben werden.

Note

Wenn Sie bereits über eine aktive AWS Access-Portal-Sitzung verfügen und diese ausführen `aws sso login`, müssen Sie keine Anmeldeinformationen angeben. Beim Anmeldevorgang werden Sie möglicherweise aufgefordert, den AWS CLI Zugriff auf Ihre Daten zu gewähren. Da AWS CLI das auf dem SDK für Python aufbaut, können Berechtigungsnachrichten Variationen des `botocore` Namens enthalten.

Angabe der Region und anderer Konfigurationen

Das CDK Toolkit muss die AWS Region kennen, in der Sie die Bereitstellung durchführen, und darüber, wie Sie sich authentifizieren. AWS Dies wird für Bereitstellungsvorgänge und zum Abrufen von Kontextwerten während der Synthese benötigt. Ihr Konto und Ihre Region bilden zusammen die Umgebung.

Die Region kann mithilfe von Umgebungsvariablen oder in Konfigurationsdateien angegeben werden. Dies sind dieselben Variablen und Dateien, die von anderen AWS Tools wie den AWS CLI und den verschiedenen AWS SDKs verwendet werden. Das CDK Toolkit sucht in der folgenden Reihenfolge nach diesen Informationen.

- Die `AWS_DEFAULT_REGION` Umgebungsvariable.
- Ein benanntes Profil, das in der AWS `config` Standarddatei definiert und mit der `--profile` Option für `cdk` Befehle angegeben wurde.
- Der `[default]` Abschnitt der AWS `config` Standarddatei.

Neben der Angabe der AWS Authentifizierung und einer Region im `[default]` Abschnitt können Sie auch einen oder mehrere `[profile NAME]` Abschnitte hinzufügen, wobei *NAME* der Name des Profils ist. Weitere Informationen zu benannten Profilen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im Referenzhandbuch für AWS SDKs und Tools.

Die AWS `config` Standarddatei befindet sich unter `~/.aws/config` (MacOS/Linux) oder `%USERPROFILE%\aws\config` (Windows). Einzelheiten und alternative Speicherorte finden Sie im Referenzhandbuch für [AWS SDKs und Tools unter Speicherort der gemeinsam genutzten Konfigurations- und Anmeldeinformationsdateien](#)

Die Umgebung, die Sie in Ihrer AWS CDK App mithilfe der `env` Stack-Eigenschaft angeben, wird bei der Synthese verwendet. Es wird verwendet, um eine umgebungsspezifische AWS CloudFormation Vorlage zu generieren, und während der Bereitstellung überschreibt es das Konto oder die Region, das mit einer der vorherigen Methoden angegeben wurde. Weitere Informationen finden Sie unter [the section called "Umgebungen"](#).

Note

Das AWS CDK verwendet Anmeldeinformationen aus denselben Quelldateien wie andere AWS Tools und SDKs, einschließlich der. [AWS Command Line Interface](#) Sie verhalten sich

jedoch AWS CDK möglicherweise etwas anders als diese Tools. Es nutzt das, was AWS SDK for JavaScript unter der Haube steckt. Vollständige Informationen zum Einrichten von Anmeldeinformationen für finden Sie unter [Anmeldeinformationen einrichten](#). AWS SDK for JavaScript

Sie können optional die Option `--role-arn` (oder `-r`) verwenden, um den ARN einer IAM-Rolle anzugeben, die für die Bereitstellung verwendet werden soll. Diese Rolle muss von dem verwendeten AWS Konto übernommen werden können.

Den App-Befehl angeben

Für viele Funktionen des CDK Toolkit müssen eine oder mehrere AWS CloudFormation Vorlagen synthetisiert werden, was wiederum die Ausführung Ihrer Anwendung erfordert. Das AWS CDK unterstützt Programme, die in einer Vielzahl von Sprachen geschrieben wurden. Daher verwendet es eine Konfigurationsoption, um den genauen Befehl anzugeben, der zum Ausführen Ihrer App erforderlich ist. Diese Option kann auf zwei Arten angegeben werden.

Erstens und am häufigsten kann sie mithilfe des `app` Schlüssels in der Datei angegeben werden `cdk.json`. Dies befindet sich im Hauptverzeichnis Ihres AWS CDK Projekts. Das CDK Toolkit bietet einen passenden Befehl, wenn Sie ein neues Projekt mit `cdk init` erstellen. Hier ist zum `cdk.json` Beispiel das aus einem neuen TypeScript Projekt.

```
{
  "app": "npm ts-node bin/hello-cdk.ts"
}
```

Das CDK Toolkit sucht `cdk.json` im aktuellen Arbeitsverzeichnis, wenn es versucht, Ihre App auszuführen. Aus diesem Grund können Sie im Hauptverzeichnis Ihres Projekts eine Shell für die Ausgabe von CDK Toolkit-Befehlen geöffnet lassen.

Das CDK Toolkit sucht auch nach dem `app`-Schlüssel in `~/.cdk.json` (d. h. in Ihrem Home-Verzeichnis), falls es ihn nicht finden kann. `./cdk.json` Das Hinzufügen des `app`-Befehls hier kann nützlich sein, wenn Sie normalerweise mit CDK-Code in derselben Sprache arbeiten.

Wenn du dich in einem anderen Verzeichnis befindest oder wenn du deine App mit einem anderen Befehl als dem in `cdk.json` ausführen möchtest, verwende die Option `--app` (oder `-a`), um ihn anzugeben.

```
cdk --app "npx ts-node bin/hello-cdk.ts" ls
```

Bei der Bereitstellung können Sie auch ein Verzeichnis angeben, das synthetisierte Cloud-Assemblies enthält `cdk.out`, z. B. als Wert von `--app`. Die angegebenen Stacks werden von diesem Verzeichnis aus bereitgestellt; die App wird nicht synthetisiert.

Stacks angeben

Viele CDK Toolkit-Befehle (zum Beispiel `cdk deploy`) funktionieren mit Stacks, die in Ihrer App definiert sind. Wenn Ihre App nur einen Stack enthält, geht das CDK Toolkit davon aus, dass Sie diesen meinen, wenn Sie einen Stack nicht explizit angeben.

Andernfalls müssen Sie den Stack oder die Stacks angeben, mit denen Sie arbeiten möchten. Sie können dies tun, indem Sie die gewünschten Stacks einzeln nach ID in der Befehlszeile angeben. Denken Sie daran, dass die ID der Wert ist, der durch das zweite Argument angegeben wird, wenn Sie den Stack instanziiieren.

```
cdk synth PipelineStack LambdaStack
```

Sie können auch Platzhalter verwenden, um IDs anzugeben, die einem Muster entsprechen.

- `?` entspricht einem beliebigen einzelnen Zeichen
- `*` entspricht einer beliebigen Anzahl von Zeichen (`*` allein entspricht allen Stapeln)
- `**` entspricht allem in einer Hierarchie

Sie können die `--all` Option auch verwenden, um alle Stapel anzugeben.

Wenn Ihre App [CDK Pipelines](#) verwendet, versteht das CDK Toolkit Ihre Stacks und Stages als Hierarchie. Außerdem stimmen die `--all` Option und der `*` Platzhalter nur mit Stacks der obersten Ebene überein. Um alle Stapel abzugleichen, verwenden Sie `** Wird auch verwendet**`, um alle Stapel unter einer bestimmten Hierarchie anzugeben.

Wenn Sie Platzhalter verwenden, setzen Sie das Muster in Anführungszeichen oder maskieren Sie die Platzhalter mit `\`. Wenn Sie dies nicht tun, versucht Ihre Shell möglicherweise, das Muster auf die Namen der Dateien im aktuellen Verzeichnis zu erweitern. Im besten Fall wird dies nicht das tun, was Sie erwarten. Im schlimmsten Fall könnten Sie Stacks bereitstellen, die Sie nicht beabsichtigt hatten. Dies ist unter Windows nicht unbedingt erforderlich, da Platzhalter `cmd.exe` nicht erweitert werden, ist aber dennoch eine bewährte Methode.

```
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

Die Reihenfolge, in der Sie die Stapel angeben, entspricht nicht unbedingt der Reihenfolge, in der sie verarbeitet werden. Das AWS CDK Toolkit berücksichtigt Abhängigkeiten zwischen Stacks, wenn es entscheidet, in welcher Reihenfolge sie verarbeitet werden sollen. Nehmen wir zum Beispiel an, dass ein Stapel einen Wert verwendet, der von einem anderen erzeugt wird (z. B. den ARN einer im zweiten Stack definierten Ressource). In diesem Fall wird der zweite Stapel aufgrund dieser Abhängigkeit vor dem ersten synthetisiert. Sie können Abhängigkeiten zwischen Stacks manuell hinzufügen, indem Sie die Methode des Stacks [addDependency\(\)](#) verwenden.

Bootstrapping für Ihre Umgebung AWS

Für die Bereitstellung von Stacks mit dem CDK müssen spezielle, dedizierte AWS CDK Ressourcen bereitgestellt werden. Der `cdk bootstrap` Befehl erstellt die erforderlichen Ressourcen für Sie. Sie müssen das Bootstrap nur ausführen, wenn Sie einen Stack bereitstellen, der diese dedizierten Ressourcen benötigt. Details dazu finden Sie unter [the section called “Bootstrapping”](#).

```
cdk bootstrap
```

Wenn der `cdk bootstrap` Befehl ohne Argumente ausgegeben wird, wie hier gezeigt, synthetisiert er die aktuelle App und bootet die Umgebungen, in denen die Stacks bereitgestellt werden. Wenn die App umgebungsunabhängige Stacks enthält, die nicht explizit eine Umgebung angeben, werden das Standardkonto und die Region oder die mit angegebene Umgebung gebootet. `--profile`

Außerhalb einer App müssen Sie die Umgebung, für die ein Bootstrap erstellt werden soll, explizit angeben. Sie können dies auch tun, um eine Umgebung zu booten, die nicht in Ihrer App oder Ihrem lokalen Profil angegeben ist. AWS Die Anmeldeinformationen müssen für das angegebene Konto und die angegebene Region konfiguriert sein (z. B. in `~/ .aws/credentials`). Sie können ein Profil angeben, das die erforderlichen Anmeldeinformationen enthält.

```
cdk bootstrap ACCOUNT-NUMBER/REGION # e.g.  
cdk bootstrap 1111111111/us-east-1  
cdk bootstrap --profile test 1111111111/us-east-1
```

Important

Jede Umgebung (Kombination aus Konto und Region), in der Sie einen solchen Stack bereitstellen, muss separat gestartet werden.

Möglicherweise fallen AWS Gebühren für das an, was in den Bootstrap-Ressourcen AWS CDK gespeichert wird. Darüber hinaus wird bei Verwendung `-bootstrap-customer-key` ein AWS-KMS-Schlüssel erstellt, für den ebenfalls Gebühren pro Umgebung anfallen.

Note

In früheren Versionen der Bootstrap-Vorlage wurde standardmäßig ein KMS-Schlüssel erstellt. Um Gebühren zu vermeiden, starten Sie das System erneut mithilfe von `--no-bootstrap-customer-key`

Note

CDK Toolkit v2 unterstützt nicht die ursprüngliche Bootstrap-Vorlage, die als Legacy-Template bezeichnet wird und standardmäßig mit CDK v1 verwendet wird.

Important

Das moderne Bootstrap-Template gewährt praktisch jedem Konto in der Liste die `--cloudformation-execution-policies` damit verbundenen Berechtigungen. AWS `--trust` Standardmäßig werden dadurch die Lese- und Schreibberechtigungen für alle Ressourcen im Bootstrap-Konto erweitert. Stellen Sie sicher, dass [Sie den Bootstrapping-Stack mit Richtlinien und vertrauenswürdigen Konten konfigurieren](#), mit denen Sie vertraut sind.

Eine neue App erstellen

Um eine neue App zu erstellen, erstellen Sie ein Verzeichnis dafür und geben Sie dann innerhalb des Verzeichnisses das Problem ein `cdk init`.

```
mkdir my-cdk-app
cd my-cdk-app
cdk init TEMPLATE --language LANGUAGE
```

Die unterstützten Sprachen (*LANGUAGE*) sind:

Code	Sprache
typescript	TypeScript
javascript	JavaScript
python	Python
java	Java
csharp	C#

TEMPLATE ist eine optionale Vorlage. Wenn die gewünschte Vorlage App ist, die Standardvorlage, können Sie sie weglassen. Die verfügbaren Vorlagen sind:

Vorlage	Beschreibung
app(Standard)	Erzeugt eine leere AWS CDK App.
sample-app	Erstellt eine AWS CDK App mit einem Stack, der eine Amazon SQS SQS-Warteschlange und ein Amazon SNS SNS-Thema enthält.

Die Vorlagen verwenden den Namen des Projektordners, um Namen für Dateien und Klassen in Ihrer neuen App zu generieren.

Stapel auflisten

Um eine Liste der IDs der Stacks in Ihrer AWS CDK Anwendung anzuzeigen, geben Sie einen der folgenden entsprechenden Befehle ein:

```
cdk list
cdk ls
```

Wenn Ihre Anwendung [CDK-Pipeline-Stacks enthält, zeigt das CDK Toolkit Stacknamen entsprechend ihrer Position in der Pipeline-Hierarchie](#) als Pfade an. (Zum Beispiel, PipelineStack und.) PipelineStack/Prod PipelineStack/Prod/MyService

Wenn Ihre App viele Stapel enthält, können Sie vollständige oder teilweise Stack-IDs der aufzulistenden Stapel angeben. Weitere Informationen finden Sie unter [the section called “Stacks angeben”](#).

Fügen Sie das `--long` Flag hinzu, um weitere Informationen zu den Stacks zu erhalten, einschließlich der Stack-Namen und ihrer Umgebungen (AWS Konto und Region).

Synthetisieren von Stacks

Der `cdk synth` Befehl (fast immer abgekürzt `synth`) synthetisiert einen in Ihrer App definierten Stack zu einer Vorlage. CloudFormation

```
cdk synth          # if app contains only one stack
cdk synth MyStack
cdk synth Stack1 Stack2
cdk synth "*"      # all stacks in app
```

Note

Das CDK Toolkit führt Ihre App tatsächlich aus und synthetisiert neue Vorlagen vor den meisten Vorgängen (z. B. beim Bereitstellen oder Vergleichen von Stacks). Diese Vorlagen werden standardmäßig im Verzeichnis gespeichert. `cdk.out` Der `cdk synth` Befehl druckt einfach die generierten Vorlagen für einen oder mehrere angegebene Stapel.

Alle verfügbaren Optionen finden `cdk synth --help` Sie unter. Einige der am häufigsten verwendeten Optionen werden im folgenden Abschnitt behandelt.

Angabe von Kontextwerten

Verwenden Sie die `-c` Option `--context` oder, um Werte für den [Laufzeitkontext](#) an Ihre CDK-App zu übergeben.

```
# 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
```

Bei der Bereitstellung mehrerer Stacks werden die angegebenen Kontextwerte normalerweise an alle übergeben. Wenn Sie möchten, können Sie für jeden Stack unterschiedliche Werte angeben, indem Sie dem Kontextwert den Stacknamen voranstellen.

```
# different context values for each stack
cdk synth --context Stack1:key=value Stack2:key=value Stack1 Stack2
```

Anzeigeformat angeben

Standardmäßig wird die synthetisierte Vorlage im YAML-Format angezeigt. Fügen Sie das `--json` Flag hinzu, um es stattdessen im JSON-Format anzuzeigen.

```
cdk synth --json MyStack
```

Ausgabeverzeichnis angeben

Fügen Sie die Option `--output` (`-o`) hinzu, um die synthetisierten Vorlagen in ein anderes Verzeichnis als `cdk.out` zu schreiben.

```
cdk synth --output=~/templates
```

Stacks bereitstellen

Der `cdk deploy` Unterbefehl stellt einen oder mehrere angegebene Stacks für Ihr Konto bereit.
AWS

```
cdk deploy          # if app contains only one stack
cdk deploy MyStack
```

```
cdk deploy Stack1 Stack2
cdk deploy "*"      # all stacks in app
```

Note

Das CDK Toolkit führt Ihre App aus und synthetisiert neue Vorlagen, bevor Sie etwas bereitstellen. AWS CloudFormation Daher können die meisten Befehlszeilenoptionen, die Sie mit verwenden können `cdk synth` (z. B. `--context`), auch mit verwendet werden. `cdk deploy`

Alle verfügbaren Optionen finden `cdk deploy --help` Sie unter. Einige der nützlichsten Optionen werden im folgenden Abschnitt behandelt.

Synthese überspringen

Der `cdk deploy` Befehl synthetisiert normalerweise die Stapel Ihrer App vor der Bereitstellung, um sicherzustellen, dass die Bereitstellung die neueste Version Ihrer App widerspiegelt. Wenn Sie wissen, dass Sie Ihren Code seit der letzten Version nicht geändert haben `cdk synth`, können Sie den redundanten Syntheseschritt bei der Bereitstellung unterdrücken. Geben Sie dazu in der `--app` Option das `cdk.out` Verzeichnis Ihres Projekts an.

```
cdk deploy --app cdk.out StackOne StackTwo
```

Rollback deaktivieren

AWS CloudFormation hat die Fähigkeit, Änderungen rückgängig zu machen, sodass Bereitstellungen atomar sind. Das bedeutet, dass sie entweder erfolgreich sind oder in ihrer Gesamtheit scheitern. Die AWS CDK erbt diese Fähigkeit, weil sie Vorlagen synthetisiert und bereitstellt AWS CloudFormation .

Rollback stellt sicher, dass sich Ihre Ressourcen jederzeit in einem konsistenten Zustand befinden, was für Produktionsstapel von entscheidender Bedeutung ist. Während Sie Ihre Infrastruktur noch entwickeln, sind einige Ausfälle jedoch unvermeidlich, und das Zurücksetzen fehlgeschlagener Bereitstellungen kann Sie verlangsamen.

Aus diesem Grund können Sie mit dem CDK Toolkit das Rollback deaktivieren, indem Sie Ihrem Befehl etwas hinzufügen `--no-rollback`. `cdk deploy` Mit diesem Flag werden fehlgeschlagene Bereitstellungen nicht rückgängig gemacht. Stattdessen bleiben Ressourcen, die vor der ausgefallenen Ressource bereitgestellt wurden, bestehen, und die nächste Bereitstellung beginnt mit

der ausgefallenen Ressource. Sie werden viel weniger Zeit damit verbringen, auf Bereitstellungen zu warten, und viel mehr Zeit mit der Entwicklung Ihrer Infrastruktur verbringen.

Wechseln im laufenden Betrieb

Verwenden Sie die `--hotswap` Markierung mit `cdk deploy`, um zu versuchen, Ihre AWS Ressourcen direkt zu aktualisieren, anstatt einen AWS CloudFormation Änderungsatz zu generieren und ihn bereitzustellen. Wenn Hot-Swapping nicht möglich ist, wird auf die AWS CloudFormation Bereitstellung zurückgegriffen.

Derzeit unterstützt Hot-Swapping Lambda-Funktionen, Step Functions Functions-Zustandsmaschinen und Amazon ECS-Container-Images. Das `--hotswap` Flag deaktiviert auch Rollback (d. h. impliziert). `--no-rollback`

Important

Hot-Swapping wird für Produktionsbereitstellungen nicht empfohlen.

Modus „Ansehen“

Der Watch-Modus (`cdk deploy --watch` oder `cdk watch` kurz) des CDK Toolkits überwacht kontinuierlich die Quelldateien und Assets Ihrer CDK-App auf Änderungen. Es führt sofort eine Bereitstellung der angegebenen Stacks durch, wenn eine Änderung erkannt wird.

Standardmäßig verwenden diese Bereitstellungen das `--hotswap` Flag, das die Bereitstellung von Änderungen an Lambda-Funktionen beschleunigt. Es wird auch auf die Bereitstellung über zurückgegriffen, AWS CloudFormation falls Sie die Infrastrukturkonfiguration geändert haben. Wenn Sie `cdk watch` immer vollständige AWS CloudFormation Bereitstellungen durchführen möchten, fügen Sie das `--no-hotswap` Flag zu `cdk watch` hinzu.

Alle Änderungen, die während `cdk watch` der Ausführung einer Bereitstellung vorgenommen werden, werden zu einer einzigen Bereitstellung zusammengefasst, die beginnt, sobald die Bereitstellung abgeschlossen ist.

Im Überwachungsmodus wird anhand des `"watch"` Schlüssels im Projekt bestimmt `cdk.json`, welche Dateien überwacht werden sollen. Standardmäßig handelt es sich bei diesen Dateien um Ihre Anwendungsdateien und -ressourcen. Dies kann jedoch geändert werden, indem Sie die `"exclude"` Einträge `"include"` und im `"watch"` Schlüssel ändern. Die folgende `cdk.json` Datei zeigt ein Beispiel für diese Einträge.

```
{
  "app": "mvn -e -q compile exec:java",
  "watch": {
    "include": "src/main/**",
    "exclude": "target/*"
  }
}
```

`cdk watch` führt den "build" Befehl von `auscdk.json`, um Ihre App vor der Synthese zu erstellen. Wenn für Ihre Bereitstellung Befehle erforderlich sind, um Ihren Lambda-Code (oder etwas anderes, das nicht in Ihrer CDK-App enthalten ist) zu erstellen oder zu verpacken, fügen Sie sie hier hinzu.

Platzhalter im Git-Stil, sowohl als auch `**`, können in den "watch" Tasten `*` und verwendet werden. "build" Jeder Pfad wird relativ zum übergeordneten Verzeichnis von interpretiert. `cdk.json` Der Standardwert von `include` ist `**/*`, d. h. alle Dateien und Verzeichnisse im Stammverzeichnis des Projekts. `exclude` ist optional.

Important

Der Überwachungsmodus wird für Produktionsbereitstellungen nicht empfohlen.

Parameter angeben AWS CloudFormation

Das AWS CDK Toolkit unterstützt die Angabe von AWS CloudFormation [Parametern](#) bei der Bereitstellung. Sie können diese in der Befehlszeile nach dem `--parameters` Flag angeben.

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

Um mehrere Parameter zu definieren, verwenden Sie mehrere `--parameters` Flags.

```
cdk deploy MyStack --parameters uploadBucketName=UpBucket --parameters
downloadBucketName=DownBucket
```

Wenn Sie mehrere Stacks bereitstellen, können Sie für jeden Stack einen anderen Wert für jeden Parameter angeben. Stellen Sie dazu dem Namen des Parameters den Stacknamen und einen Doppelpunkt voran. Andernfalls wird derselbe Wert an alle Stacks übergeben.

```
cdk deploy MyStack YourStack --parameters MyStack:uploadBucketName=UploadBucket --parameters YourStack:uploadBucketName=UpBucket
```

Standardmäßig AWS CDK behält die Werte von Parametern aus früheren Bereitstellungen bei und verwendet sie in späteren Bereitstellungen, sofern sie nicht explizit angegeben wurden. Verwenden Sie das `--no-previous-parameters` Flag, um zu verlangen, dass alle Parameter angegeben werden.

Ausgabedatei angeben

Wenn Ihr Stack AWS CloudFormation Ausgaben deklariert, werden diese normalerweise nach Abschluss der Bereitstellung auf dem Bildschirm angezeigt. Um sie in eine Datei im JSON-Format zu schreiben, verwenden Sie das `--outputs-file` Flag.

```
cdk deploy --outputs-file outputs.json MyStack
```

Sicherheitsrelevante Änderungen

Um Sie vor unbeabsichtigten Änderungen zu schützen, die sich auf Ihre Sicherheitslage auswirken, fordert das AWS CDK Toolkit Sie auf, sicherheitsrelevante Änderungen zu genehmigen, bevor Sie sie implementieren. Sie können den Grad der Änderung angeben, für den eine Genehmigung erforderlich ist:

```
cdk deploy --require-approval LEVEL
```

LEVEL kann einer der folgenden Werte sein:

Begriff	Bedeutung
<code>never</code>	Eine Genehmigung ist niemals erforderlich
<code>any-change</code>	Erfordert eine Genehmigung für jede IAM oder Änderung <code>security-group-related</code>
<code>broadening (Standard)</code>	Erfordert eine Genehmigung, wenn IAM-Anweisungen oder Verkehrsregeln hinzugefügt werden; für Löschungen ist keine Genehmigung erforderlich

Die Einstellung kann auch in der Datei konfiguriert werden. `cdk.json`

```
{
  "app": "...",
  "requireApproval": "never"
}
```

Stapel vergleichen

Der `cdk diff` Befehl vergleicht die aktuelle Version eines in Ihrer App definierten Stacks (und seine Abhängigkeiten) mit den bereits bereitgestellten Versionen oder mit einer gespeicherten AWS CloudFormation Vorlage und zeigt eine Liste der Änderungen an.

```
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

```

So vergleichen Sie die Stacks Ihrer App mit der vorhandenen Bereitstellung:

```
cdk diff MyStack
```

So vergleichen Sie die Stacks Ihrer App mit einer gespeicherten CloudFormation Vorlage:

```
cdk diff --template ~/stacks/MyStack.old MyStack
```

Importieren vorhandener Ressourcen in einen Stack

Sie können den `cdk import` Befehl verwenden, um Ressourcen CloudFormation für einen bestimmten AWS CDK Stack unter die Verwaltung zu stellen. Dies ist nützlich AWS CDK, wenn Sie zu Stacks migrieren oder Ressourcen zwischen Stacks verschieben oder deren logische ID ändern. `cdk import` Verwendet [CloudFormation Ressourcenimporte](#). [Eine Liste der Ressourcen, die importiert werden können, finden Sie hier](#).

Gehen Sie wie folgt vor, um eine vorhandene Ressource in einen AWS CDK Stack zu importieren:

- Stellen Sie sicher, dass die Ressource derzeit nicht von einem anderen CloudFormation Stack verwaltet wird. Ist dies der Fall, legen Sie zunächst die Entfernungsrichtlinie auf den Stapel fest, `RemovalPolicy.RETAIN` in dem sich die Ressource gerade befindet, und führen Sie eine Bereitstellung durch. Entfernen Sie dann die Ressource aus dem Stapel und führen Sie eine weitere Bereitstellung durch. Durch diesen Vorgang wird sichergestellt, dass die Ressource nicht mehr von verwaltet wird, sie wird CloudFormation aber nicht gelöscht.
- Führen Sie `cdk diff`, um sicherzustellen, dass an dem AWS CDK Stack, in den Sie Ressourcen importieren möchten, keine ausstehenden Änderungen vorliegen. Die einzigen Änderungen, die bei einem „Import“-Vorgang zulässig sind, sind das Hinzufügen neuer Ressourcen, die Sie importieren möchten.
- Fügen Sie Konstrukte für die Ressourcen hinzu, die Sie in Ihren Stack importieren möchten. Wenn Sie beispielsweise einen Amazon S3 S3-Bucket importieren möchten, fügen Sie etwas hinzu wie `new s3.Bucket(this, 'ImportedS3Bucket', {});`. Nehmen Sie keine Änderungen an anderen Ressourcen vor.

Sie müssen außerdem sicherstellen, dass der aktuelle Status der Ressource in der Definition exakt modelliert wird. Achten Sie beim Beispiel des Buckets darauf, AWS KMS Schlüssel, Lebenszyklusrichtlinien und alles andere, was für den Bucket relevant ist, anzugeben. Wenn Sie dies nicht tun, bewirken nachfolgende Aktualisierungsvorgänge möglicherweise nicht das, was Sie erwarten.

Sie können wählen, ob Sie den physischen Bucket-Namen einbeziehen möchten oder nicht. Wir empfehlen normalerweise, keine Ressourcennamen in Ihre AWS CDK Ressourcendefinitionen aufzunehmen, damit es einfacher wird, Ihre Ressourcen mehrfach bereitzustellen.

- Führen Sie `cdk import STACKNAME`.
- Wenn die Ressourcennamen nicht in Ihrem Modell enthalten sind, werden Sie von der CLI aufgefordert, die tatsächlichen Namen der Ressourcen, die Sie importieren, zu übergeben. Danach beginnt der Import.
- Wenn ein Erfolg `cdk import` gemeldet wird, wird die Ressource jetzt von AWS CDK und verwaltet CloudFormation. Alle nachfolgenden Änderungen, die Sie an den Ressourceneigenschaften in Ihrer AWS CDK App und der Construct-Konfiguration vornehmen, werden bei der nächsten Bereitstellung angewendet.
- Um zu überprüfen, ob die Ressourcendefinition in Ihrer AWS CDK App dem aktuellen Status der Ressource entspricht, können Sie einen [Vorgang zur Erkennung CloudFormation von Abweichungen](#) starten.

Diese Funktion unterstützt derzeit nicht den Import von Ressourcen in verschachtelte Stacks.

Konfiguration () `cdk.json`

Standardwerte für viele CDK Toolkit-Befehlszeilenflags können in einer `cdk.json` Projektdatei oder in der `.cdk.json` Datei in Ihrem Benutzerverzeichnis gespeichert werden. Im Folgenden finden Sie eine alphabetische Referenz zu den unterstützten Konfigurationseinstellungen.

Schlüssel	Hinweise	CDK Toolkit-Option
<code>app</code>	Der Befehl, der die CDK-Anwendung ausführt.	<code>--app</code>
<code>assetMetadata</code>	Falls <code>false</code> , fügt CDK Ressourcen, die Assets verwenden, keine Metadaten hinzu.	<code>--no-asset-metadata</code>
<code>bootstrapKmsKeyId</code>	Überschreibt die ID des AWS KMS Schlüssels, der zur Verschlüsselung des Amazon	<code>--bootstrap-kms-key-id</code>

Schlüssel	Hinweise	CDK Toolkit-Option
	S3 S3-Bereitstellungs-Buckets verwendet wurde.	
build	Der Befehl, der die CDK-Anwendung vor der Synthese kompiliert oder erstellt. Nicht erlaubt in. ~/ .cdk . json	--build
browser	Der Befehl zum Starten eines Webbrowsers für den cdk docs Unterbefehl.	--browser
context	Siehe the section called "Kontext" . Kontextwerte in einer Konfigurationsdatei werden von nicht gelöscht. cdk context --clear (Das CDK Toolkit platziert zwischeng espeicherte Kontextwerte in.) cdk . context . json	--context
debug	Fallstrue, gibt das CDK Toolkit detailliertere Informationen aus, die für das Debuggen nützlich sind.	--debug
language	Die Sprache, die für die Initialisierung neuer Projekte verwendet werden soll.	--language
lookups	Fallsfalse, sind keine Kontext-Lookups zulässig. Die Synthese schlägt fehl, wenn Kontext-Lookups durchgeführt werden müssen.	--no-lookups

Schlüssel	Hinweise	CDK Toolkit-Option
<code>notices</code>	Wenn <code>false</code> , unterdrückt die Anzeige von Meldungen über Sicherheitslücken, Regressionen und nicht unterstützte Versionen.	<code>--no-notices</code>
<code>output</code>	Der Name des Verzeichnisses, in das die synthetisierte Cloud-Assembly ausgegeben wird (Standard). <code>"cdk.out"</code>	<code>--output</code>
<code>outputsFile</code>	Die Datei, in die AWS CloudFormation Ausgaben von bereitgestellten Stacks geschrieben werden (im JSON-Format).	<code>--outputs-file</code>
<code>pathMetadata</code>	Falls <code>false</code> , werden CDK-Pfadmetadaten nicht zu synthetisierten Vorlagen hinzugefügt.	<code>--no-path-metadata</code>
<code>plugin</code>	JSON-Array, das die Paketnamen oder lokalen Pfade von Paketen angibt, die das CDK erweitern	<code>--plugin</code>
<code>profile</code>	Name des AWS Standardprofils, das für die Angabe der Region und der Kontoanmeldedaten verwendet wird.	<code>--profile</code>

Schlüssel	Hinweise	CDK Toolkit-Option
progress	Wenn auf gesetzt "events", zeigt das CDK Toolkit alle AWS CloudFormation Ereignisse während der Bereitstellung an und nicht einen Fortschrittsbalken.	--progress
requireApproval	Standardgenehmigungsstufe für Sicherheitsänderungen. Siehe the section called "Sicherheitsrelevante Änderungen"	--require-approval
rollback	Falls false, werden fehlgeschlagene Bereitstellungen nicht rückgängig gemacht.	--no-rollback
staging	Falls false, werden die Ressourcen nicht in das Ausgabeverzeichnis kopiert (zum lokalen Debuggen der Quelldateien mit AWS SAM verwenden).	--no-staging
tags	JSON-Objekt, das Tags (Schlüssel-Wert-Paare) für den Stapel enthält.	--tags
toolkitBucketName	Der Name des Amazon S3 S3-Buckets, der für die Bereitstellung von Ressourcen wie Lambda-Funktionen und Container-Images verwendet wird (siehe the section called "Bootstrapping für Ihre Umgebung AWS").	--toolkit-bucket-name

Schlüssel	Hinweise	CDK Toolkit-Option
<code>toolkitStackName</code>	Der Name des Bootstrap-Stacks (siehe. the section called "Bootstrapping für Ihre Umgebung AWS")	<code>--toolkit-stack-name</code>
<code>versionReporting</code>	Wenn <code>false</code> , deaktiviert die Versionsberichterstattung.	<code>--no-version-reporting</code>
<code>watch</code>	JSON-Objekt, das "exclude" Schlüssel "include" und Schlüssel enthält, die angeben, welche Dateien bei Änderungen eine Neuerstellung des Projekts auslösen sollen (oder nicht). Siehe the section called "Modus „Ansehen“" .	<code>--watch</code>

AWS Toolkit for Visual Studio Code

Das [AWS Toolkit for Visual Studio Code](#) ist ein Open-Source-Plugin für Visual Studio Code, das das Erstellen, Debuggen und Bereitstellen von Anwendungen auf vereinfacht AWS. Das Toolkit bietet eine integrierte Erfahrung für die Entwicklung von AWS CDK Anwendungen. Sie enthält die AWS CDK Explorer-Funktion zum Auflisten Ihrer AWS CDK Projekte und zum Durchsuchen der verschiedenen Komponenten der CDK-Anwendung. [Installieren Sie das AWS Toolkit](#) und erfahren Sie mehr über [die Verwendung von AWS CDK Explorer](#) .

AWS SAM -Integration

Die AWS CDK und die AWS Serverless Application Model (AWS SAM) können zusammenarbeiten, um Ihnen das lokale Erstellen und Testen von Serverless-Anwendungen zu ermöglichen, die im CDK definiert sind. Vollständige Informationen finden Sie unter [AWS Cloud Development Kit \(AWS CDK\)](#) im - AWS SAM Entwicklerhandbuch. Informationen zum Installieren der SAM CLI finden Sie unter [Installieren der AWS SAM CLI](#).

Konstrukte testen

Mit dem AWS CDK kann Ihre Infrastruktur genauso testbar sein wie jeder andere Code, den Sie schreiben. Der Standardansatz zum Testen von AWS CDK Apps verwendet AWS CDK das [Assertions-Modul](#) und beliebte Testframeworks wie [Jest](#) for JavaScript und/oder TypeScript [Pytest](#) for Python.

Es gibt zwei Kategorien von Tests, die Sie für Apps schreiben können. AWS CDK

- Präzise Assertionen testen bestimmte Aspekte der generierten AWS CloudFormation Vorlage, z. B. „Diese Ressource hat diese Eigenschaft mit diesem Wert“. Diese Tests können Regressionen erkennen. Sie sind auch nützlich, wenn Sie mithilfe der testgetriebenen Entwicklung neue Funktionen entwickeln. (Sie können zuerst einen Test schreiben und ihn dann bestehen lassen, indem Sie eine korrekte Implementierung schreiben.) Feinkörnige Assertionen sind die am häufigsten verwendeten Tests.
- Snapshot-Tests testen das synthetisierte AWS CloudFormation Template mit einem zuvor gespeicherten Baseline-Template. Mit Snapshot-Tests können Sie frei umgestalten, da Sie sicher sein können, dass der umgestaltete Code genauso funktioniert wie das Original. Wenn die Änderungen beabsichtigt waren, können Sie eine neue Ausgangsbasis für zukünftige Tests akzeptieren. CDK-Upgrades können jedoch auch dazu führen, dass sich synthetisierte Vorlagen ändern, sodass Sie sich nicht nur auf Snapshots verlassen können, um sicherzustellen, dass Ihre Implementierung korrekt ist.

Note

Vollständige Versionen der TypeScript Python- und Java-Apps, die in diesem Thema als Beispiele verwendet werden, sind [unter verfügbar GitHub](#).

Erste Schritte

Um zu veranschaulichen, wie diese Tests geschrieben werden, erstellen wir einen Stack, der eine AWS Step Functions Zustandsmaschine und eine AWS Lambda Funktion enthält. Die Lambda-Funktion abonniert ein Amazon SNS SNS-Thema und leitet die Nachricht einfach an die Zustandsmaschine weiter.

Erstellen Sie zunächst mit dem CDK-Toolkit ein leeres CDK-Anwendungsprojekt und installieren Sie die benötigten Bibliotheken. Die Konstrukte, die wir verwenden werden, befinden sich alle im CDK-Hauptpaket, was eine Standardabhängigkeit in Projekten ist, die mit dem CDK-Toolkit erstellt wurden. Sie müssen jedoch Ihr Test-Framework installieren.

TypeScript

```
mkdir state-machine && cd state-machine
cdk init --language=typescript
npm install --save-dev jest @types/jest
```

Erstellen Sie ein Verzeichnis für Ihre Tests.

```
mkdir test
```

Bearbeiten Sie die Projektpackage .json, um NPM mitzuteilen, wie Jest ausgeführt werden soll, und um Jest mitzuteilen, welche Arten von Dateien gesammelt werden sollen. Die erforderlichen Änderungen lauten wie folgt.

- Fügen Sie dem `scripts` Abschnitt einen neuen `test` Schlüssel hinzu
- Fügen Sie Jest und seine Typen zum `devDependencies` Abschnitt hinzu
- Fügen Sie einen neuen Schlüssel der `jest` obersten Ebene mit einer Deklaration hinzu `moduleFileExtensions`

Diese Änderungen werden in der folgenden Übersicht dargestellt. Platzieren Sie den neuen Text an der unter angegebenen Stellepackage .json. Die Platzhalter „...“ stehen für vorhandene Teile der Datei, die nicht geändert werden sollten.

```
{
  ...
  "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
```

Erstellen Sie ein Verzeichnis für Ihre Tests.

```
mkdir test
```

Bearbeiten Sie die Projektpackage .json, um NPM mitzuteilen, wie Jest ausgeführt werden soll, und um Jest mitzuteilen, welche Arten von Dateien gesammelt werden sollen. Die erforderlichen Änderungen lauten wie folgt.

- Fügen Sie dem `scripts` Abschnitt einen neuen `test` Schlüssel hinzu
- Füge Jest zum Abschnitt hinzu `devDependencies`
- Fügen Sie einen neuen Schlüssel der `jest` obersten Ebene mit einer Deklaration hinzu `moduleFileExtensions`

Diese Änderungen werden in der folgenden Übersicht dargestellt. Platzieren Sie den neuen Text an der unter angegebenen Stellepackage .json. Die Platzhalter „...“ stehen für vorhandene Teile der Datei, die nicht geändert werden sollten.

```
{
  ...
  "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 # On Windows, run '.\venv\Scripts\activate' instead
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
```

Öffnen Sie das Projekt in Ihrer bevorzugten Java-IDE. (Verwenden Sie in Eclipse „Datei“ > „Importieren“ > „Bestehende Maven-Projekte“.)

C#

```
mkdir state-machine && cd state-machine
cdk init --language=csharp
```

`src\StateMachine.sln` in Visual Studio öffnen.

Klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf die Lösung und wählen Sie Hinzufügen > Neues Projekt. Suchen Sie nach MSTest C# und fügen Sie ein MSTest-Testprojekt für C# hinzu. (Der Standardname `TestProject1` ist in Ordnung.)

Klicken Sie mit der rechten Maustaste `TestProject1`, wählen Sie Hinzufügen > Projektreferenz und fügen Sie das `StateMachine` Projekt als Referenz hinzu.

Der Beispielstapel

Hier ist der Stack, der in diesem Thema getestet wird. Wie bereits beschrieben, enthält es eine Lambda-Funktion und eine Step Functions Functions-Zustandsmaschine und akzeptiert ein oder mehrere Amazon SNS SNS-Themen. Die Lambda-Funktion abonniert die Amazon SNS SNS-Themen und leitet sie an die Zustandsmaschine weiter.

Sie müssen nichts Besonderes tun, um die App testbar zu machen. Tatsächlich unterscheidet sich dieser CDK-Stack in keiner wichtigen Weise von den anderen Beispielstapeln in diesem Handbuch.

TypeScript

Platzieren Sie den folgenden Code in: `lib/state-machine-stack.ts`

```
import * as cdk from "aws-cdk-lib";
import * as sns from "aws-cdk-lib/aws-sns";
import * as sns_subscriptions from "aws-cdk-lib/aws-sns-subscriptions";
import * as lambda from "aws-cdk-lib/aws-lambda";
import * as sfm 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 sfm.StateMachine(this, "StateMachine", {
      definition: new sfm.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

Geben Sie den folgenden Code ein `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 sfm = 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 sfm.StateMachine(this, "StateMachine", {
      definition: new sfm.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

Geben Sie den folgenden Code ein `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);
    }
}
}
```

Wir werden den Haupteinstiegspunkt der App so ändern, dass wir unseren Stack nicht wirklich instanzieren. Wir wollen es nicht versehentlich bereitstellen. Unsere Tests werden eine App und eine Instanz des Stacks zum Testen erstellen. In Kombination mit testgetriebener Entwicklung ist dies eine nützliche Taktik: Stellen Sie sicher, dass der Stack alle Tests besteht, bevor Sie die Bereitstellung aktivieren.

TypeScript

In `bin/state-machine.ts`:

```
#!/usr/bin/env node
import * as cdk from "aws-cdk-lib";

const app = new cdk.App();

// Stacks are intentionally not created here -- this application isn't meant to
// be deployed.
```

JavaScript

In `bin/state-machine.js`:

```
#!/usr/bin/env node
const cdk = require("aws-cdk-lib");
```

```
const app = new cdk.App();

// Stacks are intentionally not created here -- this application isn't meant to
// be deployed.
```

Python

In `app.py`:

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

import aws_cdk as cdk

app = cdk.App()

# Stacks are intentionally not created here -- this application isn't meant to
# be deployed.

app.synth()
```

Java

```
package software.amazon.samples.awscdkassertionssamples;

import software.amazon.awscdk.App;

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

        // Stacks are intentionally not created here -- this application isn't meant
        to be deployed.

        app.synth();
    }
}
```

C#

```
using Amazon.CDK;
```

```
namespace AwsCdkAssertionSamples
{
    sealed class Program
    {
        public static void Main(string[] args)
        {
            var app = new App();

            // Stacks are intentionally not created here -- this application isn't
            meant to be deployed.

            app.Synth();
        }
    }
}
```

Die Lambda-Funktion

Unser Beispielstapel enthält eine Lambda-Funktion, die unsere Zustandsmaschine startet. Wir müssen den Quellcode für diese Funktion bereitstellen, damit das CDK ihn als Teil der Erstellung der Lambda-Funktionsressource bündeln und bereitstellen kann.

- Erstellen Sie den Ordner `start-state-machine` im Hauptverzeichnis der App.
- Erstellen Sie in diesem Ordner mindestens eine Datei. Sie können beispielsweise den folgenden Code in `speichernstart-state-machines/index.js` speichern.

```
exports.handler = async function (event, context) {
    return 'hello world';
};
```

Es wird jedoch jede Datei funktionieren, da wir den Stack nicht wirklich bereitstellen werden.

Ausführen von Tests

Als Referenz finden Sie hier die Befehle, mit denen Sie Tests in Ihrer AWS CDK App ausführen. Dies sind dieselben Befehle, mit denen Sie die Tests in jedem Projekt mit demselben Testframework ausführen würden. Fügen Sie bei Sprachen, die einen Build-Schritt erfordern, diesen hinzu, um sicherzustellen, dass Ihre Tests kompiliert wurden.

TypeScript

```
tsc && npm test
```

JavaScript

```
npm test
```

Python

```
python -m pytest
```

Java

```
mvn compile && mvn test
```

C#

Erstellen Sie Ihre Lösung (F6), um die Tests zu ermitteln, und führen Sie dann die Tests aus (Test > Alle Tests ausführen). Um auszuwählen, welche Tests ausgeführt werden sollen, öffnen Sie den Test Explorer (Test > Test Explorer).

Oder:

```
dotnet test src
```

Fein abgestufte Behauptungen

Der erste Schritt zum Testen eines Stacks mit feinkörnigen Assertionen besteht darin, den Stack zu synthetisieren, da wir Assertionen anhand der generierten Vorlage schreiben. AWS CloudFormation

Unser `StateMachineStackStack` verlangt, dass wir ihm das Amazon SNS SNS-Thema übergeben, damit es an den Zustandsmaschine weitergeleitet wird. In unserem Test erstellen wir also einen separaten Stapel, der das Thema enthält.

Normalerweise können Sie beim Schreiben einer CDK-App das Amazon SNS SNS-Thema im Stack Konstruktor des Stacks unterordnen und instanziierten. In unserem Test instanziierten wir Stack direkt, übergeben diesen Stack dann als Gültigkeitsbereich und hängen ihn an den Topic Stack an.

Das ist funktionell äquivalent und weniger ausführlich. Es trägt auch dazu bei, dass Stacks, die nur in Tests verwendet werden, „anders aussehen“ als die Stacks, die Sie bereitstellen möchten.

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
    }
}
}
```

Jetzt können wir behaupten, dass die Lambda-Funktion und das Amazon SNS SNS-Abonnement erstellt wurden.

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);
```

Unser Lambda-Funktionstest bestätigt, dass zwei bestimmte Eigenschaften der Funktionsressource spezifische Werte haben. Standardmäßig führt die `hasResourceProperties` Methode eine teilweise Übereinstimmung der Eigenschaften der Ressource durch, wie sie in der synthetisierten CloudFormation Vorlage angegeben sind. Dieser Test setzt voraus, dass die angegebenen Eigenschaften vorhanden sind und die angegebenen Werte haben. Die Ressource kann jedoch auch andere Eigenschaften haben, die nicht getestet wurden.

Unsere Amazon SNS SNS-Behauptung behauptet, dass die synthetisierte Vorlage ein Abonnement enthält, aber nichts über das Abonnement selbst. Wir haben diese Behauptung hauptsächlich aufgenommen, um zu veranschaulichen, wie die Anzahl der Ressourcen bestätigt werden kann.

Die `Template` Klasse bietet spezifischere Methoden, um Assertions für die Mapping Abschnitte `ResourcesOutputs`, und der CloudFormation Vorlage zu schreiben.

Matcher

Das standardmäßige Verhalten bei teilweisem Abgleich von `hasResourceProperties` kann mithilfe von Matchern aus der [Match](#) Klasse geändert werden.

Die Übereinstimmungen reichen von lenient (`Match.anyValue`) bis strikt (`()`). `Match.objectEquals` Sie können verschachtelt werden, um unterschiedliche Abgleichmethoden auf verschiedene Teile der Ressourceneigenschaften anzuwenden. Mithilfe von `Match.objectEquals` und `Match.anyValue` zusammen können wir beispielsweise die IAM-Rolle der Zustandsmaschine umfassender testen, ohne dass spezifische Werte für Eigenschaften erforderlich sind, die sich ändern könnten.

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" }
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
));

```

Viele CloudFormation Ressourcen enthalten serialisierte JSON-Objekte, die als Zeichenketten dargestellt werden. Der `Match.serializedJson()` Matcher kann verwendet werden, um Eigenschaften in diesem JSON abzugleichen.

Beispielsweise werden Step Functions Functions-Zustandsmaschinen mithilfe einer Zeichenfolge in der JSON-basierten [Amazon States-Sprache definiert](#). Damit stellen wir `sicherMatch.serializedJson()`, dass unser Ausgangsstatus der einzige Schritt ist. Auch hier

verwenden wir verschachtelte Matcher, um verschiedene Arten von Matching auf verschiedene Teile des Objekts anzuwenden.

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

```

Erfassen

Es ist oft nützlich, Eigenschaften zu testen, um sicherzustellen, dass sie bestimmten Formaten entsprechen oder denselben Wert wie eine andere Eigenschaft haben, ohne ihre genauen Werte im Voraus kennen zu müssen. Das `assertions` Modul bietet diese Funktion in seiner [Capture](#)-Klasse.

Durch die Angabe einer `Capture`-Instanz anstelle eines Werts in `hasResourceProperties` wird dieser Wert im `Capture`-Objekt beibehalten. Der tatsächlich erfasste Wert kann mit den `as`-Methoden des Objekts, einschließlich `asNumber()`, und `asString()asObject`, abgerufen und einem Test unterzogen werden. Verwenden Sie `Capture` ihn zusammen mit einem `Matcher`, um die genaue Position des zu erfassenden Werts in den Eigenschaften der Ressource, einschließlich serialisierter JSON-Eigenschaften, anzugeben.

Das folgende Beispiel testet, ob der Startstatus unserer Zustandsmaschine einen Namen hat, der mit `Start` beginnt. Es testet auch, ob dieser Status in der Liste der Zustände in der Maschine vorhanden ist.

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()));
```

Snapshot-Tests

Beim Snapshot-Testen vergleichen Sie die gesamte synthetisierte CloudFormation Vorlage mit einer zuvor gespeicherten Baseline-Vorlage (oft als „Master“ bezeichnet). Im Gegensatz zu feinkörnigen Behauptungen sind Snapshot-Tests nicht nützlich, um Regressionen abzufangen. Das liegt daran, dass Snapshot-Tests für die gesamte Vorlage gelten und Dinge außer Codeänderungen zu kleinen (oder not-so-small) Unterschieden in den Syntheseergebnissen führen können. Diese Änderungen wirken sich möglicherweise nicht einmal auf Ihre Bereitstellung aus, führen aber dennoch dazu, dass ein Snapshot-Test fehlschlägt.

Sie könnten beispielsweise ein CDK-Konstrukt aktualisieren, um eine neue bewährte Methode einzubeziehen, die zu Änderungen an den synthetisierten Ressourcen oder deren Organisation führen kann. Alternativ könnten Sie das CDK Toolkit auf eine Version aktualisieren, die zusätzliche Metadaten meldet. Änderungen an den Kontextwerten können sich auch auf die synthetisierte Vorlage auswirken.

Snapshot-Tests können jedoch beim Refactoring eine große Hilfe sein, solange Sie alle anderen Faktoren, die sich auf das synthetisierte Template auswirken könnten, konstant halten. Sie werden sofort wissen, ob eine Änderung, die Sie vorgenommen haben, die Vorlage unbeabsichtigt verändert hat. Wenn die Änderung beabsichtigt ist, akzeptieren Sie einfach die neue Vorlage als Grundlage.

Wenn wir zum Beispiel dieses `DeadLetterQueue` Konstrukt haben:

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()
            });
        }
    }
}
```

Wir können es so testen:

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());
    }
}
}
```

Tipps für Tests

Denken Sie daran, dass Ihre Tests genauso lange gültig sind wie der Code, den sie testen, und sie werden genauso oft gelesen und geändert. Daher lohnt es sich, sich einen Moment Zeit zu nehmen, um zu überlegen, wie man sie am besten schreibt.

Kopieren Sie keine Setup-Zeilen oder allgemeine Behauptungen und fügen Sie sie nicht ein. Refaktorisieren Sie diese Logik stattdessen in Fixtures oder Hilfsfunktionen. Verwenden Sie aussagekräftige Namen, die widerspiegeln, was jeder Test tatsächlich testet.

Versuchen Sie nicht, in einem Test zu viel zu tun. Vorzugsweise sollte ein Test nur ein Verhalten testen. Wenn Sie dieses Verhalten versehentlich brechen, sollte genau ein Test fehlschlagen, und der Name des Tests sollte Ihnen sagen, was fehlgeschlagen ist. Dies ist jedoch eher ein Ideal, das angestrebt werden sollte. Manchmal werden Sie unweigerlich (oder versehentlich) Tests schreiben, die mehr als ein Verhalten testen. Snapshot-Tests sind aus Gründen, die wir bereits beschrieben haben, besonders anfällig für dieses Problem. Verwenden Sie sie daher sparsam.

Sicherheit für AWS Cloud Development Kit (AWS CDK)

Cloud-Sicherheit genießt bei Amazon Web Services (AWS) höchste Priorität. Als AWS Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die auf die Anforderungen der sicherheitsempfindlichsten Unternehmen zugeschnitten sind. Sicherheit ist eine gemeinsame Verantwortung von Ihnen AWS und Ihnen. Im [Modell der übergreifenden Verantwortlichkeit](#) wird Folgendes mit „Sicherheit der Cloud“ bzw. „Sicherheit in der Cloud“ umschrieben:

Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der alle in der AWS Cloud angebotenen Dienste ausgeführt werden, und für die Bereitstellung von Diensten, die Sie sicher nutzen können. Unsere Sicherheitsverantwortung hat bei uns höchste Priorität AWS, und die Wirksamkeit unserer Sicherheit wird im Rahmen der [AWS Compliance-Programme](#) regelmäßig von externen Prüfern getestet und verifiziert.

Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem von Ihnen genutzten AWS Dienst und anderen Faktoren, wie der Sensibilität Ihrer Daten, den Anforderungen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften.

Das AWS CDK folgt dem [Modell der gemeinsamen Verantwortung](#) durch die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Themen

- [Identitäts- und Zugriffsmanagement für AWS Cloud Development Kit \(AWS CDK\)](#)
- [Konformitätsprüfung für AWS Cloud Development Kit \(AWS CDK\)](#)
- [Resilienz für AWS Cloud Development Kit \(AWS CDK\)](#)
- [Sicherheit der Infrastruktur für AWS Cloud Development Kit \(AWS CDK\)](#)

Identitäts- und Zugriffsmanagement für AWS Cloud Development Kit (AWS CDK)

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf AWS Ressourcen sicher zu kontrollieren. IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um Ressourcen zu

verwenden. AWS IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

Zielgruppe

Die Art und Weise, wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von der Arbeit ab, in der Sie tätig sind. AWS

Dienstbenutzer — Wenn Sie dies AWS-Services für Ihre Arbeit verwenden, stellt Ihnen Ihr Administrator die erforderlichen Anmeldeinformationen und Berechtigungen zur Verfügung. Wenn Sie für Ihre Arbeit mehr AWS Funktionen verwenden, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Funktionsweise der Zugriffskontrolle nachvollziehen, wissen Sie bereits, welche Berechtigungen Sie von Ihrem Administrator anzufordern müssen.

Serviceadministrator — Wenn Sie in Ihrem Unternehmen für AWS Ressourcen verantwortlich sind, haben Sie wahrscheinlich vollen Zugriff auf AWS Ressourcen. Es ist Ihre Aufgabe, zu bestimmen, auf welche AWS-Services Ressourcen Ihre Servicebenutzer zugreifen sollen. Sie müssen dann Anträge an Ihren IAM-Administrator stellen, um die Berechtigungen Ihrer Servicenutzer zu ändern. Lesen Sie die Informationen auf dieser Seite, um die Grundkonzepte von IAM nachzuvollziehen.

IAM-Administrator: Wenn Sie als IAM-Administrator fungieren, sollten Sie Einzelheiten dazu kennen, wie Sie Richtlinien zur Verwaltung des Zugriffs auf AWS-Services verfassen können.

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen als IAM-Benutzer authentifiziert (angemeldet AWS) sein oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich AWS als föderierte Identität anmelden, indem Sie Anmeldeinformationen verwenden, die über eine Identitätsquelle bereitgestellt wurden. AWS IAM Identity Center (IAM Identity Center) -Benutzer, die Single Sign-On-Authentifizierung Ihres Unternehmens und Ihre Google- oder Facebook-Anmeldeinformationen sind Beispiele für föderierte Identitäten. Wenn Sie sich als Verbundidentität anmelden, hat der Administrator vorher mithilfe von IAM-Rollen einen Identitätsverbund eingerichtet. Wenn Sie über den Verbund darauf zugreifen AWS , übernehmen Sie indirekt eine Rolle.

Je nachdem, welcher Benutzertyp Sie sind, können Sie sich beim AWS Management Console oder beim AWS Zugangsportale anmelden. Weitere Informationen zur Anmeldung finden Sie AWS unter [So melden Sie sich bei Ihrem an AWS-Konto](#) im AWS-Anmeldung Benutzerhandbuch.

Für den AWS programmgesteuerten Zugriff AWS stellt es Software Development Kits (SDKs) und eine Befehlszeilenschnittstelle (CLI) bereit, mit der Sie Ihre Anfragen mit Ihren Anmeldeinformationen kryptografisch signieren können. AWS CDK Wenn Sie keine AWS Tools verwenden, müssen Sie Anfragen selbst signieren. Weitere Informationen über die empfohlene Methode zur eigenständigen Signierung von Anfragen finden Sie unter [Signierprozess mit Signaturversion 4](#) in der Allgemeine AWS-Referenz.

Unabhängig von der verwendeten Authentifizierungsmethode müssen Sie möglicherweise zusätzliche Sicherheitsinformationen angeben. AWS Empfiehlt beispielsweise, die Multi-Faktor-Authentifizierung (MFA) zu verwenden, um die Sicherheit Ihres Kontos zu erhöhen. Weitere Informationen finden Sie unter [Multi-Faktor-Authentifizierung](#) im AWS IAM Identity Center - Benutzerhandbuch und [Verwenden der Multi-Faktor-Authentifizierung \(MFA\) in AWS](#) im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, die vollständigen Zugriff auf alle AWS-Services Ressourcen im Konto hat. Diese Identität wird als AWS-Konto Root-Benutzer bezeichnet. Sie können darauf zugreifen, indem Sie sich mit der E-Mail-Adresse und dem Passwort anmelden, mit denen Sie das Konto erstellt haben. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Schützen Sie Ihre Root-Benutzer-Anmeldeinformationen und verwenden Sie diese, um die Aufgaben auszuführen, die nur der Root-Benutzer ausführen kann. Eine vollständige Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter [Aufgaben, die Root-Benutzer-Anmeldeinformationen erfordern](#) im IAM-Benutzerhandbuch.

Verbundidentität

Als bewährte Methode sollten menschliche Benutzer, einschließlich Benutzer, die Administratorzugriff benötigen, für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen den Verbund mit einem Identitätsanbieter verwenden.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensbenutzerverzeichnis, einem Web-Identitätsanbieter AWS Directory Service, dem Identity Center-Verzeichnis oder einem beliebigen Benutzer, der mithilfe AWS-Services von Anmeldeinformationen zugreift, die über eine Identitätsquelle bereitgestellt wurden. Wenn föderierte Identitäten darauf zugreifen AWS-Konten, übernehmen sie Rollen, und die Rollen stellen temporäre Anmeldeinformationen bereit.

Für die zentrale Zugriffsverwaltung empfehlen wir Ihnen, AWS IAM Identity Center zu verwenden. Sie können Benutzer und Gruppen in IAM Identity Center erstellen, oder Sie können eine Verbindung zu einer Gruppe von Benutzern und Gruppen in Ihrer eigenen Identitätsquelle herstellen und diese synchronisieren, um sie in all Ihren AWS-Konten Anwendungen zu verwenden. Informationen zu IAM Identity Center finden Sie unter [Was ist IAM Identity Center?](#) im AWS IAM Identity Center - Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein [IAM-Benutzer](#) ist eine Identität innerhalb Ihres Unternehmens AWS-Konto , die über spezifische Berechtigungen für eine einzelne Person oder Anwendung verfügt. Wenn möglich, empfehlen wir, temporäre Anmeldeinformationen zu verwenden, anstatt IAM-Benutzer zu erstellen, die langfristige Anmeldeinformationen wie Passwörter und Zugriffsschlüssel haben. Bei speziellen Anwendungsfällen, die langfristige Anmeldeinformationen mit IAM-Benutzern erfordern, empfehlen wir jedoch, die Zugriffsschlüssel zu rotieren. Weitere Informationen finden Sie unter [Regelmäßiges Rotieren von Zugriffsschlüsseln für Anwendungsfälle, die langfristige Anmeldeinformationen erfordern](#) im IAM-Benutzerhandbuch.

Eine [IAM-Gruppe](#) ist eine Identität, die eine Sammlung von IAM-Benutzern angibt. Sie können sich nicht als Gruppe anmelden. Mithilfe von Gruppen können Sie Berechtigungen für mehrere Benutzer gleichzeitig angeben. Gruppen vereinfachen die Verwaltung von Berechtigungen, wenn es zahlreiche Benutzer gibt. Sie könnten beispielsweise einer Gruppe mit dem Namen IAMAdmins Berechtigungen zum Verwalten von IAM-Ressourcen erteilen.

Benutzer unterscheiden sich von Rollen. Ein Benutzer ist einer einzigen Person oder Anwendung eindeutig zugeordnet. Eine Rolle kann von allen Personen angenommen werden, die sie benötigen. Benutzer besitzen dauerhafte Anmeldeinformationen. Rollen stellen temporäre Anmeldeinformationen bereit. Weitere Informationen finden Sie unter [Erstellen eines IAM-Benutzers \(anstatt einer Rolle\)](#) im IAM-Benutzerhandbuch.

IAM-Rollen

Eine [IAM-Rolle](#) ist eine Identität innerhalb Ihres Unternehmens AWS-Konto , die über bestimmte Berechtigungen verfügt. Es ähnelt einem IAM-Benutzer, ist jedoch keiner bestimmten Person zugeordnet. Sie können vorübergehend eine IAM-Rolle in der übernehmen, AWS Management Console indem Sie die Rollen [wechseln](#). Sie können eine Rolle übernehmen, indem Sie eine AWS CLI oder AWS API-Operation aufrufen oder eine benutzerdefinierte URL verwenden. Weitere Informationen zu Methoden für die Verwendung von Rollen finden Sie unter [Verwenden von IAM-Rollen](#) im IAM-Benutzerhandbuch.

IAM-Rollen mit temporären Anmeldeinformationen sind in folgenden Situationen hilfreich:

- **Verbundbenutzerzugriff** – Um einer Verbundidentität Berechtigungen zuzuweisen, erstellen Sie eine Rolle und definieren Berechtigungen für die Rolle. Wird eine Verbundidentität authentifiziert, so wird die Identität der Rolle zugeordnet und erhält die von der Rolle definierten Berechtigungen. Informationen zu Rollen für den Verbund finden Sie unter [Erstellen von Rollen für externe Identitätsanbieter](#) im IAM-Benutzerhandbuch. Wenn Sie IAM Identity Center verwenden, konfigurieren Sie einen Berechtigungssatz. Wenn Sie steuern möchten, worauf Ihre Identitäten nach der Authentifizierung zugreifen können, korreliert IAM Identity Center den Berechtigungssatz mit einer Rolle in IAM. Informationen zu Berechtigungssätzen finden Sie unter [Berechtigungssätze](#) im AWS IAM Identity Center -Benutzerhandbuch.
- **Temporäre IAM-Benutzerberechtigungen** – Ein IAM-Benutzer oder eine -Rolle kann eine IAM-Rolle übernehmen, um vorübergehend andere Berechtigungen für eine bestimmte Aufgabe zu erhalten.
- **Kontoübergreifender Zugriff** – Sie können eine IAM-Rolle verwenden, um einem vertrauenswürdigen Prinzipal in einem anderen Konto den Zugriff auf Ressourcen in Ihrem Konto zu ermöglichen. Rollen stellen die primäre Möglichkeit dar, um kontoübergreifendem Zugriff zu gewähren. Bei einigen können Sie AWS-Services jedoch eine Richtlinie direkt an eine Ressource anhängen (anstatt eine Rolle als Proxy zu verwenden). Informationen zu den Unterschieden zwischen Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter [So unterscheiden sich IAM-Rollen von ressourcenbasierten Richtlinien](#) im IAM-Benutzerhandbuch.
- **Serviceübergreifender Zugriff** — Einige AWS-Services verwenden Funktionen in anderen AWS-Services. Wenn Sie beispielsweise einen Aufruf in einem Service tätigen, führt dieser Service häufig Anwendungen in Amazon-EC2 aus oder speichert Objekte in Amazon-S3. Ein Dienst kann dies mit den Berechtigungen des aufrufenden Prinzipals mit einer Servicerolle oder mit einer serviceverknüpften Rolle tun.
 - **Servicerolle** – Eine Servicerolle ist eine [IAM-Rolle](#), die ein Service übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter [Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service](#) im IAM-Benutzerhandbuch.
 - **Dienstbezogene Rolle** — Eine dienstbezogene Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Servicebezogene Rollen erscheinen in Ihrem Dienst AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.

- Auf Amazon EC2 ausgeführte Anwendungen — Sie können eine IAM-Rolle verwenden, um temporäre Anmeldeinformationen für Anwendungen zu verwalten, die auf einer EC2-Instance ausgeführt werden und API-Anfragen stellen AWS CLI . AWS Das ist eher zu empfehlen, als Zugriffsschlüssel innerhalb der EC2-Instance zu speichern. Um einer EC2-Instance eine AWS Rolle zuzuweisen und sie allen ihren Anwendungen zur Verfügung zu stellen, erstellen Sie ein Instance-Profil, das an die Instance angehängt ist. Ein Instance-Profil enthält die Rolle und ermöglicht, dass Programme, die in der EC2-Instance ausgeführt werden, temporäre Anmeldeinformationen erhalten. Weitere Informationen finden Sie unter [Verwenden einer IAM-Rolle zum Erteilen von Berechtigungen für Anwendungen, die auf Amazon-EC2-Instances ausgeführt werden](#) im IAM-Benutzerhandbuch.

Informationen dazu, wann Sie IAM-Rollen oder IAM-Benutzer verwenden sollten, finden Sie unter [Erstellen einer IAM-Rolle \(anstatt eines Benutzers\)](#) im IAM-Benutzerhandbuch.

Konformitätsprüfung für AWS Cloud Development Kit (AWS CDK)

Das AWS CDK folgt dem [Modell der gemeinsamen Verantwortung](#) durch die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Die Sicherheit und Konformität der AWS Services wird von externen Prüfern im Rahmen mehrerer AWS Compliance-Programme bewertet. Dazu gehören SOC, PCI, FedRAMP, HIPAA und andere. AWS bietet unter AWS Services in Scope by Compliance Program eine häufig aktualisierte Liste der [AWS Services im](#) Rahmen bestimmter Compliance-Programme.

Prüfberichte von Drittanbietern können Sie mit AWS Artifact herunterladen. Weitere Informationen finden Sie unter [Berichte herunterladen in AWS Artifact](#).

Weitere Informationen zu AWS Compliance-Programmen finden Sie unter [AWS Compliance-Programme](#).

Ihre Compliance-Verantwortung bei der Nutzung von AWS CDK für den Zugriff auf einen AWS Service richtet sich nach der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften. Wenn Ihre Nutzung eines AWS Dienstes der Einhaltung von Standards wie HIPAA, PCI oder FedRAMP unterliegt, AWS bietet Ihnen folgende Ressourcen:

- [Schnellstartanleitungen zu Sicherheit und Compliance — Implementierungsleitfäden](#), in denen architektonische Überlegungen erörtert und Schritte zur Implementierung von sicherheits- und Compliance-orientierten Basisumgebungen beschrieben werden. AWS
- [AWS Ressourcen zur Einhaltung](#) von Vorschriften — Eine Sammlung von Arbeitsmappen und Leitfäden, die möglicherweise auf Ihre Branche und Ihren Standort zutreffen.
- [AWS Config](#): Ein Service, der die Konformität Ihrer Ressourcenkonfigurationen mit internen Praktiken, Branchenrichtlinien und Vorschriften bewertet.
- [AWS Security Hub](#)— Ein umfassender Überblick über Ihren aktuellen Sicherheitsstatus AWS , anhand dessen Sie überprüfen können, ob Sie die Sicherheitsstandards und Best Practices der Branche einhalten.

Resilienz für AWS Cloud Development Kit (AWS CDK)

Die globale Infrastruktur von Amazon Web Services (AWS) basiert auf AWS Regionen und Availability Zones.

AWS Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind.

Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Availability Zones ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser hoch verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter [AWS Globale Infrastruktur](#).

Das AWS CDK folgt dem [Modell der gemeinsamen Verantwortung](#) durch die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Sicherheit der Infrastruktur für AWS Cloud Development Kit (AWS CDK)

Das AWS CDK folgt dem [Modell der gemeinsamen Verantwortung](#) durch die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Fehlerbehebung bei häufigen AWS CDK Problemen

In diesem Thema wird beschrieben, wie Sie die folgenden Probleme mit dem beheben AWS CDK.

- [Nach der Aktualisierung der meldet AWS CDK das AWS CDK Toolkit \(CLI\) eine Nichtübereinstimmung mit der AWS Konstruktbibliothek](#)
- [Bei der Bereitstellung meines AWS CDK Stacks erhalte ich eine NoSuchBucket Fehlermeldung](#)
- [Bei der Bereitstellung meines AWS CDK Stacks erhalte ich eine forbidden: null Nachricht](#)
- [Beim Synthetisieren eines - AWS CDK Stacks erhalte ich die Nachricht --app is required either in command-line, in cdk.json or in ~/.cdk.json](#)
- [Beim Synthetisieren eines - AWS CDK Stacks erhalte ich eine Fehlermeldung, da die AWS CloudFormation Vorlage zu viele Ressourcen enthält.](#)
- [Ich habe drei \(oder mehr\) Availability Zones für meine Auto Scaling-Gruppe oder VPC angegeben, diese wurde jedoch nur in zwei bereitgestellt](#)
- [Mein S3-Bucket, meine DynamoDB-Tabelle oder eine andere Ressource wird nicht gelöscht, wenn ich ausstelle cdk destroy](#)

Nach der Aktualisierung der meldet AWS CDK das AWS CDK Toolkit (CLI) eine Nichtübereinstimmung mit der AWS Konstruktbibliothek

Die Version des AWS CDK Toolkits (die den `cdk` Befehl bereitstellt) muss mindestens der Version des AWS Construct Library-Hauptmoduls entsprechen, `aws-cdk-lib`. Das Toolkit ist abwärtskompatibel. Die neueste 2.x-Version des Toolkits kann mit jeder 1.x- oder 2.x-Version der Bibliothek verwendet werden. Aus diesem Grund empfehlen wir Ihnen, diese Komponente global zu installieren und auf dem neuesten Stand zu halten.

```
npm update -g aws-cdk
```

Wenn Sie mit mehreren Versionen des AWS CDK Toolkits arbeiten müssen, installieren Sie eine bestimmte Version des Toolkits lokal in Ihrem Projektordner.

Wenn Sie TypeScript oder verwenden JavaScript, enthält Ihr Projektverzeichnis bereits eine versionierte lokale Kopie des CDK Toolkits.

Wenn Sie eine andere Sprache verwenden, verwenden Sie `npm` um das AWS CDK Toolkit zu installieren, lassen Sie das `--g` Flag weg und geben Sie die gewünschte Version an. Beispielsweise:

```
npm install aws-cdk@2.0
```

Um ein lokal installiertes AWS CDK Toolkit auszuführen, verwenden Sie den Befehl `npx aws-cdk` anstelle von nur `cdk`. Beispielsweise:

```
npx aws-cdk deploy MyStack
```

`npx aws-cdk` führt die lokale Version des AWS CDK Toolkits aus, falls eine vorhanden ist. Er greift auf die globale Version zurück, wenn ein Projekt keine lokale Installation hat. Möglicherweise ist es praktisch, einen Shell-Alias einzurichten, um sicherzustellen, dass immer auf diese Weise aufgerufen `cdk` wird.

macOS/Linux

```
alias cdk="npx aws-cdk"
```

Windows

```
doskey cdk=npx aws-cdk $*
```

[\(Zurück zur Liste von \)](#)

Bei der Bereitstellung meines AWS CDK Stacks erhalte ich eine **NoSuchBucket** Fehlermeldung

Ihre AWS Umgebung wurde nicht gestartet und verfügt daher nicht über einen Amazon S3-Bucket, in dem Ressourcen während der Bereitstellung gespeichert werden können. Sie können den Staging-Bucket und andere erforderliche Ressourcen mit dem folgenden Befehl erstellen:

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

Um unerwartete AWS Gebühren zu vermeiden, startet die AWS CDK nicht automatisch eine Umgebung. Sie müssen jede Umgebung, in der Sie bereitstellen werden, explizit booten.

Standardmäßig werden die Bootstrap-Ressourcen in der Region oder den Regionen erstellt, die von Stacks in der aktuellen AWS CDK Anwendung verwendet werden. Alternativ werden sie in der

Region erstellt, die in Ihrem lokalen AWS Profil angegeben ist (festgelegt durch `aws configure`), indem das Konto dieses Profils verwendet wird. Sie können in der Befehlszeile wie folgt ein anderes Konto und eine andere Region angeben. (Sie müssen das Konto und die Region angeben, wenn Sie sich nicht im Verzeichnis einer App befinden.)

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

Weitere Informationen finden Sie unter [the section called "Bootstrapping"](#).

[\(Zurück zur Liste von \)](#)

Bei der Bereitstellung meines AWS CDK Stacks erhalte ich eine **forbidden: null** Nachricht

Sie stellen einen Stack bereit, der Bootstrap-Ressourcen benötigt, verwenden jedoch eine IAM-Rolle oder ein IAM-Konto, für das keine Schreibberechtigung erforderlich ist. (Der Staging-Bucket wird verwendet, wenn Stacks bereitgestellt werden, die Komponenten enthalten oder eine - AWS CloudFormation Vorlage synthetisieren, die größer als 50K00 ist.) Verwenden Sie ein Konto oder eine Rolle, das/die berechtigt ist, die Aktion `s3: *` für den in der Fehlermeldung genannten Bucket auszuführen.

[\(Zurück zur Liste von \)](#)

Beim Synthetisieren eines - AWS CDK Stacks erhalte ich die Nachricht **--app is required either in command-line, in cdk.json or in ~/.cdk.json**

Diese Meldung bedeutet in der Regel, dass Sie sich nicht im Hauptverzeichnis Ihres AWS CDK Projekts befinden, wenn Sie `cdk synth` ausgeben. Die Datei `cdk.json` in diesem Verzeichnis, die durch den `cdk init` Befehl erstellt wurde, enthält die Befehlszeile, die zum Ausführen (und damit Synthetisieren) Ihrer AWS CDK App erforderlich ist. Für eine TypeScript App sieht die Standardeinstellung beispielsweise etwa wie folgt `cdk.json` aus:

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

Wir empfehlen, `cdk` Befehle nur im Hauptverzeichnis Ihres Projekts auszugeben, damit das AWS CDK Toolkit `cdk.json` dort finden und Ihre App erfolgreich ausführen kann.

Wenn dies aus irgendeinem Grund nicht sinnvoll ist, sucht das AWS CDK Toolkit an zwei anderen Speicherorten nach der Befehlszeile der App:

- In `cdk.json` in Ihrem Stammverzeichnis
- Auf dem `cdk synth` Befehl selbst mit der `-a` Option

Sie können beispielsweise einen Stack aus einer TypeScript App wie folgt synthetisieren.

```
cdk synth --app "npx ts-node my-cdk-app.ts" MyStack
```

[\(Zurück zur Liste von\)](#)

Beim Synthetisieren eines - AWS CDK Stacks erhalte ich eine Fehlermeldung, da die AWS CloudFormation Vorlage zu viele Ressourcen enthält.

Der AWS CDK generiert und stellt AWS CloudFormation template. AWS CloudFormation hat ein festes Limit für die Anzahl der Ressourcen, die ein Stack enthalten kann. Mit der können AWS CDK Sie dieses Limit schneller als erwartet erreichen.

Note

Das AWS CloudFormation Ressourcenlimit beträgt 500 bei diesem Schreiben. Siehe [-AWS CloudFormation Kontingente](#) für das aktuelle Ressourcenlimit.

Die absichtsbasierten übergeordneten Konstrukte der AWS Construct Library stellen automatisch alle Hilfsressourcen bereit, die für die Protokollierung, Schlüsselverwaltung, Autorisierung und andere Zwecke benötigt werden. Wenn Sie beispielsweise einer Ressource Zugriff auf eine andere gewähren, werden alle IAM-Objekte generiert, die für die Kommunikation der relevanten Services erforderlich sind.

Unserer Erfahrung nach führt die reale Verwendung von absichtsbasierten Konstrukten zu 1–5 AWS CloudFormation Ressourcen pro Konstrukt, obwohl dies variieren kann. Für Serverless-Anwendungen sind 5–8 AWS Ressourcen pro API-Endpunkt typisch.

Mit Mustern, die eine höhere Abstraktionsstufe darstellen, können Sie noch mehr AWS Ressourcen mit noch weniger Code definieren. Der AWS CDK Code in generiert [the section called "ECS"](#)beispielsweise mehr als 50 AWS CloudFormation Ressourcen und definiert nur drei Konstrukte!

Das Überschreiten des AWS CloudFormation Ressourcenlimits ist ein Fehler während der AWS CloudFormation Synthetisierung. Die AWS CDK gibt eine Warnung aus, wenn Ihr

Stack 80 % des Limits überschreitet. Sie können ein anderes Limit verwenden, indem Sie die `-maxResources`Eigenschaft für Ihren Stack festlegen, oder die Validierung deaktivieren, indem Sie `maxResources` auf 0 setzen.

Tip

Mit dem folgenden Hilfsprogrammskript können Sie eine genaue Anzahl der Ressourcen in Ihrer synthetisierten Ausgabe abrufen. (Da jeder AWS CDK Entwickler Node.js benötigt, wird das Skript in geschrieben 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");
```

Wenn sich die Ressourcenanzahl Ihres Stacks dem Limit nähert, sollten Sie eine Neuarchitektur in Betracht ziehen, um die Anzahl der Ressourcen zu reduzieren, die Ihr Stack enthält, z. B. indem Sie einige Lambda-Funktionen kombinieren oder Ihren Stack in mehrere Stacks aufteilen. Das CDK unterstützt [Verweise zwischen Stacks](#), sodass Sie die Funktionalität Ihrer App in jeder Weise in verschiedene Stacks aufteilen können, die für Sie am sinnvollsten ist.

Note

AWS CloudFormation Experten schlagen häufig die Verwendung verschachtelter Stacks als Lösung für das Ressourcenlimit vor. unterstützt AWS CDK diesen Ansatz über das [NestedStack](#)Konstrukt.

[\(Zurück zur Liste von \)](#)

Ich habe drei (oder mehr) Availability Zones für meine Auto Scaling-Gruppe oder VPC angegeben, diese wurde jedoch nur in zwei bereitgestellt

Um die Anzahl der von Ihnen angeforderten Availability Zones abzurufen, geben Sie das Konto und die Region in der Stack-envEigenschaft an. Wenn Sie nicht beides angeben AWS CDK, synthetisiert das standardmäßig den Stack als umgebungsunabhängig. Anschließend können Sie den Stack mithilfe von in einer bestimmten Region bereitstellen AWS CloudFormation. Da einige Regionen nur zwei Availability Zones haben, verwendet eine umgebungsunabhängige Vorlage nicht mehr als zwei.

Note

In der Vergangenheit wurden -Regionen gelegentlich mit nur einer Availability Zone gestartet. Umgebungsunabhängige AWS CDK Stacks können nicht in solchen Regionen bereitgestellt werden. Zum Zeitpunkt dieses Schreibens haben alle AWS Regionen jedoch mindestens zwei AZs .

Sie können dieses Verhalten ändern, indem Sie die Stack-Eigenschaft [availabilityZones](#) (Python: `availability_zones`) überschreiben, um die Zonen, die Sie verwenden möchten, explizit anzugeben.

Weitere Informationen zur Angabe des Kontos und der Region eines Stacks zur Generierungszeit finden Sie unter [the section called "Umgebungen"](#).

[\(Zurück zur Liste\)](#)

Mein S3-Bucket, meine DynamoDB-Tabelle oder eine andere Ressource wird nicht gelöscht, wenn ich ausstelle **cdk destroy**

Standardmäßig haben Ressourcen, die Benutzerdaten enthalten können, eine `removalPolicy` (Python: `removal_policy`)-Eigenschaft von und die Ressource wird nicht gelöscht `RETAIN`, wenn der Stack zerstört wird. Stattdessen ist die Ressource aus dem Stack verwaist. Anschließend müssen Sie die Ressource manuell löschen, nachdem der Stack zerstört wurde. Bis dahin schlägt die erneute Bereitstellung des Stacks fehl. Dies liegt daran, dass der Name der neuen Ressource, die während der Bereitstellung erstellt wird, mit dem Namen der verwaisten Ressource in Konflikt steht.

Wenn Sie die Entfernungsrichtlinie einer Ressource auf festlegen `DESTROY`, wird diese Ressource gelöscht, wenn der Stack zerstört wird.

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 kann einen nicht leeren Amazon S3-Bucket nicht löschen. Wenn Sie die Entfernerichtlinie eines Amazon S3-Buckets auf festlegen und diese Daten enthält `DESTROY`, schlägt der Versuch, den Stack zu löschen, fehl, da der Bucket nicht

gelöscht werden kann. Sie können die Objekte im Bucket AWS CDK löschen lassen, bevor Sie versuchen, sie zu löschen, indem Sie die `autoDeleteObjects` -Eigenschaft des Buckets auf `true` setzen.

[\(Zurück zur Liste von \)](#)

OpenPGP-Schlüssel für die AWS CDK und jsii

Dieses Thema enthält aktuelle und historische OpenPGP-Schlüssel für die AWS CDK und jsii.

Aktuelle Schlüssel

Diese Schlüssel sollten verwendet werden, um aktuelle Versionen von AWS CDK und jsii zu validieren.

AWS CDK OpenPGP-Schlüssel

Schlüssel-ID:	0x42B9CF2286CD987A
Type:	RSA
Größe:	4096/4096
Erstellt:	2022-07-05
Läuft ab:	2026-07-04
Benutzer-ID:	AWS Cloud Development Kit <aws-cdk@amazon.com>
Schlüssel-Fingerabdruck:	69B5 2D5B A295 1D11 FA65 413B 42B9 CF22 86CD 987A

Wählen Sie das Symbol „Kopieren“, um den folgenden OpenPGP-Schlüssel zu kopieren:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
```

```
mQINBGLEg0sBEADCoAMwvnszMLybJ+AD9cHhVyX6+rYIUEXYSgVnfk16Z7qawIwv  
wgd/a5fEs9Kiz2XJmfwS9Rxb4d+0+Y11s1A+gnpw9FMLcZ1qkC9KLnS2MqvUXWLB  
t3z4kjZaL9fQ+58PoD4gy/M2hDg6gZrYqR3gtJuw8FcFpb/1K1kzRQUM8eAMFxf2  
TyfjP0V0tSHwCB+84oushX7fUXVMyc3+0HsCP0e/WBFMI1WgKA+n33JKIQ1UUC8f  
kCWBAAsAFupi101CveT6mZu5s1NR1c1I3iBLjUZ3/MtLygfgqAMkwUVXeawtDvRIZe  
PrAFc2Ny0DEhly2JG6K0FW7eIcvBqR3rg8U49t9Y74ELTM0kKnfd+f1vq35xWqQC  
0zghnk3kDppRTN4zWBgTKiCMxBcsHXG0oGn57t4B9VY9Zy3vkeySigeiw1/Tw9nJ
```

```

PE0SRnwEc/HnjTTFX+GTG1aQVE0xSVyZ4m5ymRNCu6+rNH81Kwo5FujlXJ+GXPkp
qT+Lx6Ix/Ny7PaoweWxwtZUKLRS4pWUsg0yotZrGyIbS+X3yMEG8WBTFI9hf6HTq
0ryfi5/TsBrdRgKqWB99EC9xYEGgtHp4fK05X0yn0agV0hf0jSe8t1uyuJPGb2Gc
MQagSys5xMhdG/ZnEY4Cb+JDtH/4j3tca0+4Z5RQ7kF9IhCncFtrbjJbwARAQAB
tC5BV1MgQ2xvdWQgRGV2ZWxvcG11bnQgS210IDxhd3MtY2RrQGFtYXpvbi5jb20+
iQI/BBMBAgApBQJixIDrAhsVBQkHhM4ABwsJCAcDAgEGFQgCCQoLBBYCAwECHgEC
F4AACgkQQ0rnPIobNmHo2qg//Zt9p/kN1Devf1zxWKouUX0AS7UmUtRYXu5k/EEbu
wkYNHPUr7+lZ+Me5YyjcIpt6UwuG9cW4SvwuxIfXucyKAWiwEbydCQauvnrYDxDa
J6Yr/ntk7Sii6An9re99qic3IsvX+x1UXh+qJ/34ooP/1PHziCMqykvW/DwAIyhx
2qvTXy+9+0l0WSUBhkCnNz5XKb4XQGq73DqaLZX1nH4dG6fckZmYRX+dpw2njfTw
ZLdZ7bkrfiL84FI4A21RfSbEU4s4ngiV17LZ9ivilBKTbDv3da7+yc919M7C5N4J
yr1xvtyYNDQKAD2WYZAnpEbG/shu3f56Ry0Jd56tXGwL9nKPh+F9y+379XthSwA
xZTURFtjWf7wWHaDZadU0DKi+0eeszjg2f/VJaGmmS8PIg7q6GiSHHpqHqNvACHm
ZXMw12QFd3qt3xu0JMmE11ZC5VBgblwpkQTr004Sq1r0pJwXI90DMS/ZEhAIoYmT
OR7oukn1Ax6mj9fwpavWDAAJHLdVUMYBZTXiQYFzDvx51ivvTRWkB1zTJcFdqShY
B37+Jz2jLDNdMrcHk2yfVp/VvfbxKcexg8wEwrrtQUslTUenl5jBZJouoz/wW81s
Y4U1nCPCdTK5/C7JCKzR2gVnCpe6uaxAWkkM2feQhjqJZkTC4cFVgBT+4M6WcT1r
yq4=
=ahbs
-----END PGP PUBLIC KEY BLOCK-----

```

jsii OpenPGP-Schlüssel

Schlüssel-ID:	0x056C4E15DAE3D8D9
Type:	RSA
Größe:	4096/4096
Erstellt:	2022-07-05
Läuft ab:	2026-07-04
Benutzer-ID:	AWS JSII Team <aws-jsii@amazon.com>
Schlüssel-Fingerabdruck:	1E07 31D4 57E5 FE87 87E5 530A 056C 4E15 DAE3 D8D9

Wählen Sie das Symbol „Kopieren“, um den folgenden OpenPGP-Schlüssel zu kopieren:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
```

```
mQINBGLEg0kBEAD27EPVG9g2mHQ3+M6tF61e+tfhARJ2EV7m7NKIrTdS1CZATLWn
AVL1xG1unW34N1kKZbcbR86gAxRnnAhuEhPuloU/S5wAqPGbRiF158YjYZDNJw6U
1SSMpE401sfjxv9yAbiRihLYtvksyHHZmaDhYner2aK1PdeWu+BKq/tjfm3Yzsd2
uuVEduJ72YoQk/29dEiG0HfT+2kUKxUX+0tJSJ9MG1Ef4NtQE4WLzrT6Xqb2SG4+
a1IiIVxIEi0XKDN7n8ZLjFwfJw0YxVYLtEUKqFWM8e8vgoc9/nYc+vDXZVED2g3Z
FwRwSnDSXbQpnMa2cLhD4xLpDHUS3i2p7r3dkJQGLo/5JG0opLibr0AbYZ72izhu
H/TuPFogSz0mNFPglrWdnLF04UIjIq420+06V4WQZC9n55Zjcbki/0hnC3B9pAdU
tiy8zq070bwQ45dPGf5STkPPn7G8A2zmKefy051iLi26ZzW78siB+FvcGRhdg25
39sHJ1cmrTeC+B+k4KeV5sQ/m3UucimrZnk1xdaiVp8mWzRqWb8bB6Rs8K9RMrMV
tFB0K0BAT2Qx0QtRGAantVgm193E1T1cmNpD0FKAKkDdPs64rKBewFiHxccXHbah
eMd1weVwn3AKFD6uAm8ZRMV+dyssfcQxqpo/kfT1XpA6cQe0mGD0cKBfdwARAQAB
tCNBV1MgS1NJSSBUZWFtIDxhd3MtanNpaUBhbWF6b24uY29tPokCPwQTAQIAKQUC
YsSA6QIbLwUJB4TOAAcLCQgHAWIBBhUIAgkKCwQWAgMBAh4BAheAAAoJEAVsThXa
49jZjU4QANoyq0JUT4gRrXshE3N0mW5Ad4i8Ke09GA62HyvTtfbsA+2nkNVGJpXm
sFMzdaF095Q65RkLS9vW4nhhjXBEC2XYNCt2AnARudA/41ykjDPwU112z9ZTB9he
y4ItIeNgpHvMWr51fihl0y2nkp0D0Beiv44jScLbHy0mZfki1f5fuIu2U2IbUGK3
5FtYyeHcgRHnpYkzLuzK4Pfay0yWqQPJ7M9DWrHf+v5Cu4ZCZD0IKfzF+ew7MWwc
6KaoWHCYbFpX8jxPpbGsSF0Q8S12quoP0TLz9Wsq70Khi6C2P8JI61m0HRL0+1M
jFbQxN0wAcN3k4HswunAjXB1mT/6oc1RsdBdpXBaZ2AWseIXwSYZqNXp+5L179uZ
vSiD3DSSUqLJbdQRV0sJi3/87V5QU59byq2dToHveRjtSbVnK0TkTx9Z1gkcpjvM
BwHNqWhratV6af2Upjq2YQ0fdSB42f3pgopInxNJPMv1Ab+cCfr0Pfwu7ge7UooQ
WHTxbpCvwn/HNctMGpWsc002WswgoYVjnVFay/XphE77pQ9rRUKhMe6VKXfxj/n
0CZJKrydluIIwR8vv0NNq0+QwZ1xDEh07MaSZ10m1AuUZIXFPgaWQkPZHkiwFA/
QWnL/+shuRtMH2geTjkev198Jgb5HyXFm4SyYtZferQR0yliEhik
=BuGv
-----END PGP PUBLIC KEY BLOCK-----
```

Historische Schlüssel

Diese Schlüssel können verwendet werden, um Releases von AWS CDK und jsii vor dem 2022-07-05 zu validieren.

Important

Neue Schlüssel werden erstellt, bevor die vorherigen ablaufen. Daher kann zu einem bestimmten Zeitpunkt mehr als ein Schlüssel gültig sein. Schlüssel werden verwendet, um Artefakte ab dem Tag ihrer Erstellung zu signieren. Verwenden Sie daher den kürzlich ausgestellten Schlüssel, bei dem sich die Gültigkeit von Schlüsseln überschneidet.

AWS CDK OpenPGP-Schlüssel (2022-04-07)

Note

Dieser Schlüssel wurde nach dem 2022-07-05 nicht zum Signieren von AWS CDK Artefakten verwendet.

Schlüssel-ID:	0x015584281F44A3C3
Type:	RSA
Größe:	4096/4096
Erstellt:	2022-04-07
Läuft ab:	2026-04-06
Benutzer-ID:	AWS Cloud Development Kit <aws-cdk@amazon.com>
Schlüssel-Fingerabdruck:	EAE1 1A24 82B0 AA86 456E 6C67 0155 8428 1F44 A3C3

Wählen Sie das Symbol „Kopieren“, um den folgenden OpenPGP-Schlüssel zu kopieren:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
```

```
mQINBGJPLgUBEADt1R5jQtxtBmR0QvmWlP0ViqqnJNhk0dULc3tXnq8NS/16X81r
wHk+/CHG5kBunwvM0qaqLFRC6z9NnnNDxEHcTi47n+0AjWyDM6unxxW0Pz8Dfaps
Uq/ZWa4by292ZeqRC9Ir2wdrizb69JbRjeshBwlJDAS/qtqCAqBRH/f7Zw7QSD6/
XTxyIy+K0VjZwFPFNHMRQ/NmgUc/Rfxsa0pUjk1YAj/AkvQ1wwD8DEnASoBh00DP
QonZxouLqIpgp4LsGo8TZdQv30ocIj0C9DuYUiuXWlCP1YPgDj6Iwf3rgpMQ6nB9
wC91x4t/L3Zg1HUD52y8aymndmbdHVn90mz1Ng4XWyc58rioYrEk57YwbDnea/Kk
Hv4kVHZRfJ4/0FPyqs5ex1X3X6rb07VvA1tflgPyw09XF2Xws8Yw0WcEobaWtcnb
AzyVC6wKya8rEQzXkYJ6UkJ1hDB6g6bZwIpsI2zlimG+kSBsyFvE2oRYMS0cXPqU
o+tX0+4TvxEyW3RrUQzQHIpqXrb0X1Q8Z2idPn5dwsipDEa4gsFXtrSXmbB/0Cee
eJVvKwQAsxo13+NE9L/yozq3cz5PWh0SSbmCLRcs781MJ23MmzbMWV7BWC9DXdY+
TywY5IkDUPjGCK1D8V1rI3TgC222bH6qaua6LYCiTtRtvpDYuJNA1UjhawARAQAB
tC5BV1MgQ2xvdWQgRGV2ZWxvcG11bnQgS2l0IDxhd3MtY2RrQGFTYXpvi5jb20+
```

```
iQI/BBMBAgApBQJiT4FAhsvBQkHhM4ABwsJCAcDAgEGFQgCCQoLBBYCAwEChgEC
F4AACgkQAVWEKB9Eo8NpbxAAiBF0kR/1Vw3vuam60mk410iGMVsP8Xq6g/buzbE0
2MEB4Ftk04q0noa+93S0ZiLR9PqxrsGS4ADDX3Vtc4uxwzU1KUi1ywEhQ1cwyL
YHQI3Hd75K1J81ozMEu6qJH+yF0TtTDZMeZtH/XvuIYJW3Lx4o5ZF1sEegFPAgX
YCCpUS+k9qC6M8g2VjcltQJpyjGswsKm6FWaKHW+B9dfjd0H1ImB9E2jaknJ8eoY
zb9zHgFANLuMzpZ6rYVSiCuXiEgYmazQWCv1PcM0P7nX+1hq1z11LMqeSnfE09gX
H+0Yho9cMEJkb1dZX1H9MRpy1FIIn9tL+2iCp4UPJjnqi6uawWyLZ2tp4G11haqQq
1yAh69u233I8GZKFUySszjHwH5qWGRgBTjrZ6FdcjSS2w/wMkVKuCPkWtdvo/TJrm
msCd1Reye8SEKYqrs0ujTwm1vWmUZm006AdUjo1kWiBKes1TJrWEuG7Yk4pF0oA4
dsaq83gxp0JNVCh6M3y4DLNrv17dhF95NwTWMR0Pj2otw7NIjF4/cdzve2+P7YNN
pVAtyCtTJdD3eZbQPVAL3T8cf1VGqt6++pnLGnWJ0+X3TyvfmTohdJvN3TE+tg7A
7cprDX/q9c56HaXdJzVpxEzuf/YC+JuYKeHwsX3QouDhyRg3PsigdZES/02Wtr8so
16U=
=MQI4
-----END PGP PUBLIC KEY BLOCK-----
```

jsii OpenPGP-Schlüssel (2022-04-07)

Note

Dieser Schlüssel wurde nach dem 2022-07-05 nicht zum Signieren von jsii-Artefakten verwendet.

Schlüssel-ID:	0x985F5BC974B79356
Type:	RSA
Größe:	4096/4096
Erstellt:	2022-04-07
Läuft ab:	2026-04-06
Benutzer-ID:	AWS JSII Team <aws-jsii@amazon.com>
Schlüssel-Fingerabdruck:	35A7 1785 8FA6 282D C5AC CD95 985F 5BC9 74B7 9356

Wählen Sie das Symbol „Kopieren“, um den folgenden OpenPGP-Schlüssel zu kopieren:


```

-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGJPLewBEADHH4TXup/g01HrKDZRbj8MvsMTdM6eDteA6/c32UYV/YsK9rDA
jN8Jv/x1fos0ebcHrfnFpHF9VTkmjuOpN695XdwMrW/Nv1EPISTGEJf21x6ZTQ2r
1xWfYzC3s13FZmvj9XAXTmygdv+XM3TqsFgZeCaBkZVdiLbQf+FhYrovU1gotb5D
YiCQI3ofV5QTE+141jh05Pkd3ZIoBG+P826LaT8NXhwS0o1XqVk39DCZNoFshNmR
WFZpkVCTHyv5ZhVey1NWXnD8op0375htGNV4AeSmSIH9YkURD1g5F+2t7RiosKFo
kJrfPmUjhHn8IFpReGc8qmMMZX0WaV3t+VAwf0HGGyrXdfQ4xz1VCot75C2+qypM
+qhw0A00P0zA7CfI96ULZzSH/j8HuQk300DsUCybpMuKEazEMxP3tgGtRerwDaFG
jQvAlK8Rbq3v8buBI6YJuXTwSzJE8KLjleUiTFumE6WP4rsAv1P/5rBvubeMfa3n
NIMm5Rk136Z+jt3e2Z2ZqWDPpBRta8m7QHccrZhkvqu3YC3G16kdnm4Vio3Xfpg2
qtWhIQutQ6DmItewV+weQHas3h188RPJtSrfWIIIMkpbF7Y4vbX9xcnsYCLlp2Mz
tWbbnU+EWATNSsufm1/Kdnu9iEEuLmeovE11I69nwjN0q9P+GJ3r/FUB2wARAQAB
tCNBV1MgS1NjSSBUZWFtIDxhd3MtanNpaUBhbWF6b24uY29tPokCPwQTAQIAKQUC
Yk8t7AIbLwUJB4T0AAcLCQgHAWIBBUiAgkKCwQWAgMBAh4BAheAAoJEJhfW810
t5NWo64P/2y7gcMRy1LLW/wbrCjton204+YRocwQxKm1cBm19FVDUR5967YczNuu
EwE0fH/Pu3UALrBfKafxPNhKchLwYi0BNh2Wk5UUXRcldNHTLb5jn5gxCeWNA5l/
Tc46qY+0bdBMD0f2Vu33UC0g83WLbg1bfBoA8Bm1cd0X0btLGucu606EBt1dBkKq
9UTcbJfuGivY2Xjy5r4kEiMHBolKcFrSo2Mm7VtY1E4Mabjyj9+orqUio7qx0160
aa7Psa6rMvs1Ip9I0rAdG7o5Y29tQpeINH0R1/u47Br1TEAgG63Dfy49w2h/1g0G
c9KPXVuN550WRiU0hhsiySDMK/2ERsF348TU3NURZ1tnC0xp6pH1bPJIxRVTNa9Cn
f8tbLB3y3HfA80516g+qwNYIYiqksDdV2bz+VbvmCwC0+Fe11DZ1i831gyMGa5JJ
rq7d01Er6nqjcnKiVwItTQXyFYmKTAXweQtVC72g1sd3oZIyqa7T8pvhWpKXxoJV
WP+OPBhGg/JEVC9sguhuv53tzVwayrNwb54JxJsD2nemfhQm1Wyvb2bPTEaJ3mrV
mhPUvXZj/I9rqsEq3L/sm2Xjy09nra4o3oe3bhEL8n0j11wkIodi17VaGP0y+H3s
I5zB5UztS6dy+cH+J7DoRaxzVzq7qtH/ZY2quCl30wwqDHUX1ef
=iYX
-----END PGP PUBLIC KEY BLOCK-----

```

AWS CDK OpenPGP-Schlüssel (2018-06-19)

Schlüssel-ID:	0x0566A784E17F3870
Type:	RSA
Größe:	4096/4096
Erstellt:	2018-06-19
Läuft ab:	2022-06-18
Benutzer-ID:	AWS-CDK-Team <aws-cdk@amazon.com>

Schlüssel-Fingerabdruck:

E88B E3B6 F0B1 E350 9E36 4F96 0566 A784
E17F 3870

Wählen Sie das Symbol „Kopieren“, um den folgenden OpenPGP-Schlüssel zu kopieren:

-----BEGIN PGP PUBLIC KEY BLOCK-----

```
mQINBFsovE8BEADEFVChEAVPvoQgsjVu9FPUCzxy9P+2zGIT/MLI3/vPLiULQwRy
IN2oxyBNDtcdToNa/ftkV3Ev0NTP4V1h+uBoKDZD/p+dTmSDRfByECMI0sGZ3UsG
0hhy120f44s0sL8gdLtDnqSRLf+ZrfT3gpgUnplW7VltkWLxr78jDpW4QD8p8dZ9
WNm3JgB55jyPgaJKqA1Ln4Vduni/1XkrG42nxrrU71uUdZPvPZ2ELLJa6n0/raG8
jq3le+xQh45gAIs6PGaAgy7jAsfbwkGTBhjjujITAY1DwvQH5iS310aCM9n4JNpc
xGZeJAVYTLilzfnf2QtS/a50t+Z0mpq67Ssp2j6qYpiumm0Lo9q3K/R4/yF0FZ8SL
1TuNX0ecXEptiMVUfTiqrLsANg18EPtLZZ0YW+ZkbcVytkDpiqj7bMwA7mI7zGCJ
1gjaTbcEm0mVdQYS1G6ZptwbTtvrgA6AfnZxX1HUxLRQ7tT/wvRtABfbQKAh85Ff
a3U9W4oC3c1MP5IyhNV1Wo8Zm0flZiZc0iZnojTtSG6UbcxNNL4Q8e08FWjhungj
yxSsIBnQ01Aeo1N4Bbz1I+n9iaXVDUN7Kz1QEYs4PNpjuvUyrUiQ+a9C5sRA7WP+x
IE0aBBGpoAXB3oLsdTN06AcwcDd9+r2N1X1hWC4/uH2YHQUIegPqHmPwxwARAQAB
tCFBV1MgQ0RLIFRlYW0gPGF3cy1jZGtAYW1hem9uLmNvbT6JAj8EEwEIAckFA1so
vE8CGy8FCQeEzgaHCwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIXgAAKCRAFZqeE4X84
cLGxD/0XHNhoR2xvz38GM8HQ1w1Zy9W1wVhQKmNDQUavw8Zx7+iRR3m7nq3xM7Qq
BDbcbKSg11VLSBQ6H2V6vRpys0hkPSH1nN2d08DtvSKIPcxK48+1x7lm0+ksSs/+
oo1Uv0mTDaRz0itYh3k0GXHHXk/111GtF2FGQzYssX5iM4PHcjBsK1unThs56IMh
0JeZezEYzBaskTu/ytRJ236bPP2kZIExfzAvhmTytuXWUXEftx0xc6fIACyikTha
aofG7Wyr+Fvb1j5gNLcbY552QMxa23NZd5cSZH7468WEW1SGJ3AdLA7k5xvsPP0C
2YvQFD+vU0Z1JJuu6B5rHkiEMhRTLk1kvqXESHTxuXiCp7iT0o6TBCmrWAT4eQr7
htLmq1XrgKi8qPkWmRdXXG+MQBzI/UyZq2q8KC6cx2md1PhANmeeFhiM7FZZfeNM
WLonWfh8gVCsNH5h8WJ9fxsQCADD3Xxx3Ne1S2zDYBPRoaqZEEBbgUP6LnWFprA2
EkSlc/RoDqZCpBGgcoy1FFWvV/ZLgNU60TQ1YH6oY0Wiy1SjNaTDyurktsxJI6d
4gdsFb6tqwTGecuUPvvZaEuvhWEXLxAbhu780FdAPXgVTX+YCLi2zf+dWQvkFQf
80RE7ayn7BsialzFBVux/zz/WgvudsZX18r8tDiVQBL510Rmqw==
=0wuQ
```

-----END PGP PUBLIC KEY BLOCK-----

jsii OpenPGP-Schlüssel (2018-08-06)

Schlüssel-ID:

0x1C7ACE4CB2A1B93A

Type:

RSA

Größe:	4096/4096
Erstellt:	2018-08-06
Läuft ab:	2022-08-05
Benutzer-ID:	AWS JSII Team <aws-jsii@amazon.com>
Schlüssel-Fingerabdruck:	85EF 6522 4CE2 1E8C 72DB 28EC 1C7A CE4C B2A1 B93A

Wählen Sie das Symbol „Kopieren“, um den folgenden OpenPGP-Schlüssel zu kopieren:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
```

```
mQINBFtoSs0BEAD6WweLD0B26h0F7Jo9iR6tVQ4PgQBK1Va5H/eP+A2Iqw79UyxZ
WNzHYhzQ5MjYYI1SgcPavXy5/LV1N8HJ7QzyKszybnLYpNTPYArWE8ZM9ZmjvIR
p1GzwnVBGQfo0lxyeutE9T5ZkAn45dTS5jln04unji4gHjnwXKf2nP1APU2CZfdK
8vDpL0gj9LeeGlerYNbx+7xtY/I+csFIQvK09FPLSNMJQLkBy0r6Rt9ZQG+653
tJn+AUjyM237w0UIX1IqyYc5I0NXu8Hk1PGu0NYuX9AY/63Ak2Cyfj0w/PZ1vueQ
noQNM3j0nk0EsT0EXCyaLQw9iBKpxvLnm5RjMS0DDCkj8c9uu0LHr7J4E0tgt2S1
pem7Y/c/N+/Z+Ksg9fP8fVTfYwRPvdI1x2sCiRDfLoQSG9tdrN5VwPFi4sGV04sI
x7A18Vf/0BjAGZrDaJgM/gVvb9SKAQUA6t3ofeP14gDrS0eYodEXZ+lamnxFglx
Sn8NRC4JFNmkXSUAtnGUdFf//F0D69PRNT8CnFfmniGj0CphN5037PCA2LC/Buq2
3+K6mTPkCcCHYPC/SwItp/xIDAQsGuDc1i1SfDYXrjsK7u0uwC5jLA9X6wZ/jgXQ
4umRRJBAV1aW8b1+yfaYYC02AfXX06ca0bv8IvH7Pc4leC2Doqy1D3Kk1QARAQAB
tCNBV1MgS1NJSSBUZWFtIDxhd3MtanNpaUBhbWF6b24uY29tPokCPwQTAQgAKQUC
W2hKzQIbLwUJB4TOAAcLCQgHAWIBBhUIAgkKCwQWAgMBAh4BAheAAAoJEBx6zkyy
obk6B34P/iNb5QjKyhT0glZiq1wK7tuDDRpR6fC/sp6Jd/GhaNj04Bz1DbUPSjW5
950VT+qwaHXbIma/QVP7EIRztfwWy7m8e0odjpiu7JyJprhwG9nocXiNsLADcMoH
BvabkDRWXWIWSurq2wbcFMlTVwxjHPIQs6kt2oojPzP985CDS/KTzyjow6/gfMim
DLdhSSbDUM34STEGew79L2sQzL7cvM/N59k+AGyEMHZDXHkEw/Bge50vz50Y0nsp
lisH4BzPRIw7uWqPlkVPzJKwMuo2WvMjDfgbYLbyjfv5mqDxT2GTwAx/rd2taU6
iSqP0QmLM54BtTVVdoVXZSmJyTmXAAG1ITq8ECZ/coUW9K2pUSgVuWyu631ktFP6
MyCQYRmXPh9aSd4+ielteXM9Y39snlyLgEJBhMxioZXV02oszwluPuhPoAp4ekwj
/umVsBf6As6PoAchg7Qzr+1RZGmV9YTJ0gDn2Z7jf/7t0es0g/mdiXTQMSGtp/Fp
ggNifTBx3iXkrQhqlHwtam8XTHGHY3MvX17Zs1NuB8Pjh+07hhCvx0VUVZPUHJqJ
ZsLa398LMteQ8UMxwJ3t06jwDwAd7mbr2tatIiLLHtWWBFoCwBh1XLe/03ENCpDp
njZ70sBsBK2nVvcN0H2v5ey0T1yE93o6r7x0wCwBiVp5skTCRJob
=2Tag
```

```
-----END PGP PUBLIC KEY BLOCK-----
```

AWS CDK Verlauf des -Entwicklerhandbuchs

Weitere Informationen zu [-Versionen](#) finden Sie unter [- AWS CDK Versionen](#). Die AWS CDK wird etwa einmal pro Woche aktualisiert. Wartungsversionen können zwischen wöchentlichen Versionen veröffentlicht werden, um kritische Probleme zu beheben. Jede Version enthält ein passendes AWS CDK Toolkit (CDK CLI), eine AWS Konstruktbibliothek und eine API-Referenz. Aktualisierungen dieses Handbuchs werden im Allgemeinen nicht mit [- AWS CDK Versionen](#) synchronisiert.

Note

Die folgende Tabelle stellt wichtige Meilensteine der Dokumentation dar. Wir beheben Fehler und verbessern Inhalte kontinuierlich.

Änderung	Beschreibung	Datum
Dokumentation für CDK-Migrate-Feature hinzufügen	Verwenden Sie den AWS CDK CLI <code>cdk migrate</code> Befehl , um bereitgestellte AWS Ressourcen, bereitgestellte AWS CloudFormation Stacks und lokale AWS CloudFormation Vorlagen zu migrieren AWS CDK. Weitere Informationen finden Sie unter Migrieren zu AWS CDK .	2. Februar 2024
Aktualisierungen der bewährten Methoden für IAM	Aktualisierter Leitfaden, angepasst an die bewährten IAM-Methoden. Weitere Informationen finden Sie unter Bewährte IAM-Methoden .	23. März 2023

Dokument cdk.json	Fügen Sie Dokumentation der cdk.json Konfigurationswerte hinzu.	20. April 2022
Abhängigkeitsverwaltung	Fügen Sie das Thema zum Verwalten von Abhängigkeiten mit dem hinzu AWS CDK.	7. April 2022
Entfernen von doppelten Klammern aus Java-Beispielen	Ersetzen Sie dieses Anti-Muster Map.of durch Java 9.	9. März 2022
AWS CDK Version v2	Version 2 des - AWS CDK Entwicklerhandbuchs wurde veröffentlicht. Dokumentverlauf für CDK v1.	4. Dezember 2021

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.