개발자 가이드

AWS Cloud Development Kit (AWS CDK) v2

버전 2

AWS Cloud Development Kit (AWS CDK) v2: 개발자 가이드

Table of Contents

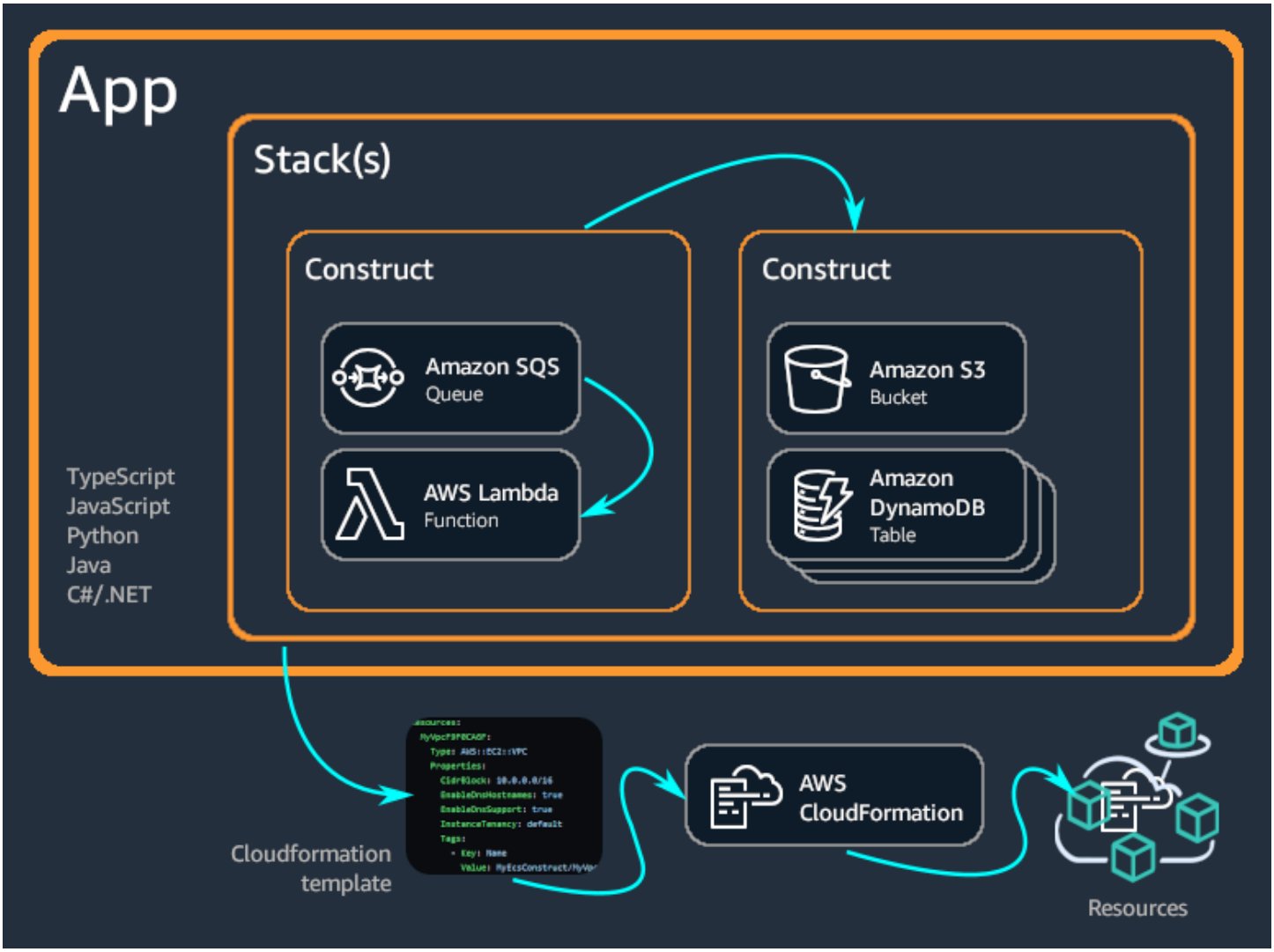
이게 뭐야 AWS CDK?

코드로 클라우드 인프라를 정의하고 이를 통해 프로비저닝하기 위한 오픈 소스 소프트웨어 개발 AWS Cloud Development Kit (AWS CDK) 프레임워크입니다. AWS CloudFormation

두 가지 주요 AWS CDK 부분으로 구성되어 있습니다.

- AWS CDK Construct Library — 미리 작성된 모듈식 및 재사용 가능한 코드 (구성이라고 함) 의 모음으로, 이를 사용, 수정 및 통합하여 인프라를 빠르게 개발할 수 있습니다. AWS CDK Construct Library의 목표는 애플리케이션을 구축할 때 AWS 서비스를 정의하고 통합하는 데 필요한 복잡성을 줄이는 것입니다. AWS
- AWS CDK 툴킷 — CDK 앱과 상호작용하기 위한 명령줄 도구입니다. AWS CDK 툴킷을 사용하여 프로젝트를 생성, 관리, 배포할 수 있습니다. AWS CDK

AWS CDK 는TypeScript,, JavaScript Python JavaC#/.Net, 및 Go 를 지원합니다. 지원되는 이러한 프로그래밍 언어를 사용하여 구조라고 하는 재사용 가능한 클라우드 구성 요소를 정의할 수 있습니다. 이들을 함께 스택과 앱으로 구성합니다. 그런 다음 CDK 애플리케이션을 AWS CloudFormation 배포하여 리소스를 프로비저닝하거나 업데이트합니다.



주제

- 의 이점 AWS CDK
- 예시 AWS CDK
- AWS CDK features
- 다음 단계
- 자세히 알아보기

의 이점 AWS CDK

AWS CDK 를 사용하여 프로그래밍 언어의 뛰어난 표현력을 갖춘 안정적이고 확장 가능하며 비용 효율적인 애플리케이션을 클라우드에서 개발할 수 있습니다. 이 접근 방식은 다음과 같은 많은 이점을 제공합니다.

코드형 인프라 (IaC) 개발 및 관리

코드로서의 인프라를 연습하여 프로그래밍, 설명 및 선언적 방식으로 인프라를 생성, 배포 및 유지 관리하세요. IaC를 사용하면 개발자가 코드를 다루는 것과 같은 방식으로 인프라를 취급할 수 있습니다. 그 결과 인프라 관리에 대한 확장 가능하고 구조화된 접근 방식이 가능해집니다. IaC에 대해 자세히 알아보려면 AWS 백서 소개에서 코드로서의 인프라를 참조하십시오. DevOps

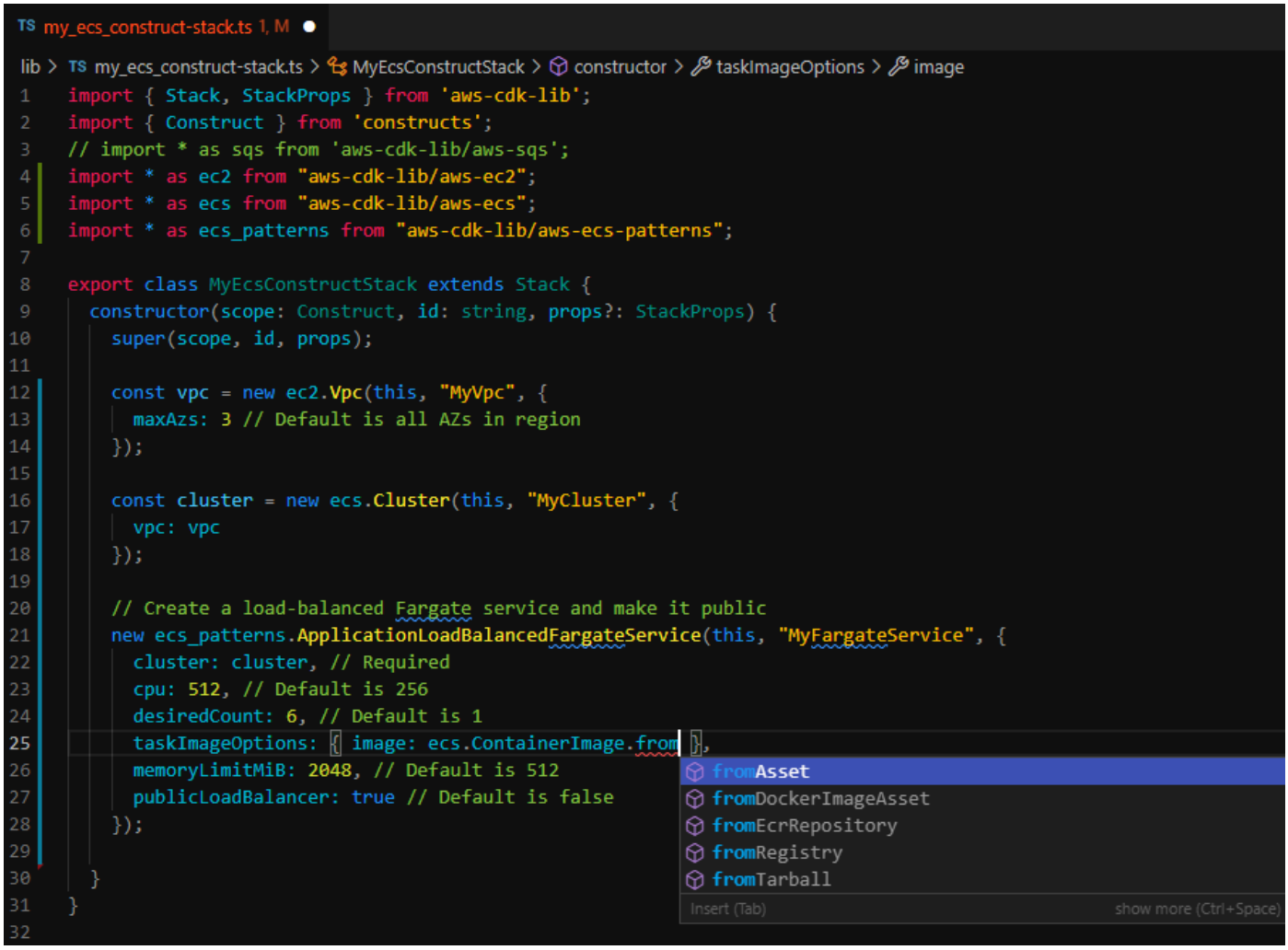
를 사용하면 인프라 AWS CDK, 애플리케이션 코드 및 구성을 모두 한 곳에 모아 모든 마일스톤에서 클라우드로 배포할 수 있는 완전한 시스템을 확보할 수 있습니다. 코드 검토, 단위 테스트, 소스 제어와 같은 소프트웨어 엔지니어링 베스트 프랙티스를 활용하여 인프라를 더욱 견고하게 만드세요.

범용 프로그래밍 언어를 사용하여 클라우드 인프라를 정의하십시오.

를 사용하면 AWS CDK,,,, 및 등의 프로그래밍 언어를 사용하여 클라우드 인프라를 정의할 수 있습니다. TypeScript JavaScript Python Java C#/.Net Go 선호하는 언어를 선택하고 매개변수, 조건부, 루프, 구성, 상속 등의 프로그래밍 요소를 사용하여 인프라의 원하는 결과를 정의하십시오.

동일한 프로그래밍 언어를 사용하여 인프라와 애플리케이션 로직을 정의하십시오.

선호하는 IDE (통합 개발 환경) 에서 구문 강조 표시 및 지능형 코드 완성과 같은 인프라 개발의 이점을 누리십시오.



를 통해 인프라를 배포하십시오. AWS CloudFormation

AWS CDK 와 AWS CloudFormation 통합하여 인프라를 배포 및 프로비저닝합니다. AWS AWS CloudFormation 서비스 프로비저닝을 위한 리소스 및 속성 구성을 광범위하게 AWS 서비스 지원하는 관리형입니다. AWS를 사용하면 롤백 오류가 발생하여 예측 가능하고 반복적으로 인프라 배포를 수행할 수 있습니다. AWS CloudFormation이미 잘 알고 있다면 처음 AWS CloudFormation시작할 때 새로운 IaC 관리 서비스를 배울 필요가 없습니다. AWS CDK

구문을 사용하여 애플리케이션 개발을 빠르게 시작하세요.

구조라고 하는 재사용 가능한 구성 요소를 사용하고 공유하여 더 빠르게 개발하세요. 저수준 구조를 사용하여 개별 AWS CloudFormation 리소스와 해당 속성을 정의할 수 있습니다. 높은 수준의 구조를 사용하면 AWS 리소스에 대해 합리적이고 안전한 기본값을 사용하여 애플리케이션의 더 큰 구성 요소를 빠르게 정의하고 더 적은 코드로 더 많은 인프라를 정의할 수 있습니다.

고유한 사용 사례에 맞게 사용자 지정된 자체 구문을 만들어 조직 전체 또는 대중과 공유하세요.

예시 AWS CDK

다음은 AWS CDK 구성 라이브러리를 사용하여 시작 유형의 Amazon Elastic Container Service (Amazon ECS) 서비스를 생성하는 예제입니다. AWS Fargate (Fargate) 이 예제에 대한 자세한 내용은 을 참조하십시오. 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

}
```

이 클래스는 500줄이 넘는 AWS CloudFormation 템플릿을 생성합니다. AWS CDK 앱을 배포하면 다음과 같은 유형의 리소스가 50개 이상 생성됩니다.

- 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

리포지토리 AWS CDKGitHub

공식 AWS CDK GitHub 리포지토리에 대해서는 aws-cdk를 참조하십시오. 여기에서 이슈를 제출하고, 라이선스를 보고, 릴리스를 추적하는 등의 작업을 할 수 있습니다.

오픈 소스이기 때문에 팀에서는 여러분이 이 도구를 더 나은 도구로 만드는 데 기여할 것을 권장합니다. AWS CDK 자세한 내용은 기여를 참조하십시오. AWS Cloud Development Kit (AWS CDK)

AWS CDK API 레퍼런스

AWS CDK 구성 라이브러리는 CDK 애플리케이션을 정의하고 애플리케이션에 CDK 구문을 추가하기 위한 API를 제공합니다. 자세한 내용은 AWS CDK API 참조를 참조하십시오.

컨스트럭트 프로그래밍 모델

구성 프로그래밍 모델 (CPM) 은 기본 개념을 추가 AWS CDK 영역으로 확장합니다. CPM을 사용하는 기타 도구는 다음과 같습니다.

- 테라폼용 CDK (cdkTf)
- 쿠버네티스를 위한 CDK (CDK8)
- 프로젝트, 프로젝트 구성 구축용

컨스트럭트 허브

Construct Hub는 오픈 소스 AWS CDK 라이브러리를 찾고, 게시하고, 공유할 수 있는 온라인 레지스트리입니다.

다음 단계

사용을 시작하려면 AWS CDK을 참조하십시오시작하기 AWS CDK.

자세히 알아보기

에 대해 계속 AWS CDK알아보려면 다음을 참조하십시오.

- AWS CDK 개념— 에 대한 중요한 개념 및 용어 AWS CDK.
- AWS CDK 워크숍 — 배우고 사용할 수 있는 실습 워크숍. AWS CDK
- AWS CDK 패턴 — 전문가가 제작한 AWS 서버리스 아키텍처 패턴의 오픈 소스 컬렉션입니다. AWS CDK AWS
- AWS CDK 코드 예제 — 예제 AWS CDK 프로젝트의 GitHub 리포지토리.
- cdk.dev — 커뮤니티 기반 허브로, 커뮤니티 작업 공간을 포함합니다. AWS CDKSlack
- 멋진 CDK — AWS CDK 오픈 소스 프로젝트, 가이드, 블로그 및 기타 리소스의 엄선된 목록이 들어 있는 GitHub 저장소입니다.
- AWS 솔루션 구성 — 검증된 IaC (코드형 인프라) 패턴으로 제작이 가능한 애플리케이션으로 쉽게 조합할 수 있는 코드형 인프라 (IaC) 패턴
- AWS 개발자 도구 블로그 — 필터링된 블로그 게시물입니다. AWS CDK
- AWS CDK on Stack Overflow — aws-cdk로 태그가 지정된 질문 켜짐 Stack Overflow
- AWS CDK 튜토리얼 AWS Cloud9 — 개발 환경에서 를 사용하는 방법에 대한 자습서. AWS CDK AWS Cloud9

와 관련된 주제에 대한 자세한 내용은 AWS CDK다음을 참조하십시오.

- AWS CloudFormation 개념 — AWS CDK 은 (와) 함께 AWS CloudFormation사용할 수 있도록 만들어졌으므로 주요 AWS CloudFormation 개념을 배우고 이해하는 것이 좋습니다.
- AWS 용어집 — 여러 곳에서 AWS사용되는 주요 용어에 대한 정의

서버리스 애플리케이션 개발 및 배포를 단순화하는 데 사용할 수 AWS CDK 있는 관련 도구에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Serverless Application Model— 서버리스 애플리케이션을 구축하고 실행하는 경험을 단순화하고 개선하는 오픈 소스 개발자 도구입니다. AWS
- AWSChalice— 서버리스 앱을 작성하기 위한 프레임워크입니다. Python

AWS CDK 개념

그 이면에 있는 핵심 개념을 알아보십시오 AWS Cloud Development Kit (AWS CDK).

AWS CDK 그리고 IaC

코드를 사용하여 AWS 인프라를 관리하는 데 사용할 수 있는 오픈 소스 AWS CDK 프레임워크입니다. 이 접근 방식을 코드로서의 인프라 (IaC) 라고 합니다. 인프라를 코드로 관리하고 프로비전하면 개발자가 코드를 다루는 것과 같은 방식으로 인프라를 취급할 수 있습니다. 이는 버전 제어 및 확장성과 같은 많은 이점을 제공합니다. IaC에 대해 자세히 알아보려면 코드형 인프라란? 을 참조하십시오.

AWS CDK 그리고 AWS CloudFormation

와 AWS CDK 긴밀하게 통합되어 있습니다. AWS CloudFormation AWS CloudFormation 인프라를 관리하고 프로비저닝하는 데 사용할 수 있는 완전 관리형 AWS서비스입니다. AWS CloudFormation 를 사용하면 템플릿에서 인프라를 정의하고 배포할 수 AWS CloudFormation있습니다. 그러면 AWS CloudFormation 서비스가 템플릿에 정의된 구성에 따라 인프라를 프로비저닝합니다.

AWS CloudFormation 템플릿은 선언적이므로 인프라의 원하는 상태 또는 결과를 선언합니다. JSON 또는 YAML을 사용하여 리소스와 속성을 정의하여 AWS 인프라를 선언합니다. AWS 리소스는 많은 서비스를 AWS 나타내며 속성은 해당 서비스의 원하는 구성을 나타냅니다. 템플릿을 에 AWS CloudFormation배포하면 리소스와 구성된 속성이 템플릿에 설명된 대로 프로비저닝됩니다.

를 사용하면 AWS CDK범용 프로그래밍 언어를 사용하여 인프라를 필수적으로 관리할 수 있습니다. 원하는 상태를 선언적으로 정의하는 대신 원하는 상태에 도달하는 데 필요한 로직이나 시퀀스를 정의할 수 있습니다. 예를 들어 인프라의 원하는 최종 상태에 도달하는 방법을 결정하는 `if` 명령문이나 조건부 루프를 사용할 수 있습니다.

를 사용하여 만든 AWS CDK 인프라는 결국 변환되거나 AWS CloudFormation 템플릿으로 합성되어 AWS CloudFormation 서비스를 사용하여 배포됩니다. 따라서 에서는 인프라 생성 시 다른 접근 방식을 AWS CDK 제공하지만 광범위한 AWS 리소스 구성 지원 및 강력한 배포 프로세스와 같은 이점은 여전히 누릴 수 있습니다. AWS CloudFormation

자세히 AWS CloudFormation알아보려면 What is AWS CloudFormation? 를 참조하십시오. AWS CloudFormation 사용 설명서에서.

AWS CDK 및 추상화

AWS CloudFormation를 사용하면 리소스 구성 방식의 모든 세부 사항을 정의해야 합니다. 이를 통해 인프라를 완벽하게 제어할 수 있다는 이점이 있습니다. 하지만 이를 위해서는 리소스 구성 세부 정보와 권한 및 이벤트 기반 상호 작용과 같은 리소스 간 관계를 포함하는 강력한 템플릿을 배우고 이해하고 만들어야 합니다.

AWS CDK를 사용하면 리소스 구성을 동일하게 제어할 수 있습니다. 그러나 강력한 추상화 AWS CDK 기능도 제공하므로 인프라 개발 프로세스를 가속화하고 단순화할 수 있습니다. 예를 들어, AWS CDK 여기에는 적절한 기본 구성을 제공하는 구문과 보일러플레이트 코드를 자동으로 생성하는 도우미 메서드가 포함됩니다. AWS CDK 또한 인프라 관리 작업을 대신 수행하는 AWS CDK 명령줄 인터페이스 (AWS CDK CLI) 와 같은 도구도 제공합니다.

핵심 AWS CDK 개념에 대해 자세히 알아보십시오.

와의 상호 작용 AWS CDK

와 함께 사용할 때는 주로 AWS 구성 라이브러리 및 와 상호 작용합니다. AWS CDK AWS CDK CLI

를 사용하여 개발하기 AWS CDK

지원되는 모든 프로그래밍 언어로 작성할 AWS CDK 수 있습니다. 먼저 에셋을 포함한 폴더 및 파일 구조를 포함하는 CDK 프로젝트로 시작합니다. 프로젝트 내에서 CDK 애플리케이션을 생성합니다. 앱 내에서 스택을 정의하는데, 스택은 CloudFormation 스택을 직접 나타냅니다. 스택 내에서 구문을 사용하여 AWS 리소스와 속성을 정의합니다.

를 사용하여 배포하기 AWS CDK

CDK 앱을 AWS 환경에 배포합니다. 배포하기 전에 일회성 부트스트래핑을 수행하여 환경을 준비해야 합니다.

자세히 알아보기

AWS CDK 핵심 개념에 대해 자세히 알아보려면 이 섹션의 항목을 참조하십시오.

지원되는 프로그래밍 언어

AWS Cloud Development Kit (AWS CDK) 는 다음과 같은 범용 프로그래밍 언어를 최고 수준으로 지원합니다.

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

다른 JVM .NET CLR AND 언어도 이론적으로 사용될 수 있지만 현재로서는 공식 지원을 제공하지 않습니다.

> **Note**
>
> 이 가이드에는 현재 Go 외에 대한 지침이나 코드 예제가 포함되어 있지 않습니다the section called “In Go”.

AWS CDK 는 한 가지 언어로 개발되었습니다TypeScript. 다른 언어를 지원하기 위해 에서는 라는 AWS CDK JSII도구를 사용하여 언어 바인딩을 생성합니다.

AWS CDK 최대한 자연스럽고 직관적인 방식으로 개발할 수 있도록 각 언어의 일반적인 규칙을 제공하려고 합니다. 예를 들어, 사용자가 선호하는 언어의 표준 리포지토리를 사용하여 AWS Construct Library 모듈을 배포하고, 사용자는 해당 언어의 표준 패키지 관리자를 사용하여 설치합니다. 메서드와 속성의 이름도 해당 언어의 권장 이름 지정 패턴을 사용하여 지정됩니다.

다음은 몇 가지 코드 예제입니다.

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**
>
> 이 코드 스니펫은 설명을 위한 용도로만 사용됩니다. 불완전하며 원래대로 실행되지 않습니다.

AWS 구성 라이브러리는,, NPM PyPiMaven, 등 각 언어의 표준 패키지 관리 도구를 사용하여 배포됩니다. NuGet 또한 각 언어에 대한 AWS CDK API 참조 버전도 제공합니다.

원하는 언어로 를 사용할 수 있도록 이 안내서에는 지원되는 언어에 대한 다음 항목이 포함되어 있습니다. AWS CDK

- the section called “에서 TypeScript”
- the section called “에서 JavaScript”
- the section called “Python에서”
- the section called “자바에서”
- the section called “C #에서”
- the section called “In Go”

TypeScript는 에서 가장 먼저 지원한 AWS CDK언어이며 AWS CDK 예제 코드는 대부분 이 언어로 작성되었습니다TypeScript. 이 가이드에는 지원되는 다른 언어와 함께 사용할 수 있도록 TypeScript AWS CDK 코드를 조정하는 방법을 보여주는 특정 항목이 포함되어 있습니다. 자세한 내용은 다른 AWS CDKTypeScript 언어와의 비교을(를) 참조하세요.

AWS CDK 프로젝트

AWS Cloud Development Kit (AWS CDK) 프로젝트는 CDK 코드가 들어 있는 파일 및 폴더를 나타냅니다. 내용은 프로그래밍 언어에 따라 달라집니다.

AWS CDK 프로젝트를 수동으로 만들거나 AWS CDK 명령줄 인터페이스 (AWS CDK CLI) `cdk init` 명령을 사용하여 만들 수 있습니다. 이 주제에서는 AWS CDK CLI로 생성된 파일 및 폴더의 프로젝트 구조와 명명 규칙을 참조합니다. 필요에 맞게 CDK 프로젝트를 사용자 지정하고 구성할 수 있습니다.

> **Note**
>
> 에서 만든 프로젝트 구조는 시간이 지남에 따라 버전마다 AWS CDK CLI 다를 수 있습니다.

주제

- 범용 파일 및 폴더
- 언어별 파일 및 폴더

범용 파일 및 폴더

.git

를 `git` 설치한 경우 는 프로젝트의 Git 저장소를 AWS CDK CLI 자동으로 초기화합니다. `.git`디렉토리에는 저장소에 대한 정보가 들어 있습니다.

.gitignore

에서 무시할 파일 및 폴더를 지정하는 Git 데 사용되는 텍스트 파일입니다.

README.md

AWS CDK 프로젝트 관리를 위한 기본 지침과 중요한 정보를 제공하는 텍스트 파일입니다. 필요에 따라 이 파일을 수정하여 CDK 프로젝트에 관한 중요한 정보를 기록하십시오.

cdk.json

의 구성 파일. AWS CDK이 파일은 앱 실행 방법에 AWS CDK CLI 대한 지침을 제공합니다.

언어별 파일 및 폴더

다음 파일 및 폴더는 지원되는 각 프로그래밍 언어에 고유합니다.

TypeScript

다음은 `cdk init --language typescript` 명령을 사용하여 `my-cdk-ts-project` 디렉터리에 만든 예제 프로젝트입니다.

```
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

패키지를 레지스트리에 게시할 때 무시할 파일 및 폴더를 지정하는 파일입니다. npm 이 파일은 패키지와 `.gitignore` 비슷하지만 npm 패키지에만 해당됩니다.

bin/ .ts my-cdk-ts-project

애플리케이션 파일은 CDK 앱을 정의합니다. CDK 프로젝트에는 하나 이상의 애플리케이션 파일이 포함될 수 있습니다. 애플리케이션 파일은 `bin` 폴더에 저장됩니다.

다음은 CDK 앱을 정의하는 기본 애플리케이션 파일의 예입니다.

```
#!/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

에 대한 구성 파일. Jest Jest널리 사용되는 JavaScript 테스트 프레임워크입니다.

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

스택 파일은 CDK 스택을 정의합니다. 스택 내에서 구문을 사용하여 AWS 리소스와 속성을 정의합니다.

다음은 CDK 스택을 정의하는 기본 스택 파일의 예시입니다.

```
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
 }
}
```

노드_모듈

Node.js프로젝트에 대한 종속성을 포함하는 프로젝트의 공통 폴더입니다.

package-lock.json

파일과 함께 작동하여 종속성 버전을 관리하는 메타데이터 `package.json` 파일입니다.

package.json

프로젝트에서 일반적으로 사용되는 메타데이터 파일입니다. Node.js 이 파일에는 프로젝트 이름, 스크립트 정의, 종속성 및 기타 가져오기 프로젝트 수준 정보와 같은 CDK 프로젝트에 대한 정보가 들어 있습니다.

테스트/ my-cdk-ts-project .test.ts

테스트 폴더는 CDK 프로젝트의 테스트를 정리하기 위해 생성됩니다. 샘플 테스트 파일도 생성됩니다.

테스트를 실행하기 전에 테스트를 작성하고 TypeScript 코드를 Jest 컴파일하는 데 사용할 수 있습니다. TypeScript

tsconfig.json

컴파일러 옵션과 프로젝트 설정을 지정하는 TypeScript 프로젝트에 사용되는 구성 파일입니다.

JavaScript

다음은 `cdk init --language javascript` 명령을 사용하여 `my-cdk-js-project` 디렉터리에 만든 예제 프로젝트입니다.

```
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

패키지를 레지스트리에 게시할 때 무시할 파일 및 폴더를 지정하는 파일입니다. npm 이 파일은 패키지와 `.gitignore` 비슷하지만 npm 패키지에만 해당됩니다.

bin/ .js my-cdk-js-project

애플리케이션 파일은 CDK 앱을 정의합니다. CDK 프로젝트에는 하나 이상의 애플리케이션 파일이 포함될 수 있습니다. 애플리케이션 파일은 `bin` 폴더에 저장됩니다.

다음은 CDK 앱을 정의하는 기본 애플리케이션 파일의 예입니다.

```
#!/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

에 대한 구성 파일. Jest Jest널리 사용되는 JavaScript 테스트 프레임워크입니다.

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

스택 파일은 CDK 스택을 정의합니다. 스택 내에서 구문을 사용하여 AWS 리소스와 속성을 정의합니다.

다음은 CDK 스택을 정의하는 기본 스택 파일의 예시입니다.

```
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 }
```

노드_모듈

Node.js프로젝트에 대한 종속성을 포함하는 프로젝트의 공통 폴더입니다.

package-lock.json

파일과 함께 작동하여 종속성 버전을 관리하는 메타데이터 `package.json` 파일입니다.

package.json

프로젝트에서 일반적으로 사용되는 메타데이터 파일입니다. Node.js 이 파일에는 프로젝트 이름, 스크립트 정의, 종속성 및 기타 가져오기 프로젝트 수준 정보와 같은 CDK 프로젝트에 대한 정보가 들어 있습니다.

테스트/ my-cdk-js-project .test.js

테스트 폴더는 CDK 프로젝트의 테스트를 정리하기 위해 생성됩니다. 샘플 테스트 파일도 생성됩니다.

테스트를 실행하기 전에 테스트를 작성하고 JavaScript 코드를 Jest 컴파일하는 데 사용할 수 있습니다. JavaScript

Python

다음은 `cdk init --language python` 명령어를 사용하여 `my-cdk-py-project` 디렉터리에 만든 예제 프로젝트입니다.

```
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

CDK는 프로젝트를 위한 가상 환경을 CLI 자동으로 생성합니다. `.venv`디렉토리는 이 가상 환경을 가리킵니다.

app.py

애플리케이션 파일은 CDK 앱을 정의합니다. CDK 프로젝트에는 하나 이상의 애플리케이션 파일이 포함될 수 있습니다.

다음은 CDK 앱을 정의하는 기본 애플리케이션 파일의 예시입니다.

```
#!/usr/bin/env python3
import os

import aws_cdk as cdk

from my_cdk_py_project.my_cdk_py_project_stack import MyCdkPyProjectStack

app = cdk.App()
MyCdkPyProjectStack(app, "MyCdkPyProjectStack")

app.synth()
```

my_cdk_py_project

스택 파일이 들어 있는 디렉터리 CLICDK는 여기에 다음을 생성합니다.

- `__init__.py` — 빈 Python 패키지 정의 파일.
- `my_cdk_py_project`— CDK 스택을 정의하는 파일입니다. 그런 다음 구문을 사용하여 스택 내의 AWS 리소스와 속성을 정의합니다.

다음은 스택 파일의 예시입니다.

```
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

파일과 `requirements.txt` 비슷하지만 특히 프로덕션이 아닌 개발 목적으로 종속성을 관리하는 데 사용되는 파일입니다.

requirements.txt

Python프로젝트에서 프로젝트 종속성을 지정하고 관리하는 데 사용되는 공통 파일입니다.

source.bat

Windows이를 위한 Batch 파일은 Python 가상 환경을 설정하는 데 사용됩니다.

테스트

CDK 프로젝트를 위한 테스트가 들어 있는 디렉터리입니다.

다음은 유닛 테스트의 예시입니다.

```
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

다음은 `cdk init --language java` 명령을 사용하여 `my-cdk-java-project` 디렉터리에 만든 예제 프로젝트입니다.

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

pom.xml

CDK 프로젝트에 대한 구성 정보와 메타데이터가 들어 있는 파일입니다. 이 파일은 의 일부입니다. Maven

src/main

애플리케이션 및 스택 파일이 들어 있는 디렉터리

다음은 예제 애플리케이션 파일입니다.

```
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();
 }
```

```
}
```

다음은 예제 스택 파일입니다.

```
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

테스트 파일이 들어 있는 디렉터리 다음은 그 예제입니다.

```
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#

다음은 `cdk init --language csharp` 명령을 사용하여 `my-cdk-csharp-project` 디렉터리에 만든 예제 프로젝트입니다.

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

src/ MyCdkCsharpProject

애플리케이션과 스택 파일이 들어 있는 디렉터리

다음은 예제 애플리케이션 파일입니다.

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

다음은 예제 스택 파일입니다.

```
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
  }
 }
}
```

이 디렉토리에는 다음 항목도 포함되어 있습니다.

- `GlobalSuppressions.cs`— 프로젝트 전체에서 특정 컴파일러 경고나 오류를 억제하는 데 사용되는 파일입니다.
- `.csproj`— 프로젝트 설정, 종속성 및 빌드 구성을 정의하는 데 사용되는 XML 기반 파일입니다.

MyCdkCsharpProjectsrc/ .sln

Microsoft Visual Studio Solution File관련 프로젝트를 구성하고 관리하는 데 사용됩니다.

Go

다음은 `cdk init --language go` 명령을 사용하여 `my-cdk-go-project` 디렉터리에 만든 예제 프로젝트입니다.

```
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

모듈 정보를 포함하고 프로젝트의 종속성 및 버전 관리를 관리하는 데 사용되는 파일입니다. Go

my-cdk-go-project.go

CDK 애플리케이션 및 스택을 정의하는 파일입니다.

다음은 그 예제입니다.

```
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

샘플 테스트를 정의하는 파일입니다.

다음은 그 예제입니다.

```
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 앱

AWS Cloud Development Kit (AWS CDK) 애플리케이션 또는 앱은 하나 이상의 CDK 스택으로 구성된 컬렉션입니다. 스택은 리소스와 속성을 정의하는 AWS 하나 이상의 구문의 모음입니다. 따라서 스택과 구조를 전체적으로 그룹화하는 것을 CDK 앱이라고 합니다.

주제

앱 정의

프로젝트의 애플리케이션 파일에서 앱 인스턴스를 정의하여 앱을 생성합니다. 이렇게 하려면 `App` AWS 구성 라이브러리에서 구문을 가져와 사용합니다. 이 `App` 구문에는 초기화 인수가 필요하지 않습니다. 루트로 사용할 수 있는 유일한 구문입니다.

AWS 구성 라이브러리의 `App` 및 `Stack` 클래스는 고유한 구조입니다. 다른 구문에 비해 자체적으로 AWS 리소스를 구성하지는 않습니다. 대신 다른 구문에 컨텍스트를 제공하는 데 사용됩니다. AWS 리소스를 나타내는 모든 구문은 구문 범위 내에서 직접 또는 간접적으로 정의되어야 합니다. `Stack` `Stack`구문은 구문 범위 내에서 정의됩니다. `App`

그런 다음 앱을 합성하여 스택용 AWS CloudFormation 템플릿을 만듭니다. 다음은 그 예제입니다.

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)
```

단일 앱 내의 스택은 서로의 리소스 및 속성을 쉽게 참조할 수 있습니다. 스택을 올바른 순서로 배포할 수 있도록 스택 간의 AWS CDK 종속성을 유추합니다. 단일 명령으로 앱 내에 스택 일부 또는 전체를 배포할 수 있습니다. `cdk deploy`

구성 트리

구문은 App 클래스를 루트로 하여 모든 구문에 전달되는 scope 인수를 사용하여 다른 구문 내에서 정의됩니다. 이런 방식으로 AWS CDK 앱은 구성 트리라고 하는 구문의 계층 구조를 정의합니다.

이 트리의 루트는 App 클래스의 인스턴스인 앱입니다. 앱 내에서 하나 이상의 스택을 인스턴스화합니다. 스택 내에서 구문을 인스턴스화합니다. 구문은 자체적으로 리소스 또는 기타 구문을 인스턴스화하는 식입니다.

구문은 항상 다른 구문의 범위 내에서 명시적으로 정의되므로 구문 간에 관계가 생성됩니다. 거의 항상 `this` (Python에서는`self`) 를 범위로 전달해야 합니다. 이는 새 구문이 현재 구문의 자식임을 나타냅니다. 의도한 패턴은 구문을 파생한 다음 생성자에서 사용하는 구문을 인스턴스화하는 것입니다. `Construct`

범위를 명시적으로 전달하면 각 구문이 트리에 자신을 추가할 수 있으며, 이 동작은 기본 클래스 내에 완전히 포함됩니다. `Construct` 이 방법은 에서 지원하는 모든 언어에서 동일한 방식으로 AWS CDK 작동하며 추가 사용자 지정이 필요하지 않습니다.

> **Important**
>
> 엄밀히 따지자면 구문을 인스턴스화할 `this` 때 이외에 다른 범위를 전달할 수 있습니다. 트리의 아무 곳에나, 심지어는 같은 앱의 다른 스택에도 구문을 추가할 수 있습니다. 예를 들어 인수로 전달된 범위에 구문을 추가하는 믹스인 스타일 함수를 작성할 수 있습니다. 여기서 현실적으로 어려운 점은 구문에 대해 선택한 ID가 다른 사람의 범위 내에서 고유한지 쉽게 확인할 수 없다는 것입니다. 또한 이렇게 하면 코드를 이해하고 유지 관리하고 재사용하기가 더 어려워집니다. 논증을 악용하지 않고 의도를 표현할 방법을 찾는 것이 거의 항상 더 좋습니다. `scope`

AWS CDK 는 트리의 루트에서 각 하위 구문까지의 경로에 있는 모든 구문의 ID를 사용하여 에서 요구하는 고유한 ID를 생성합니다. AWS CloudFormation이 접근 방식은 구성 ID가 AWS CloudFormation 네이티브처럼 전체 스택 내에서 고유하지 않고 해당 범위 내에서만 고유해야 한다는 것을 의미합니다. 하지만 구문을 다른 범위로 이동하면 생성된 스택 고유 ID가 변경되고 동일한 리소스로 AWS CloudFormation 간주되지 않습니다.

구문 트리는 코드에 정의한 구문과는 별개입니다. AWS CDK 하지만 모든 구문의 `node` 속성, 즉 트리에서 해당 구문을 나타내는 노드에 대한 참조를 통해 액세스할 수 있습니다. 각 노드는 `Node`인스턴스이며, 각 노드의 속성을 통해 트리의 루트와 노드의 부모 범위 및 자식에 액세스할 수 있습니다.

1. `node.children`— 구문의 직계 하위입니다.
2. `node.id`— 해당 범위 내 구문의 식별자.
3. `node.path`— 모든 상위의 ID를 포함한 구문의 전체 경로.
4. `node.root`— 구성 트리의 루트 (앱).
5. `node.scope`— 구문의 범위 (부모) 이며, 노드가 루트인 경우 정의되지 않습니다.
6. `node.scopes`— 구조체의 모든 부모 (루트까지)
7. `node.uniqueId`— 트리 내 이 구문의 고유한 영숫자 식별자 (기본적으로 `node.path` 및 해시에서 생성됨).

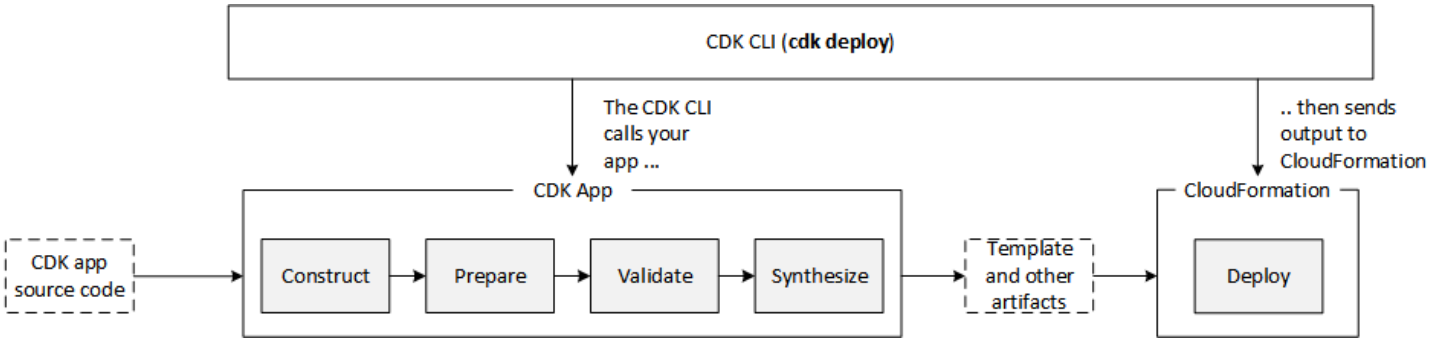
구성 트리는 구문이 최종 템플릿의 리소스에 합성되는 암시적 순서를 정의합니다. AWS CloudFormation 한 리소스가 다른 리소스보다 먼저 생성되어야 하는 경우, AWS CloudFormation 그렇지 않으면 일반적으로 AWS 구성 라이브러리가 종속성을 유추합니다. 그런 다음 리소스가 올바른 순서로 생성되었는지 확인합니다.

를 사용하여 두 노드 사이에 명시적 종속성을 추가할 수도 있습니다. `node.addDependency()` 자세한 내용은 API 참조의 AWS CDK 종속성을 참조하십시오.

는 구성 트리의 모든 노드를 방문하여 각 노드에 대해 작업을 수행할 수 있는 간단한 방법을 AWS CDK 제공합니다. 자세한 설명은 the section called “속성” 섹션을 참조하세요.

앱 라이프사이클

CDK 앱을 배포하면 다음 단계가 수행됩니다. 이를 앱 라이프사이클이라고 합니다.



AWS CDK 앱은 수명 주기에서 다음 단계를 거칩니다.

- 구성 (또는 초기화) — 코드는 정의된 모든 구문을 인스턴스화한 다음 함께 연결합니다. 이 단계에서는 모든 구문 (앱, 스택, 하위 구문) 이 인스턴스화되고 생성자 체인이 실행됩니다. 대부분의 앱 코드는 이 단계에서 실행됩니다.
- 준비 — `prepare` 메서드를 구현한 모든 구문은 최종 수정 단계에 참여하여 최종 상태를 설정합니다. 준비 단계는 자동으로 진행됩니다. 사용자는 이 단계에서 어떤 피드백도 볼 수 없습니다. “준비” 후크를 사용할 필요는 거의 없으며 일반적으로 권장되지 않습니다. 이 단계에서 구성 트리를 변경할 때는 작업 순서가 동작에 영향을 줄 수 있으므로 매우 주의해야 합니다.
- 유효성 검사 — `validate` 메서드를 구현한 모든 구문은 올바르게 배포될 수 있는 상태인지 확인하기 위해 자체 유효성을 검사할 수 있습니다. 이 단계에서 발생하는 모든 검증 실패에 대한 알림을 받게 됩니다. 일반적으로 가능한 한 빨리 (보통 입력이 들어오는 즉시) 검증을 수행하고 가능한 한 빨리 예외를 발생시키는 것이 좋습니다. 검증을 조기에 수행하면 스택 추적이 더 정확해지기 때문에 안정성이 향상되고 코드가 안전하게 계속 실행될 수 있습니다.
- 합성 — AWS CDK 앱 실행의 마지막 단계입니다. 호출에 의해 트리거되며`app.synth()`, 구성 트리를 탐색하고 모든 구문에서 `synthesize` 메서드를 호출합니다. 구현하는 구문은 합성에 참여하고

결과 클라우드 어셈블리에 배포 아티팩트를 내보낼 `synthesize` 수 있습니다. 이러한 아티팩트에는 AWS CloudFormation 템플릿, AWS Lambda 애플리케이션 번들, 파일 및 Docker 이미지 자산, 기타 배포 아티팩트가 포함됩니다. the section called “클라우드 어셈블리”이 단계의 출력을 설명합니다. 대부분의 경우 `synthesize` 메서드를 구현하지 않아도 됩니다.

- 배포 — 이 단계에서는 합성 단계에서 생성된 배포 아티팩트 클라우드 어셈블리를 AWS CDK CLI 가져와 환경에 배포합니다. AWS Amazon S3와 Amazon ECR 또는 필요한 곳 어디에나 자산을 업로드합니다. 그런 다음 AWS CloudFormation 배포를 시작하여 애플리케이션을 배포하고 리소스를 생성합니다.

AWS CloudFormation 배포 단계가 시작될 무렵에는 AWS CDK 앱이 이미 완료되고 종료된 상태입니다. 이는 다음과 같은 의미를 가집니다.

- AWS CDK 앱은 배포 중에 발생하는 이벤트 (예: 리소스 생성 또는 전체 배포 완료) 에 응답할 수 없습니다. 배포 단계에서 코드를 실행하려면 코드를 AWS CloudFormation 템플릿에 사용자 지정 리소스로 삽입해야 합니다. 앱에 사용자 지정 리소스를 추가하는 방법에 대한 자세한 내용은 AWS CloudFormation 모듈 또는 사용자 지정 리소스 예제를 참조하십시오.
- AWS CDK 앱이 실행 당시에는 알 수 없는 값으로 작동해야 할 수도 있습니다. 예를 들어 AWS CDK 앱이 자동으로 생성된 이름으로 Amazon S3 버킷을 정의하고 사용자가 `bucket.bucketName` (Python:`bucket_name`) 속성을 검색하는 경우 해당 값은 배포된 버킷의 이름이 아닙니다. 대신 `Token` 값을 얻게 됩니다. 특정 값을 사용할 수 있는지 확인하려면 `cdk.isUnresolved(value)` (Python:`is_unresolved`) 를 호출하십시오. 세부 정보는 the section called “토큰”를 참조하세요.

클라우드 어셈블리

를 `app.synth()` 호출하면 앱에서 클라우드 어셈블리를 AWS CDK 합성하도록 지시합니다. 일반적으로 클라우드 어셈블리와 직접 상호 작용하지 않습니다. 앱을 클라우드 환경에 배포하는 데 필요한 모든 것이 포함된 파일입니다. 예를 들어 앱의 각 스택에 대한 AWS CloudFormation 템플릿이 포함되어 있습니다. 또한 앱에서 참조하는 모든 파일 자산 또는 Docker 이미지의 사본도 포함됩니다.

클라우드 어셈블리 형식 지정 방법에 대한 자세한 내용은 클라우드 어셈블리 사양을 참조하십시오.

AWS CDK 앱에서 생성하는 클라우드 어셈블리와 상호 작용하려면 일반적으로 를 사용합니다. AWS CDK CLI 하지만 클라우드 어셈블리 형식을 읽을 수 있는 모든 도구를 사용하여 앱을 배포할 수 있습니다.

앱 실행

CDK는 AWS CDK 앱 실행 방법을 CLI 알아야 합니다. `cdk init`명령어를 사용하여 템플릿에서 프로젝트를 만든 경우 앱 `cdk.json` 파일에 `app` 키가 포함됩니다. 이 키는 앱이 작성된 언어에 필요한 명령을 지정합니다. 언어에 컴파일이 필요한 경우 앱을 실행하기 전에 명령줄에서 이 단계를 수행하므로 잊지 말고 실행해야 합니다.

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"
```

```
}
```

CLICDK를 사용하여 프로젝트를 만들지 않았거나 에서 cdk.json 지정한 명령줄을 재정의하려는 경우 명령을 실행할 때 --app 옵션을 사용할 수 있습니다. cdk

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

명령의 ## ### 부분은 CDK 애플리케이션을 실행하기 위해 실행해야 하는 명령을 나타냅니다. 이러한 명령에는 공백이 포함되므로 표시된 대로 따옴표를 사용하십시오. cdk-### ### 수행하려는 작업을 CLI CDK에 deploy 알려주는 synth OR와 같은 하위 명령입니다. 이 뒤에 해당 하위 명령에 필요한 추가 옵션을 모두 포함하세요.

또한 이미 AWS CDK CLI 합성된 클라우드 어셈블리와 직접 상호 작용할 수도 있습니다. 이렇게 하려면 클라우드 어셈블리가 저장된 디렉터리를 전달해야 합니다. --app 다음 예제는 저장되어 있는 클라우드 어셈블리에 정의된 스택을 나열합니다. ./my-cloud-assembly

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

스택

AWS Cloud Development Kit (AWS CDK) 스택은 리소스를 정의하는 AWS 하나 이상의 구문의 모음입니다. 각 CDK 스택은 CDK AWS CloudFormation 앱의 스택을 나타냅니다. 배포 시 스택 내의 구조는 스택이라는 단일 단위로 프로비저닝됩니다. AWS CloudFormation AWS CloudFormation 스택에 대해 자세히 알아보려면 사용 설명서의 스택 작업을 참조하십시오.AWS CloudFormation

CDK 스택은 스택을 통해 구현되므로 AWS CloudFormation 할당량과 제한이 적용됩니다. AWS CloudFormation 자세히 알아보려면 할당량을 참조하십시오.AWS CloudFormation

주제

- 스택 정의
- 스택 작업

스택 정의

스택은 앱 컨텍스트 내에서 정의됩니다. AWS 구성 라이브러리의 Stack 클래스를 사용하여 스택을 정의합니다. 스택은 다음 방법 중 하나로 정의할 수 있습니다.

- 앱 범위 내에서 직접.
- 트리 내의 모든 구문을 통해 간접적으로

다음 예제는 스택이 두 개 포함된 CDK 앱을 정의합니다.

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();
```

다음 예제는 별도의 파일에 스택을 정의하는 일반적인 패턴입니다. 여기서는 `Stack` 클래스를 확장하거나 상속하고, 및 를 허용하는 `scope` 생성자를 정의합니다. `id` `props` 그런 다음 수신된, 및 를 `super` 사용하여 기본 `Stack` 클래스 생성자를 호출합니다. `scope` `id` `props`

TypeScript

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

    //...
  }
}
```

JavaScript

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

    //...
  }
}
```

Python

```
class HelloCdkStack(Stack):

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

```
        # ...
```

Java

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

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

        // ...
    }
}
```

C#

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

Go

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

  return stack
}
```

다음 예제는 Amazon S3 버킷 하나를 `MyFirstStack` 포함하는 이름이 지정된 스택 클래스를 선언합니다.

TypeScript

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

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

JavaScript

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

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

Python

```
class MyFirstStack(Stack):

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

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

Java

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

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

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

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

C#

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

Go

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

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

하지만 이 코드에서는 스택만 선언했습니다. 스택을 실제로 AWS CloudFormation 템플릿으로 합성하여 배포하려면 인스턴스화해야 합니다. 또한 모든 CDK 구문과 마찬가지로 일부 컨텍스트에서 인스턴스화해야 합니다. 이것이 바로 그 컨텍스트입니다. App

표준 AWS CDK 개발 템플릿을 사용하는 경우 스택은 객체를 인스턴스화하는 동일한 파일에서 인스턴스화됩니다. App

TypeScript

프로젝트 폴더에 있는 프로젝트 이름을 딴 파일 (예:hello-cdk.ts). bin

JavaScript

프로젝트 `bin` 폴더에 있는 프로젝트 이름을 딴 파일 (예:`hello-cdk.js`).

Python

프로젝트의 기본 디렉터리에 `app.py` 있는 파일입니다.

Java

예를 들어 *`ProjectName`*`App.java`HelloCdkApp.java, 이름이 지정된 파일은 `src/main` 디렉터리 깊숙이 중첩되어 있습니다.

C#

예를 들어 `src\`*`ProjectName`*`src\HelloCdk\Program.cs`, `Program.cs` under 라는 이름의 파일입니다.

스택 API

Stack 객체는 다음을 포함한 다양한 API를 제공합니다.

- `Stack.of(construct)`— 구문이 정의된 Stack을 반환하는 정적 메서드입니다. 이는 재사용 가능한 구조 내에서 스택과 상호 작용해야 하는 경우에 유용합니다. 스코프에서 스택을 찾을 수 없는 경우 호출이 실패합니다.
- `stack.stackName`(Python:`stack_name`) — 스택의 물리적 이름을 반환합니다. 앞서 언급했듯이 모든 AWS CDK 스택에는 합성 중에 AWS CDK 확인할 수 있는 물리적 이름이 있습니다.
- `stack.region`및 `stack.account` — 이 스택을 배포할 AWS 지역과 계정을 각각 반환합니다. 이러한 속성은 다음 중 하나를 반환합니다.
  - 스택이 정의될 때 명시적으로 지정된 계정 또는 지역
  - 계정 및 지역의 AWS CloudFormation 유사 매개변수로 확인되는 문자열 인코딩 토큰으로, 이 스택이 환경에 구애받지 않는다는 것을 나타냅니다.

  스택의 환경을 결정하는 방법에 대한 자세한 내용은 을 참조하십시오. the section called “환경”
- `stack.addDependency(stack)`(Python: `stack.add_dependency(stack)` — 두 스택 간의 종속성 순서를 명시적으로 정의하는 데 사용할 수 있습니다. 여러 스택을 한 번에 배포할 때는 cdk deploy 명령이 이 순서를 따릅니다.
- `stack.tags`— 스택 수준 태그를 추가하거나 제거하는 데 사용할 수 TagManager있는 a를 반환합니다. 이 태그 관리자는 스택 내의 모든 리소스에 태그를 지정하고 스택이 생성될 때 스택 자체에도 태그를 지정합니다. AWS CloudFormation

- `stack.partition`, `stack.urlSuffix` (Python:`url_suffix`), `stack.stackId` (Python:`stack_id`), `stack.notificationArn` (Python:`notification_arn`) — 각 가상 파라미터 (예:) 로 AWS CloudFormation 확인되는 토큰을 반환합니다. `{ "Ref": "AWS::Partition" }` 이러한 토큰은 특정 스택 개체와 연결되므로 AWS CDK 프레임워크에서 스택 간 참조를 식별할 수 있습니다.
- `stack.availabilityZones`(Python:`availability_zones`) — 이 스택이 배포된 환경에서 사용 가능한 가용 영역 세트를 반환합니다. 환경에 구애받지 않는 스택의 경우, 이렇게 하면 항상 두 개의 가용 영역이 있는 배열이 반환됩니다. 환경별 스택의 경우 환경을 AWS CDK 쿼리하여 지정한 지역에서 사용 가능한 정확한 가용 영역 세트를 반환합니다.
- `stack.parseArn(arn)`and `stack.formatArn(comps)` (Python:`parse_arn`,`format_arn`) — Amazon 리소스 이름 (ARN) 을 사용하는 데 사용할 수 있습니다.
- `stack.toJsonString(obj)`(Python:`to_json_string`) — 임의의 객체를 템플릿에 포함할 수 있는 JSON 문자열로 포맷하는 데 사용할 수 있습니다. AWS CloudFormation 객체에는 토큰, 속성 및 참조가 포함될 수 있으며, 이러한 토큰, 속성 및 참조는 배포 중에만 확인됩니다.
- `stack.templateOptions`(Python:`template_options`) — 스택의 AWS CloudFormation 템플릿 옵션 (예: 변환, 설명, 메타데이터) 을 지정하는 데 사용합니다.

스택 작업

CDK 앱의 모든 스택을 나열하려면 명령을 사용합니다. cdk ls 이전 예제의 출력은 다음과 같습니다.

```
stack1
stack2
```

스택은 AWS CloudFormation 스택의 일부로 AWS 환경에 배포됩니다. 환경에는 특정 AND가 포함됩니다 AWS 계정 . AWS 리전

여러 스택이 있는 앱에 대해 cdk synth 명령을 실행하면 클라우드 어셈블리에 각 스택 인스턴스에 대한 별도의 템플릿이 포함됩니다. 두 스택이 같은 클래스의 인스턴스인 경우에도 는 두 개의 개별 AWS CDK 템플릿으로 내보냅니다.

명령에서 스택 이름을 지정하여 각 템플릿을 합성할 수 있습니다. cdk synth 다음 예제는 stack1의 템플릿을 합성합니다.

```
$ cdk synth stack1
```

이 접근 방식은 AWS CloudFormation 템플릿을 여러 번 배포하고 매개 변수를 통해 매개 변수화 할 수 있는 일반적인 템플릿 사용 방식과 개념적으로 다릅니다.AWS CloudFormation 에서 AWS CloudFormation 매개 변수를 정의할 수 있지만 매개 변수는 AWS CDK배포 중에만 확인되므로 AWS CloudFormation 일반적으로 사용하지 않는 것이 좋습니다. 즉, 코드에서는 해당 값을 확인할 수 없습니다.

예를 들어 매개변수 값을 기반으로 앱에 리소스를 조건부로 포함하려면 조건을 설정하고 해당 AWS CloudFormation 조건으로 리소스에 태그를 지정해야 합니다. 는 합성 시 구체적인 템플릿을 해석하는 접근 방식을 AWS CDK 취합니다. 따라서 if 문을 사용하여 값을 검사하여 리소스를 정의해야 하는지 아니면 일부 동작을 적용해야 하는지를 결정할 수 있습니다.

> **Note**
>
> 는 합성 시간 동안 최대한 많은 해상도를 AWS CDK 제공하므로 프로그래밍 언어를 관용적이고 자연스럽게 사용할 수 있습니다.

다른 구조와 마찬가지로 스택도 그룹으로 구성할 수 있습니다. 다음 코드는 제어 플레인, 데이터 플레인, 모니터링 스택의 세 가지 스택으로 구성된 서비스의 예를 보여줍니다. 서비스 구조는 두 번 정의됩니다. 한 번은 베타 환경용이고 다른 한 번은 프로덕션 환경용입니다.

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();
    }
}
```

이 AWS CDK 앱은 결국 각 환경에 3개씩 총 6개의 스택으로 구성됩니다.

```
$ cdk ls
```

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

AWS CloudFormation 스택의 물리적 이름은 트리의 스택 구성 경로를 AWS CDK 기반으로 에 의해 자동으로 결정됩니다. 기본적으로 스택 이름은 `Stack` 개체의 구성 ID에서 파생됩니다. 하지만 다음과 같이 `stackName` prop (Python에서는`stack_name`) 를 사용하여 명시적인 이름을 지정할 수 있습니다.

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"
});
```

중첩 스택

이 NestedStack구조는 스택의 AWS CloudFormation 리소스 500개 제한을 우회하는 방법을 제공합니다. 중첩된 스택은 스택에서 해당 스택을 포함하는 하나의 리소스로만 계산됩니다. 하지만 추가 중첩 스택을 포함하여 최대 500개의 리소스를 포함할 수 있습니다.

중첩된 스택의 범위는 or `Stack` 구문이어야 합니다. `NestedStack` 중첩 스택은 부모 스택 내에서 어휘적으로 선언할 필요가 없습니다. 중첩된 스택을 인스턴스화할 때는 부모 스택을 첫 번째 매개 변수 (`scope`) 로 전달하기만 하면 됩니다. 이러한 제한을 제외하면, 중첩된 스택에서 구문을 정의하는 것은 일반 스택과 완전히 동일하게 작동합니다.

합성 시 중첩된 스택은 자체 AWS CloudFormation 템플릿에 합성되며, 이 템플릿은 배포 시 AWS CDK 스테이징 버킷에 업로드됩니다. 중첩된 스택은 상위 스택에 바인딩되며 독립 배포 아티팩트로 취급되지 않습니다. 목록에 없으며 `cdk list` 배포할 수도 없습니다. `cdk deploy`

부모 스택과 중첩 스택 간의 참조는 스택 간 참조와 마찬가지로 생성된 AWS CloudFormation 템플릿의 스택 매개 변수 및 출력으로 자동 변환됩니다.

> **Warning**
>
> 중첩된 스택의 경우 배포 전에는 보안 상태의 변경 사항이 표시되지 않습니다. 이 정보는 최상위 스택에만 표시됩니다.

구조물

구조는 애플리케이션의 기본 구성 요소입니다. AWS Cloud Development Kit (AWS CDK) 구문은 하나 이상의 AWS CloudFormation 리소스와 해당 구성을 나타내는 애플리케이션 내의 구성 요소입니다. 구문을 가져오고 구성하여 애플리케이션을 하나씩 빌드합니다.

구문은 CDK 앱으로 가져오는 클래스입니다. 구문은 구성 라이브러리에서 사용할 수 있습니다. AWS 자체 구문을 만들어 배포하거나 타사 개발자가 만든 구문을 사용할 수도 있습니다.

구문은 CPM (구문 프로그래밍 모델) 의 일부입니다. Terraform(cdKTf) 용 CDK, (CDK8) 용 CDK 등의 다른 도구와 함께 사용할 수 있습니다. Kubernetes Projen

주제

- AWS 라이브러리 생성하기

- 구문 정의
- 구문 다루기
- 타사 구문을 사용한 작업
- 자세히 알아보기

AWS 라이브러리 생성하기

AWS 구성 라이브러리에는 에서 개발 및 유지 관리하는 AWS구성 컬렉션이 포함되어 있습니다. 이 모듈은 에서 사용할 수 있는 모든 리소스를 나타내는 구문을 포함하는 다양한 모듈로 구성되어 있습니다. AWS참조 정보는 AWS CDK API 참조를 참조하십시오.

기본 CDK 패키지가 `aws-cdk-lib` 호출되며 이 패키지에는 대부분의 AWS 구성 라이브러리가 포함되어 있습니다. 또한 `Stack` 및 `App` 와 같은 기본 클래스도 포함되어 있습니다.

기본 CDK 패키지의 실제 패키지 이름은 언어마다 다릅니다.

TypeScript

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

JavaScript

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

Python

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

Java

pom.xml에서 추가	Group #####.amazon.awscdk ; artifact aws-cdk-lib
Import	`import software.amazon.awscdk.App;`

C#

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

Go

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

> **Note**
>
> 를 사용하여 CDK 프로젝트를 만든 경우 수동으로 설치할 필요가 없습니다. cdk init `aws-cdk-lib`

AWS 구성 라이브러리에는 `Construct` 기본 클래스가 포함된 `constructs` 패키지도 포함되어 있습니다. Terraform용 CDK 및 Kubernetes용 CDK를 비롯한 다른 구성 기반 도구에서도 사용되므로 자체 패키지에 포함되어 있습니다. AWS CDK

또한 수많은 타사에서 와 호환되는 구문을 게시했습니다. AWS CDK컨스트럭트 허브를 방문하여 AWS CDK 컨스트럭트 파트너 에코시스템을 살펴보세요.

컨스트럭트 레벨

구성 라이브러리의 AWS 구성은 세 가지 수준으로 분류됩니다. 각 수준은 점점 더 높은 수준의 추상화를 제공합니다. 추상화가 높을수록 구성이 쉬워지고 전문 지식이 덜 필요합니다. 추상화가 낮을수록 더 많은 사용자 정의가 가능하므로 더 많은 전문 지식이 필요합니다.

레벨 1 (L1) 구조

CFN 리소스라고도 하는 L1 구문은 가장 낮은 수준의 구조이며 추상화를 제공하지 않습니다. 각 L1 구조는 단일 리소스에 직접 매핑됩니다. AWS CloudFormation L1 구문을 사용하면 특정 리소스를 나타내는 구문을 가져올 수 있습니다. AWS CloudFormation 그런 다음 구성 인스턴스 내에서 리소스의 속성을 정의합니다.

L1 구문은 AWS 리소스 속성 정의에 AWS CloudFormation 익숙하고 완벽하게 제어해야 할 때 사용하기 좋습니다.

AWS 구성 라이브러리에서는 L1 구문의 이름이 로 `Cfn` 시작하고 그 뒤에 해당 구문이 나타내는 AWS CloudFormation 리소스의 식별자가 붙습니다. 예를 들어, `CfnBucket` 구문은 리소스를 나타내는 L1 구조입니다. `AWS::S3::Bucket` AWS CloudFormation

L1 구문은 리소스 사양에서AWS CloudFormation 생성됩니다. 에 리소스가 있는 AWS CloudFormation경우 L1 AWS CDK 구문으로 사용할 수 있습니다. 새 리소스 또는 속성을 AWS 구성 라이브러리에서 사용할 수 있게 되려면 최대 1주일이 걸릴 수 있습니다. 자세한 내용은 AWS CloudFormation 사용 안내서의AWS 리소스 및 속성 유형 참조를 참조하십시오.

레벨 2 (L2) 구문

큐레이티드 구조라고도 하는 L2 구조는 CDK 팀이 신중하게 개발했으며 일반적으로 가장 널리 사용되는 구성 유형입니다. L2 구문은 L1 구문과 마찬가지로 단일 리소스에 직접 매핑됩니다. AWS CloudFormation L1 구문과 비교하여 L2 구문은 직관적인 인텐트 기반 API를 통해 더 높은 수준의 추상화를 제공합니다. L2 구조에는 합리적인 기본 속성 구성, 모범 사례 보안 정책이 포함되며 많은 상용구 코드와 글루 로직이 자동으로 생성됩니다.

또한 L2 구문은 대부분의 리소스에 대한 도우미 메서드를 제공하므로 속성, 권한, 리소스 간 이벤트 기반 상호 작용 등을 더 간단하고 빠르게 정의할 수 있습니다.

이 `s3.Bucket` 클래스는 Amazon Simple Storage 서비스 (Amazon S3) 버킷 리소스의 L2 구문 예제입니다.

AWS 구성 라이브러리에는 안정적인 것으로 지정되어 프로덕션에 바로 사용할 수 있는 L2 구문이 포함되어 있습니다. 개발 중인 L2 구문의 경우 실험용으로 지정되어 별도의 모듈로 제공됩니다.

레벨 3 (L3) 구조

패턴이라고도 하는 L3 구문은 가장 높은 수준의 추상화입니다. 각 L3 구조에는 애플리케이션 내에서 특정 작업이나 서비스를 수행하기 위해 함께 작동하도록 구성된 리소스 모음이 포함될 수 있습니다. L3 구조는 애플리케이션의 특정 사용 사례에 맞는 전체 AWS 아키텍처를 만드는 데 사용됩니다.

전체 시스템 설계 또는 대형 시스템의 상당 부분을 제공하기 위해 L3 구조는 독자적인 기본 속성 구성을 제공합니다. 문제 해결 및 솔루션 제공을 위한 특정 접근 방식을 중심으로 구축되었습니다. L3 구조를 사용하면 최소한의 입력과 코드로 여러 리소스를 빠르게 만들고 구성할 수 있습니다.

이 `ecsPatterns.ApplicationLoadBalancedFargateService` 클래스는 Amazon Elastic Container AWS Fargate Service (Amazon ECS) 클러스터에서 실행되고 애플리케이션 로드 밸런서가 앞에 있는 서비스를 나타내는 L3 구조의 예입니다.

L2 구성과 마찬가지로 프로덕션 환경에서 사용할 준비가 된 L3 구문은 구조 라이브러리에 포함됩니다. AWS 개발 중인 것들은 별도의 모듈로 제공됩니다.

구문 정의

구성

구성은 구문을 통해 상위 수준의 추상화를 정의하는 핵심 패턴입니다. 상위 수준 구문은 여러 개의 하위 수준 구문으로 구성될 수 있습니다. 상향식 관점에서 구문을 사용하여 배포하려는 개별 AWS 리소스를 구성할 수 있습니다. 목적에 맞는 추상화는 무엇이든, 필요한 수준 수만큼 사용할 수 있습니다.

컴포지션을 사용하면 재사용 가능한 구성 요소를 정의하고 다른 코드처럼 공유할 수 있습니다. 예를 들어, 팀은 백업, 글로벌 복제, 자동 조정 및 모니터링을 포함하여 Amazon DynamoDB 테이블에 대한 회사의 모범 사례를 구현하는 구조를 정의할 수 있습니다. 팀은 구성을 내부적으로 다른 팀과 공유하거나 공개적으로 공유할 수 있습니다.

팀은 다른 라이브러리 패키지처럼 구문을 사용할 수 있습니다. 라이브러리가 업데이트되면 개발자는 다른 코드 라이브러리와 마찬가지로 새 버전의 개선 사항 및 버그 수정을 이용할 수 있습니다.

Initialization(초기화)

구성은 `Construct` 기본 클래스를 확장하는 클래스로 구현됩니다. 클래스를 인스턴스화하여 구문을 정의합니다. 모든 구성은 초기화될 때 3개의 파라미터를 사용합니다.

- 범위 — 구문의 부모 또는 소유자입니다. 스택일 수도 있고 다른 구조일 수도 있습니다. 범위는 구성 트리에서의 구문 위치를 결정합니다. 일반적으로 현재 객체를 나타내는 `this` (`self`in Python) 을 범위로 전달해야 합니다.
- id — 범위 내에서 고유해야 하는 식별자입니다. 식별자는 구문 내에 정의된 모든 항목의 네임스페이스 역할을 합니다. 리소스 이름 및 AWS CloudFormation 논리적 ID와 같은 고유 식별자를 생성하는 데 사용됩니다.

  식별자는 범위 내에서만 고유해야 합니다. 이를 통해 구문에 포함될 수 있는 구문 및 식별자에 대한 걱정 없이 구문을 인스턴스화하고 재사용할 수 있으며 구문을 더 높은 수준의 추상화로 구성할 수 있습니다. 또한 범위를 사용하면 구문 그룹을 한 번에 모두 참조할 수 있습니다. 예를 들어 태그를 지정하거나 구문을 배포할 위치를 지정하는 경우를 들 수 있습니다.
- props — 언어에 따라 구문의 초기 구성을 정의하는 속성 또는 키워드 인수 세트입니다. 상위 수준 구문은 더 많은 기본값을 제공하며, 모든 prop 요소가 선택사항인 경우 props 매개변수를 완전히 생략할 수 있습니다.

구성

대부분의 구문은 구문의 구성을 정의하는 이름/값 컬렉션인 세 번째 인수 (또는 Python에서는 키워드 인수) `props` 로 받아들입니다. 다음 예제는 AWS Key Management Service (AWS KMS) 암호화와 정적 웹 사이트 호스팅이 활성화된 버킷을 정의합니다. 암호화 키를 명시적으로 지정하지 않기 때문에 `Bucket` 구조에서는 새 키를 정의하고 이를 `kms.Key` 버킷과 연결합니다.

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"),
})
```

구문과의 상호 작용

구문은 기본 Construct 클래스를 확장하는 클래스입니다. 구문을 인스턴스화하면 구문과 상호 작용하고 이를 시스템의 다른 부분에 대한 참조로 전달할 수 있는 일련의 메서드와 속성이 표시됩니다.

AWS CDK 프레임워크는 구문의 API에 어떠한 제한도 두지 않습니다. 작성자는 원하는 API를 정의할 수 있습니다. 하지만 AWS AWS 구성 라이브러리에 포함된 구문 (예:) 은 지침과 일반적인 패턴을 따릅니다. `s3.Bucket` 이를 통해 모든 AWS 리소스에서 일관된 경험을 제공할 수 있습니다.

대부분의 AWS 구성에는 보안 주체에 해당 구문에 대한 권한 AWS Identity and Access Management (IAM) 을 부여하는 데 사용할 수 있는 권한 부여 방법 집합이 있습니다. 다음 예제는 Amazon S3 `raw-data` 버킷에서 읽을 수 있는 `data-science` 권한을 IAM 그룹에 부여합니다.

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)
```

또 다른 일반적인 패턴은 AWS 구문이 다른 곳에서 제공된 데이터를 기반으로 리소스의 속성 중 하나를 설정하는 것입니다. 속성에는 Amazon 리소스 이름 (ARN), 이름 또는 URL이 포함될 수 있습니다.

다음 코드는 AWS Lambda 함수를 정의하고 환경 변수에 있는 대기열의 URL을 통해 Amazon Simple Queue Service (Amazon SQS) 대기열과 연결합니다.

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()),
 },
})
```

AWS 구성 라이브러리의 가장 일반적인 API 패턴에 대한 자세한 내용은 을 참조하십시오. the section called “리소스”

앱 및 스택 구조

AWS 구성 라이브러리의 `App` 및 `Stack` 클래스는 고유한 구조입니다. 다른 구문에 비해 자체적으로 AWS 리소스를 구성하지는 않습니다. 대신 다른 구문에 컨텍스트를 제공하는 데 사용됩니다. AWS 리소스를 나타내는 모든 구문은 구문 범위 내에서 직접 또는 간접적으로 정의되어야 합니다. `Stack` `Stack`구문은 구문 범위 내에서 정의됩니다. `App`

CDK 앱에 대한 자세한 내용은 을 참조하십시오. AWS CDK 앱 CDK 스택에 대한 자세한 내용은 을 참조하십시오. 스택

다음 예시에서는 단일 스택으로 앱을 정의합니다. 스택 내에서 L2 구조는 Amazon S3 버킷 리소스를 구성하는 데 사용됩니다.

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

`HelloCdkStack.java`파일에 정의된 스택:

```
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();
    }
}
```

`HelloCdkApp.java`파일에 정의된 앱:

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

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

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

        app.synth();
    }
}
```

C#

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

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

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

Go

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

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

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

 return stack
}
```

구문 다루기

L1 구문을 사용한 작업

L1 구조는 개별 리소스에 직접 매핑됩니다. AWS CloudFormation 리소스의 필수 구성을 제공해야 합니다.

이 예제에서는 `CfnBucket` L1 구문을 사용하여 `bucket` 객체를 만듭니다.

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"),
 })
```

단순 불리언, 문자열, 숫자 또는 컨테이너가 아닌 구성 속성은 지원되는 언어에서 다르게 처리됩니다.

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

Python에서 이러한 속성은 L1 구문의 내부 클래스로 정의된 유형으로 표시됩니다. 예를 들어 a의 선택적 속성에는 `cors_configuration` 유형의 래퍼가 `CfnBucket` 필요합니다. `CfnBucket.CorsConfigurationProperty` 여기서는 `CfnBucket` 인스턴스에 `cors_configuration` 대해 정의합니다.

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

Java

Java에서 이러한 속성은 L1 구문의 내부 클래스로 정의된 유형으로 표시됩니다. 예를 들어 a의 선택적 속성에는 `corsConfiguration` 유형의 래퍼가 `CfnBucket` 필요합니다. `CfnBucket.CorsConfigurationProperty` 여기서는 `CfnBucket` 인스턴스에 `corsConfiguration` 대해 정의합니다.

```
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#

C #에서 이러한 속성은 L1 구문의 내부 클래스로 정의된 유형으로 표현됩니다. 예를 들어 a의 선택적 `CorsConfiguration` 속성에는 형식의 래퍼가 `CfnBucket` 필요합니

다. `CfnBucket.CorsConfigurationProperty` 여기서는 `CfnBucket` 인스턴스에 `CorsConfiguration` 대해 정의합니다.

```
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

Go에서는 L1 구문의 이름, 밑줄, 속성 이름을 사용하여 이러한 유형의 이름을 지정합니다. 예를 들어 a의 선택적 `CorsConfiguration` 속성에는 유형의 래퍼가 `CfnBucket` 필요합니다. `CfnBucket_CorsConfigurationProperty` 여기서는 `CfnBucket` 인스턴스에 `CorsConfiguration` 대해 정의합니다.

```
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**
>
> L1 구문에는 L2 속성 유형을 사용할 수 없으며 그 반대의 경우도 마찬가지입니다. L1 구문을 사용할 때는 항상 사용 중인 L1 구문에 정의된 형식을 사용하십시오. 다른 L1 구문의 형식은 사용하지 마세요. 일부는 이름이 같을 수 있지만 형식은 같지 않습니다.
> 현재 일부 언어별 API 참조에서는 L1 속성 유형 경로에 오류가 있거나 이러한 클래스를 전혀 문서화하지 않고 있습니다. 이 문제를 곧 고칠 수 있기를 바랍니다. 하지만 이러한 타입은 항상 함께 사용되는 L1 구문의 내부 클래스라는 점을 기억하세요.

L2 구문 다루기

다음 예제에서는 `Bucket`L2 구조에서 객체를 생성하여 Amazon S3 버킷을 정의합니다.

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`는 AWS CloudFormation 생성한 버킷의 이름이 아닙니다. CDK 앱 컨텍스트 내에서 새 구문에 부여되는 논리적 식별자입니다. PhysicalName 값은 리소스 이름을 지정하는 데 사용됩니다. AWS CloudFormation

타사 구문을 사용한 작업

Construct Hub는 타사 및 오픈 소스 CDK 커뮤니티에서 제공하는 AWS추가 구성을 찾는 데 도움이 되는 리소스입니다.

직접 구문 작성하기

기존 구문을 사용하는 것 외에도 직접 구문을 작성하여 누구나 앱에서 사용할 수 있도록 할 수 있습니다. 모든 구문은 에서 동일합니다. AWS CDK AWS 구성 라이브러리의 구문은, 또는 를 통해 NPM 게시된 타사 라이브러리의 구문과 동일하게 취급됩니다. Maven PyPI 회사의 내부 패키지 저장소에 게시된 구문도 같은 방식으로 취급됩니다.

새 구문을 선언하려면 `constructs` 패키지에서 Construct 기본 클래스를 확장하는 클래스를 만든 다음 이니셜라이저 인수의 패턴을 따르십시오.

다음 예제는 Amazon S3 버킷을 나타내는 구문을 선언하는 방법을 보여줍니다. S3 버킷은 누군가 파일을 업로드할 때마다 Amazon Simple Notification 서비스 (Amazon SNS) 알림을 보냅니다.

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**
>
> 우리의 `NotifyingBucket` 구조는 그것으로부터 상속되는 것이 아니라 그것으로부터 `Bucket` 상속됩니다. `Construct` Amazon S3 버킷과 Amazon SNS 주제를 함께 묶을 때는 상속이 아닌 구성을 사용하고 있습니다. 일반적으로 AWS CDK 구문을 개발할 때는 상속보다 구성이 선호됩니다.

`NotifyingBucket`생성자의 일반적인 구문 서명은, `scope`id, 및 입니다. `props` 모든 props가 선택 사항이므로 마지막 인수 는 선택 사항입니다 (기본값 가져오기{}). `props` (기본 `Construct` 클래스는 `props` 인수를 취하지 않습니다.) 다음과 같은 경우를 제외하고 `props` 앱에서 이 구문의 인스턴스를 정의할 수 있습니다.

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)
```

또는 `props` (Java에서는 추가 매개변수를) 사용하여 필터링할 경로 접두사를 지정할 수 있습니다. 예를 들면 다음과 같습니다.

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/"),
})
```

일반적으로 구문에 일부 속성이나 메서드를 노출하고 싶을 수도 있습니다. 구문 뒤에 주제를 숨기는 것은 그다지 유용하지 않습니다. 구문 사용자는 주제를 구독할 수 없기 때문입니다. 다음 예와 같이 `topic` 속성을 추가하면 소비자가 내부 주제에 액세스할 수 있습니다.

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

Go에서 이 작업을 수행하려면 약간의 추가 배관이 필요합니다. 원래 NewNotifyingBucket 함수는 a를 반환했습니다. awss3.Bucket NotifyingBucket구조체를 만들어 Bucket topic 멤버를 포함하도록 확장해야 합니다. 그러면 함수가 이 유형을 반환합니다.

```
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
}
```

이제 소비자는 해당 주제를 구독할 수 있습니다. 예를 들면 다음과 같습니다.

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))
```

자세히 알아보기

다음 동영상은 CDK 구조에 대한 포괄적인 개요를 제공하고 CDK 앱에서 CDK 구조를 사용하는 방법을 설명합니다.

CDK 구조 설명

환경

환경이 AWS 계정 대상이며 스택이 AWS 리전 배포되는 대상입니다. CDK 앱의 모든 스택은 명시적 또는 암시적으로 환경 () 과 연결됩니다. env

주제

- 환경 구성
- 부트스트래핑 환경

환경 구성

프로덕션 스택의 경우 속성을 사용하여 앱의 각 스택에 대한 환경을 명시적으로 지정하는 것이 좋습니다. env 다음 예시에서는 서로 다른 두 스택에 대해 서로 다른 환경을 지정합니다.

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 });
```

이전 예와 같이 대상 계정과 지역을 하드 코딩하면 스택은 항상 해당 특정 계정 및 지역에 배포됩니다. 스택을 다른 대상에 배포할 수 있지만 합성 시 대상을 결정하려면 AWS CDK CLI에서 제공하는 두 가지 환경 변수인 및 를 스택에 사용할 수 있습니다. CDK_DEFAULT_ACCOUNT CDK_DEFAULT_REGION 이러한 변수는 --profile 옵션을 사용하여 지정한 AWS 프로필 또는 기본 프로필을 지정하지 않은 경우 기본 AWS 프로필을 기반으로 설정됩니다.

다음 코드 조각은 스택의 AWS CDK CLI에서 전달된 계정 및 지역에 액세스하는 방법을 보여줍니다.

TypeScript

Node의 `process` 객체를 통해 환경 변수에 액세스합니다.

> **Note**
>
> `process`에서 사용할 `DefinitelyTyped` 모듈이 필요합니다 TypeScript. `cdk init`이 모듈을 자동으로 설치합니다. 하지만 추가되기 전에 만든 프로젝트로 작업하는 중이거나 를 사용하여 `cdk init` 프로젝트를 설정하지 않은 경우에는 이 모듈을 수동으로 설치해야 합니다.

```
npm install @types/node
```

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

JavaScript

Node의 `process` 객체를 통해 환경 변수에 액세스할 수 있습니다.

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

Python

`os`모듈의 `environ` 사전을 사용하여 환경 변수에 액세스합니다.

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

Java

환경 변수 값을 가져오는 `System.getenv()` 데 사용합니다.

```
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#

환경 변수 값을 가져오는 `System.Environment.GetEnvironmentVariable()` 데 사용합니다.

```
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() });
```

지역 코드를 AWS 리전 사용하여 지정하십시오. 목록은 리전 엔드포인트를 참조하십시오.

env속성을 전혀 지정하지 않는 것과 및 를 사용하여 `CDK_DEFAULT_ACCOUNT` 속성을 지정하는 것을 AWS CDK 구분합니다. `CDK_DEFAULT_REGION` 전자는 스택이 환경에 구애받지 않는 템플릿을

합성해야 함을 의미합니다. 이러한 스택에 정의된 구문은 해당 환경에 대한 정보를 사용할 수 없습니다. 예를 들어 계정을 쿼리해야 하는 vpc.FromLookup (Pythonfrom_lookup:) 과 같은 코드를 `if (stack.region === 'us-east-1')` 작성하거나 프레임워크 기능을 사용할 수 없습니다. AWS 이러한 기능은 명시적 환경을 지정하기 전까지는 전혀 작동하지 않습니다. 이러한 기능을 사용하려면 먼저 지정해야 합니다. env

CDK_DEFAULT_ACCOUNT및 CDK_DEFAULT_REGION 를 사용하여 환경을 전달하면 통합 시 AWS CDK CLI에서 결정한 계정 및 지역에 스택이 배포됩니다. 이렇게 하면 환경에 따른 코드가 작동할 수 있지만 합성되는 템플릿은 합성 대상 컴퓨터, 사용자 또는 세션에 따라 달라질 수 있습니다. 이러한 동작은 개발 중에 허용되거나 바람직한 경우가 많지만 프로덕션 용도로는 패턴을 방해할 수 있습니다.

유효한 표현식을 사용하여 원하는 env 대로 설정할 수 있습니다. 예를 들어, 합성 시 계정과 지역을 재정의할 수 있도록 두 개의 추가 환경 변수를 지원하도록 스택을 작성할 수 있습니다. 이 CDK_DEPLOY_ACCOUNT 둘을 CDK_DEPLOY_REGION 여기서는 호출하겠지만 에서 설정하지 않았으므로 원하는 대로 이름을 지정할 수 있습니다. AWS CDK다음 스택의 환경에서는 대체 환경 변수가 설정된 경우 이 변수가 사용됩니다. 설정되지 않은 경우 에서 제공하는 기본 환경으로 대체됩니다 AWS CDK.

TypeScript

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

JavaScript

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

Python

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

Java

```
public class MyApp {

    // Helper method to build an environment
    static Environment makeEnv(String account, String region) {
        account = (account == null) ? System.getenv("CDK_DEPLOY_ACCOUNT") : account;
        region = (region == null) ? System.getenv("CDK_DEPLOY_REGION") : region;
        account = (account == null) ? System.getenv("CDK_DEFAULT_ACCOUNT") :
 account;
        region = (region == null) ? System.getenv("CDK_DEFAULT_REGION") : region;

        return Environment.builder()
                .account(account)
                .region(region)
                .build();
    }

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

        Environment envEU = makeEnv(null, null);
        Environment envUSA = makeEnv(null, null);

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

        app.synth();
    }
}
```

C#

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

```
            System.Environment.GetEnvironmentVariable("CDK_DEPLOY_REGION") ??
            System.Environment.GetEnvironmentVariable("CDK_DEFAULT_REGION")
    };
}

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

스택 환경을 이렇게 선언하면 다음과 같은 간단한 스크립트나 배치 파일을 작성하여 명령줄 인수에서 변수를 설정한 다음 호출할 수 `cdk deploy` 있습니다. 처음 두 개 이상의 인수는 모두 전달되어 명령줄 옵션 또는 스택을 지정하는 데 사용할 수 있습니다. `cdk deploy`

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
```

스크립트를 다른 이름으로 저장한 `cdk-deploy-to.sh` 다음 `chmod +x cdk-deploy-to.sh` 실행하여 실행 가능하게 만듭니다.

Windows

```
@findstr /B /V @ %~dpnx0 > %~dpn0.ps1 && powershell -ExecutionPolicy Bypass
 %~dpn0.ps1 %*
@exit /B %ERRORLEVEL%
if ($args.length -ge 2) {
    $env:CDK_DEPLOY_ACCOUNT, $args = $args
    $env:CDK_DEPLOY_REGION,  $args = $args
    npx cdk deploy $args
    exit $lastExitCode
} else {
    [console]::error.writeline("Provide account and region as first two args.")
    [console]::error.writeline("Additional args are passed through to cdk deploy.")
```

```
    exit 1
}
```

스크립트의 Windows 버전은 macOS/Linux 버전과 동일한 기능을 제공하는 PowerShell 데 사용됩니다. 또한 명령줄에서 쉽게 호출할 수 있도록 배치 파일로 실행할 수 있도록 하는 지침도 포함되어 있습니다. 로 `cdk-deploy-to.bat` 저장해야 합니다. 배치 파일이 `cdk-deploy-to.ps1` 호출될 때 파일이 생성됩니다.

그런 다음 “deploy-to” 스크립트를 호출하는 추가 스크립트를 작성하여 특정 환경 (스크립트당 여러 환경 포함) 에 배포할 수 있습니다.

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 %*
```

여러 환경에 배포할 때는 배포가 실패한 후에도 다른 환경에 계속 배포할지 여부를 고려하세요. 다음 예제에서는 첫 번째 프로덕션 환경에서 실패할 경우 두 번째 프로덕션 환경에 배포하지 않도록 합니다.

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 %*
```

개발자는 여전히 일반 `cdk deploy` 명령을 사용하여 자체 AWS 환경에 배포하여 개발할 수 있습니다.

스택을 인스턴스화할 때 환경을 지정하지 않으면 환경에 구애받지 않는 스택이라고 합니다. AWS CloudFormation 이러한 스택에서 합성된 템플릿은`stack.account`,, (`stack.availabilityZones`Python:) `stack.region` 와 같은 환경 관련 속성에 배포 시간 해상도를 사용하려고 합니다. `availability_zones`

를 cdk deploy 사용하여 환경에 구애받지 않는 스택을 배포하는 경우 AWS CDK CLI 는 지정된 프로필을 사용하여 배포 위치를 결정합니다. AWS CLI 프로필을 지정하지 않으면 기본 프로필이 사용됩니다. AWS 계정에서 작업을 수행할 때 사용할 AWS 자격 증명을 결정하는 것과 유사한 프로토콜을 AWS CDK CLI 따릅니다. AWS CLI 세부 정보는 the section called “AWS CDK 툴킷”를 참조하세요.

환경에 구애받지 않는 스택에서는 가용 영역을 사용하는 모든 구조에는 두 개의 가용 영역이 표시되므로 스택을 모든 지역에 배포할 수 있습니다.

부트스트래핑 환경

CDK 스택을 배포할 각 환경을 부트스트랩해야 합니다. 부트스트래핑은 배포를 위한 환경을 준비합니다. 자세한 내용은 부트스트래핑 단원을 참조하십시오.

부트스트래핑

부트스트래핑은 배포할 환경을 준비하는 프로세스입니다. 부트스트래핑은 리소스를 배포하는 모든 환경에서 수행해야 하는 일회성 작업입니다.

주제

- 부트스트래핑 환경
- 부트스트랩 방법
- 부트스트래핑 사용자 지정
- 부트스트래핑 템플릿 차이점
- 스택 신시사이저
- 커스터마이징 합성
- 부트스트래핑 템플릿 계약
- Security Hub 조사 결과

부트스트래핑 환경

> **Important**
>
> 부트스트랩된 리소스에 저장된 데이터에 대해 AWS 요금이 부과될 수 있습니다.

부트스트래핑은 파일을 저장하기 위한 Amazon Simple S3 (Amazon Simple Storage Service) 버킷 및 배포를 수행하는 데 필요한 권한을 부여하는 AWS Identity and Access Management (IAM) 역할과 같은 환경 내 리소스를 프로비저닝합니다. 이러한 리소스는 부트스트랩 스택이라는 AWS CloudFormation 스택에 프로비저닝됩니다. 일반적으로 이름이 지정됩니다. `CDKToolkit` 다른 AWS CloudFormation 스택과 마찬가지로 배포가 완료되면 환경의 AWS CloudFormation 콘솔에 나타납니다.

> **Note**
>
> CDK v2는 최신 부트스트랩 템플릿을 사용합니다. CDK v1의 레거시 템플릿은 v2에서 지원되지 않습니다.

환경은 독립적입니다. 여러 환경에 배포하려는 경우 각 환경을 개별적으로 부트스트랩해야 합니다.

부트스트랩되지 않은 환경에 CDK 앱을 배포하려고 하면 환경을 부트스트랩하라는 오류 메시지가 나타납니다.

CDK Pipeline을 사용한 부트스트래핑

CDK Pipelines를 사용하여 다른 계정의 환경에 배포하는 경우 다음과 같은 메시지가 표시됩니다.

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

이 오류 메시지는 적절한 IAM 역할이 다른 환경에 존재하지 않음을 의미합니다. 가장 가능성이 높은 원인은 환경이 부트스트랩되지 않았기 때문입니다. 환경을 부트스트랩하고 다시 시도하십시오.

> **Note**
>
> 환경이 부트스트랩된 경우 해당 환경의 부트스트랩 스택을 삭제하고 다시 만들지 마십시오. 부트스트랩 스택을 삭제하면 CDK 배포를 지원하기 위해 환경에 원래 프로비저닝된 AWS 리소스

가 삭제됩니다. 그러면 파이프라인 작동이 중지됩니다. 대신 CDK CLI `cdk bootstrap` 명령을 다시 실행하여 부트스트랩 스택을 새 버전으로 업데이트해 보세요.

부트스트랩 방법

환경을 부트스트랩하면 AWS CloudFormation 템플릿이 특정 환경에 배포됩니다. 이 템플릿은 배포를 위한 환경을 준비하기 위해 계정의 리소스를 제공합니다.

부트스트래핑 템플릿은 부트스트랩된 리소스의 일부 측면을 사용자 지정하는 매개 변수를 허용합니다. 자세한 정보는 the section called “부트스트래핑 사용자 지정”을 참조하세요.

다음 방법 중 하나로 부트스트랩할 수 있습니다.

- AWS CDK CLI 명령을 사용합니다.cdk bootstrap 이는 가장 간단한 방법이며 부트스트랩할 환경이 몇 개뿐인 경우에 적합합니다.
- 다른 AWS CloudFormation 배포 도구를 AWS CDK CLI 사용하여 에서 제공한 템플릿을 배포합니다. 이렇게 하면 OR를 사용할 AWS CloudFormation StackSets 수 AWS Control Tower 있으며 AWS CloudFormation 콘솔이나 도 사용할 수 AWS CLI있습니다. 배포 전에 템플릿을 약간 수정할 수 있습니다. 이 접근 방식은 더 유연하며 대규모 배포에 적합합니다.

환경을 두 번 이상 부트스트랩해도 오류가 발생하지 않습니다. 부트스트랩하는 환경이 이미 부트스트랩된 경우 필요한 경우 해당 부트스트랩 스택이 업그레이드됩니다. 그렇지 않으면 아무 일도 일어나지 않을 것입니다.

를 이용한 부트스트래핑 AWS CDKCLI

`cdk bootstrap`명령을 사용하여 하나 이상의 환경을 부트스트랩합니다. AWS

다음 예제는 두 환경을 부트스트랩합니다.

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

다음 예제는 환경을 부트스트래핑하는 여러 방법을 보여줍니다. 두 번째 예제에서 볼 수 있듯이 환경을 지정할 때 `aws://` 접두사는 선택 사항입니다.

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

를 cdk bootstrap 실행하면 CDK는 CLI 항상 현재 디렉터리에서 CDK 앱을 합성합니다. 환경을 하나 이상 지정하지 않으면 CLI CDK는 앱에서 참조되는 모든 환경을 부트스트랩합니다.

환경에 구애받지 않는 스택의 경우 CLI CDK는 기본 소스에서 환경을 결정하려고 시도합니다. --profile옵션을 사용하여 지정한 환경, 환경 변수 또는 기본 소스일 수 있습니다. AWS CLI 발견되면 해당 환경이 부트스트랩됩니다.

예를 들어 다음 명령은 `prod` AWS 프로필을 사용하여 현재 AWS CDK 앱을 합성한 다음 해당 환경을 부트스트랩합니다.

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

템플릿에서 부트스트래핑 AWS CloudFormation

부트스트랩 템플릿을 구하여 배포하여 환경을 부트스트랩할 수 있습니다. AWS CloudFormation

파일에서 `bootstrap-template.yaml` 이 템플릿의 사본을 가져오려면 다음 명령을 실행합니다.

macOS/Linux

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

Windows

Windows에서는 템플릿의 인코딩을 보존하는 데 PowerShell 사용해야 합니다.

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

템플릿은 AWS CDK GitHub 저장소에서도 사용할 수 있습니다.

CDK CLI 또는 선호하는 템플릿 배포 메커니즘을 AWS CloudFormation 사용하여 이 템플릿을 배포합니다. CDK CLI를 사용하여 배포하려면 를 실행합니다. cdk bootstrap --template ***TEMPLATE_FILENAME*** 아래 명령을 AWS CLI 실행하여 배포하거나 AWS CloudFormation Stack Sets 를 사용하여 한 번에 하나 이상의 계정에 배포할 수도 있습니다.

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
```

부트스트래핑 사용자 지정

사용자 환경에서 리소스의 부트스트래핑을 사용자 지정하는 두 가지 방법이 있습니다.

- 명령줄 파라미터를 명령과 함께 사용하십시오. `cdk bootstrap` 이렇게 하면 템플릿의 몇 가지 측면을 수정할 수 있습니다.
- 기본 부트스트랩 템플릿을 수정하고 직접 배포하십시오. 이렇게 하면 부트스트랩 리소스를 보다 완벽하게 제어할 수 있습니다.

다음 명령줄 옵션을 CLI cdk bootstrap CDK와 함께 사용하면 일반적으로 사용되는 부트스트래핑 템플릿을 조정할 수 있습니다.

- - Amazon S3 버킷의 이름을 -bootstrap-bucket-name 재정의합니다. CDK 앱을 변경해야 할 수 있습니다 (참조). the section called “스택 신시사이저”
- --bootstrap-kms-key-idS3 버킷을 암호화하는 데 사용되는 AWS KMS 키를 재정의합니다.
- --cloudformation-execution-policies스택 배포 AWS CloudFormation 중에 할당되는 배포 역할에 연결되어야 하는 관리형 정책의 ARN을 지정합니다. 기본적으로 스택은 정책을 사용하여 전체 관리자 권한으로 배포됩니다. `AdministratorAccess`

  정책 ARN은 개별 ARN을 쉼표로 구분하여 단일 문자열 인수로 전달해야 합니다. 예:

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

> **Important**
>
> 배포 실패를 방지하려면 지정한 정책이 부트스트랩 중인 환경에서 수행할 모든 배포에 충분한지 확인하십시오.

- --qualifier부트스트랩 스택의 모든 리소스 이름에 추가되는 문자열입니다. 한정자를 사용하면 동일한 환경에서 여러 부트스트랩 스택을 프로비전할 때 리소스 이름 충돌을 피할 수 있습니다. 기본값은 입니다 `hnb659fds` (이 값은 중요하지 않음).

  또한 한정자를 변경하려면 CDK 앱이 변경된 값을 스택 신시사이저에 전달해야 합니다. 자세한 정보는 the section called “스택 신시사이저”을 참조하세요.
- --tags부트스트랩 스택에 하나 이상의 AWS CloudFormation 태그를 추가합니다.
- --trust부트스트랩 중인 환경에 배포할 수 있는 AWS 계정을 나열합니다.

  다른 환경의 CDK 파이프라인이 배포할 환경을 부트스트래핑할 때 이 플래그를 사용하세요. 부트스트래핑을 수행하는 계정은 항상 신뢰할 수 있습니다.
- --trust-for-lookup부트스트랩 중인 환경에서 컨텍스트 정보를 조회할 수 있는 AWS 계정을 나열합니다.

  이 플래그를 사용하면 해당 스택을 직접 배포할 권한을 실제로 부여하지 않고도 환경에 배포할 스택을 합성할 수 있는 권한을 계정에 부여할 수 있습니다.
- --termination-protection부트스트랩 스택이 삭제되는 것을 방지합니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 스택 삭제 방지를 참조하십시오.

> **Important**
>
> 최신 부트스트랩 템플릿은 에서 `--cloudformation-execution-policies` 암시하는 권한을 목록에 있는 모든 AWS 계정에 효과적으로 부여합니다. `--trust` 기본적으로 이렇게 하면 부트스트랩된 계정의 모든 리소스에 대한 읽기 및 쓰기 권한이 확장됩니다. 마음에 드는 정책 및 신뢰할 수 있는 계정으로 부트스트래핑 스택을 구성해야 합니다.

템플릿 사용자 지정

CDK가 제공할 CLI 수 있는 것보다 더 많은 사용자 지정이 필요한 경우 필요에 맞게 부트스트랩 템플릿을 수정할 수 있습니다. 먼저 옵션을 사용하여 템플릿을 가져옵니다. --show-template 다음은 그 예제입니다.

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

모든 수정 사항은 부트스트래핑 템플릿 계약을 준수해야 합니다. 나중에 기본 템플릿을 cdk bootstrap 사용하여 실행 중인 사용자가 실수로 사용자 정의를 덮어쓰는 일이 없도록 템플릿 매개변수의 기본값을 변경하십시오. `BootstrapVariant` CDK CLI에서는 현재 배포된 템플릿과 동일하거나 `BootstrapVariant` 동일하거나 상위 버전의 템플릿으로만 부트스트랩 스택을 덮어쓸 수 있습니다.

그런 다음 에 설명된 대로 또는 를 사용하여 수정된 템플릿을 배포할 수 있습니다. the section called “템플릿에서 부트스트래핑 AWS CloudFormation” cdk bootstrap --template

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

부트스트래핑 템플릿 차이점

앞서 언급했듯이 AWS CDK v1은 레거시와 모던의 두 가지 부트스트래핑 템플릿을 지원했습니다. CDK v2는 최신 템플릿만 지원합니다. 참고로 이 두 템플릿 간의 주요 차이점은 다음과 같습니다.

기능	레거시 (v1만 해당)	모던 (v1 및 v2)
크로스 어카운트 배포	허용되지 않음	허용됨
AWS CloudFormation 권한	현재 사용자의 권한 (AWS 프로필, 환경 변수 등에 따라 결정됨) 을 사용하여 배포합니다.	부트스트랩 스택이 프로비저닝될 때 지정된 권한을 사용하여 배포합니다 (예: 사용). `--trust`
버전 관리	부트스트랩 스택은 한 가지 버전만 사용할 수 있습니다.	부트스트랩 스택은 버전이 지정되어 있으므로 향후 버전에서 새 리소스를 추가할 수 있으며 AWS CDK 앱에는 최소 버전이 필요할 수 있습니다.

기능	레거시 (v1만 해당)	모던 (v1 및 v2)
리소스 *	Amazon S3 버킷	Amazon S3 버킷
		AWS KMS key
		IAM 역할
		아마존 ECR 리포지토리
		버전 관리를 위한 SSM 파라미터
리소스 이름 지정	자동 생성됨	DETERMINISTIC
버킷 암호화	기본 키	고객 관리형 키

* 필요에 따라 부트스트랩 템플릿에 리소스를 추가할 예정입니다.

기존 템플릿을 사용하여 부트스트랩한 환경을 다시 부트스트랩하여 CDK v2용 최신 템플릿을 사용하도록 업그레이드해야 합니다. 기존 버킷을 삭제하기 전에 환경의 모든 AWS CDK 애플리케이션을 한 번 이상 재배포하십시오.

스택 신시사이저

배포할 수 있는 스택을 성공적으로 합성하려면 AWS CDK 앱에서 사용할 수 있는 부트스트래핑 리소스에 대해 알아야 합니다. 스택 신시사이저는 스택의 템플릿이 합성되는 방식을 제어하는 AWS CDK 클래스입니다. 여기에는 부트스트랩 리소스를 사용하는 방식 (예: 부트스트랩 버킷에 저장된 자산을 참조하는 방식) 이 포함됩니다.

에 AWS CDK내장된 스택 신시사이저가 호출됩니다. `DefaultStackSynthesizer` 여기에는 계정 간 배포 및 CDK Pipelines 배포를 위한 기능이 포함됩니다.

속성을 사용하여 스택을 인스턴스화할 때 스택 신시사이저를 스택에 전달할 수 있습니다. `synthesizer`

TypeScript

```
new MyStack(this, 'MyStack', {
  // stack properties
```

```
  synthesizer: new DefaultStackSynthesizer({
    // synthesizer properties
  }),
});
```

JavaScript

```
new MyStack(this, 'MyStack', {
  // stack properties
  synthesizer: new DefaultStackSynthesizer({
    // synthesizer properties
  }),
});
```

Python

```
MyStack(self, "MyStack",
    # stack properties
    synthesizer=DefaultStackSynthesizer(
        # synthesizer properties
))
```

Java

```
new MyStack(app, "MyStack", StackProps.builder()
    // stack properties
  .synthesizer(DefaultStackSynthesizer.Builder.create()
    // synthesizer properties
    .build())
  .build();
```

C#

```
new MyStack(app, "MyStack", new StackProps
// stack properties
{
    Synthesizer = new DefaultStackSynthesizer(new DefaultStackSynthesizerProps
    {
        // synthesizer properties
    })
});
```

synthesizer속성을 제공하지 않으면 가 사용됩니다. DefaultStackSynthesizer

커스터마이징 합성

부트스트랩 템플릿의 변경 사항에 따라 합성을 사용자 정의해야 할 수도 있습니다. 다음과 같이 설명된 속성을 사용하여 사용자 정의할 DefaultStackSynthesizer 수 있습니다.

이러한 속성 중 필요한 사용자 지정을 제공하지 않는 경우 신시사이저를 구현 IStackSynthesizer (아마도 에서 파생된) 클래스로 작성할 수 있습니다. DefaultStackSynthesizer

한정자 변경

한정자는 별도의 부트스트랩 스택에 있는 리소스를 구분하기 위해 부트스트랩 리소스 이름에 추가됩니다. 동일한 환경 (AWS 계정 및 지역) 에 서로 다른 두 버전의 부트스트랩 스택을 배포하려면 스택에 서로 다른 한정자가 있어야 합니다.

이 기능은 CDK 자체의 자동화된 테스트 간에 이름을 분리하기 위한 것입니다. AWS CloudFormation 실행 역할에 부여된 IAM 권한의 범위를 매우 정확하게 좁힐 수 없는 한 단일 계정에 두 개의 서로 다른 부트스트랩 스택을 보유해도 권한 격리의 이점은 없습니다. 따라서 일반적으로 이 값을 변경할 필요가 없습니다.

한정자를 변경하려면 다음 속성으로 신디사이저를 인스턴스화하여 DefaultStackSynthesizer 둘 중 하나를 구성하십시오.

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"
    })
});
```

또는 한정자를 컨텍스트 키로 구성할 수도 있습니다. `cdk.json`

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

리소스 이름 변경

다른 모든 `DefaultStackSynthesizer` 속성은 부트스트래핑 템플릿의 리소스 이름과 관련이 있습니다. 부트스트랩 템플릿을 수정하고 리소스 이름 또는 이름 지정 체계를 변경한 경우에만 이러한 속성을 제공하면 됩니다.

모든 속성에는 특수 자리 표시자${Qualifier},${AWS::Partition}, ${AWS::AccountId} 및 가 허용됩니다. ${AWS::Region} 이러한 플레이스홀더는 각각 qualifier 매개변수 값과 스택 환경의 AWS 파티션, 계정 ID, 지역 값으로 대체됩니다.

다음 예제는 신디사이저를 인스턴스화하는 것처럼 가장 일반적으로 사용되는 속성을 기본값과 DefaultStackSynthesizer 함께 보여줍니다. 전체 목록은 DefaultStackSynthesizerProps 단원을 참조하십시오.

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,
})
```

부트스트래핑 템플릿 계약

부트스트래핑 스택의 요구 사항은 사용 중인 스택 신시사이저에 따라 다릅니다. 스택 신시사이저를 직접 작성하면 신시사이저에 필요한 부트스트랩 리소스와 신디사이저가 리소스를 찾는 방법을 완벽하게 제어할 수 있습니다.

이 섹션에서는 부트스트래핑 템플릿에 `DefaultStackSynthesizer` 대한 기대치를 설명합니다.

버전 관리

템플릿에는 잘 알려진 이름의 SSM 매개 변수를 만들기 위한 리소스와 템플릿 버전을 반영하는 출력이 포함되어야 합니다.

```
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]
```

역할

`DefaultStackSynthesizer`에는 다섯 가지 용도를 위한 5개의 IAM 역할이 필요합니다. 기본 역할을 사용하지 않는 경우 사용하려는 역할의 ARN을 신디사이저에 알려야 합니다.

역할은 다음과 같습니다.

- AWS CDK 툴킷이 배포 역할을 맡고 환경에 AWS CodePipeline 배포하는 역할을 맡습니다. 환경에 배포할 수 있는 사용자를 `AssumeRolePolicy` 제어합니다. 템플릿에서 이 역할에 필요한 권한을 확인할 수 있습니다.
- AWS CDK 툴킷은 환경에서 컨텍스트 조회를 수행하는 조회 역할을 맡습니다. 환경에 배포할 수 있는 사용자를 `AssumeRolePolicy` 제어합니다. 이 역할에 필요한 권한은 템플릿에서 확인할 수 있습니다.
- AWS CDK 툴킷과 AWS CodeBuild 프로젝트는 환경에 자산을 게시하는 파일 게시 역할과 이미지 게시 역할을 맡습니다. 이들은 각각 S3 버킷과 ECR 리포지토리에 쓰는 데 사용됩니다. 이러한 역할에는 이러한 리소스에 대한 쓰기 권한이 필요합니다.
- 실제 배포를 수행하기 AWS CloudFormation 위해 AWS CloudFormation 실행 역할이 전달됩니다. 해당 권한은 배포가 실행될 권한입니다. 권한은 관리형 정책 ARN을 나열하는 파라미터로 스택에 전달됩니다.

결과

AWS CDK 툴킷을 사용하려면 부트스트랩 스택에 다음과 같은 CloudFormation 출력이 있어야 합니다.

- `BucketName`: 파일 자산 버킷의 이름
- `BucketDomainName`: 도메인 이름 형식의 파일 자산 버킷
- `BootstrapVersion`: 부트스트랩 스택의 현재 버전

템플릿 기록

부트스트랩 템플릿은 버전이 지정되며 시간이 지남에 따라 자체적으로 발전합니다. AWS CDK 자체 부트스트랩 템플릿을 제공하는 경우 표준 기본 템플릿을 사용하여 최신 상태로 유지하십시오. 템플릿이 모든 CDK 기능과 계속 호환되는지 확인하고 싶을 것입니다.

> **Note**
>
> 이전 버전의 부트스트랩 템플릿은 기본적으로 각 부트스트랩 환경에 를 AWS KMS key 생성했습니다. KMS 키에 대한 요금이 부과되지 않도록 하려면 를 사용하여 이러한 환경을 다시 부트스트랩하십시오. `--no-bootstrap-customer-key` 현재 기본값은 KMS 키를 사용하지 않는 것이므로 이러한 요금이 부과되지 않도록 할 수 있습니다.

이 섹션에는 각 버전에서 변경된 사항 목록이 포함되어 있습니다.

템플릿 버전	AWS CDK 버전	변경
1	1.40.0	버킷, 키, 리포지토리, 역할이 포함된 템플릿의 초기 버전
2	1.45.0	자산 게시 역할을 별도의 파일 및 이미지 게시 역할로 분할합니다.
3	1.46.0	자산 `FileAssetKeyArn` 소비자에게 암호 해독 권한을 추가할 수 있도록 내보내기를 추가하십시오.
4	1.61.0	AWS KMS 권한은 이제 Amazon S3를 통해 암시되며 더 이상 필요하지 `FileAsetKeyArn` 않습니다. 스택 이름을 몰라도 부트스트랩 스택 버전을 확인할 수 있도록 `CdkBootstrapVersion` SSM 파라미터를 추가합니다.
5	1.87.0	배포 역할은 SSM 파라미터를 읽을 수 있습니다.
6	1.108.0	배포 역할과 별도로 조회 역할을 추가합니다.

템플릿 버전	AWS CDK 버전	변경
6	1.109.0	배포, 파일 게시, 이미지 게시 역할에 `aws-cdk:bootstrap-role` 태그를 첨부합니다.
7	1.110.0	배포 역할은 더 이상 대상 계정의 버킷을 직접 읽을 수 없습니다. (하지만 이 역할은 사실상 관리자이므로 언제든지 해당 AWS CloudFormation 권한을 사용하여 버킷을 읽을 수 있게 만들 수 있습니다.)
8	1.114.0	조회 역할에는 대상 환경에 대한 전체 읽기 전용 권한이 있으며 태그도 있습니다`aws-cdk:bootstrap-role` .
9	2.1.0	Amazon S3 자산 업로드가 일반적으로 참조되는 암호화 SCP에 의해 거부되지 않도록 수정합니다.
10	2.4.0	Amazon ScanOnPush ECR은 이제 기본적으로 활성화되어 있습니다.
11	2.18.0	Lambda가 Amazon ECR 리포지토리에서 데이터를 가져와서 재부트스트래핑을 계속할 수 있도록 하는 정책을 추가합니다.
12.	2.20.0	실험에 대한 지원을 추가합니다`cdk import`.

템플릿 버전	AWS CDK 버전	변경
13	2.25.0	부트스트랩으로 생성한 Amazon ECR 리포지토리의 컨테이너 이미지를 변경할 수 없게 만듭니다.
14	2.34.0	이미지 스캔을 지원하지 않는 지역의 부트스트래핑을 허용하기 위해 리포지토리 수준에서 Amazon ECR 이미지 스캔을 기본적으로 끕니다.
15	2.60.0	KMS 키에는 태그를 지정할 수 없습니다.
16	2.69.0	KMS.2를 찾는 Security Hub 주소를 지정합니다.
17	2.72.0	ECR을 찾는 Security Hub의 주소를 지정합니다.3.
18	2.80.0	버전 16에서 변경한 내용은 모든 파티션에서 작동하지 않으므로 되돌리는 것은 권장되지 않습니다.
19	2.106.1	템플릿에서 AccessControl 속성이 제거된 버전 18의 변경 사항을 되돌렸습니다. (#27964)
20	2.119.0	IAM AWS CloudFormation 배포 역할에 `ssm:GetParameters` 작업을 추가합니다. 자세한 내용은 #28336 을 참조하십시오.

Security Hub 조사 결과

를 사용하는 AWS Security Hub경우 AWS CDK 부트스트래핑 프로세스에서 생성된 일부 리소스에 대한 결과가 보고될 수 있습니다. Security Hub 검색 결과는 정확성과 안전성을 다시 확인해야 하는 리소스 구성을 찾는 데 도움이 됩니다. 보안 팀과 이러한 특정 리소스 구성을 검토한 결과 AWS 보안 문제를 구성하지 않는다고 확신합니다.

[KMS.2] IAM 보안 주체에는 모든 KMS 키에 대한 암호 해독 작업을 허용하는 IAM 인라인 정책이 없어야 합니다.

배포 역할 (기본 이름`cdk-hnb659fds-deploy-role-ACCOUNT-REGION`) 에는 Amazon S3에 저장된 암호화된 데이터를 읽을 권한이 있습니다. 이 정책은 그 자체로 어떤 데이터에도 권한을 부여하지 않습니다. Amazon S3에서 읽은 데이터만 복호화할 수 있으며, 배포 역할이 해당 버킷 정책을 통해 데이터를 읽을 수 있도록 명시적으로 허용하는 버킷 및 배포 역할이 키 정책을 사용하여 해당 데이터를 사용하여 암호를 해독하도록 명시적으로 허용하는 키에서만 해독할 수 있습니다. 이 명령문은 파이프라인이 계정 간 배포를 수행할 수 있도록 하는 데 사용됩니다. AWS CDK

Security Hub에서 플래그를 지정하는 이유는 무엇인가요? 정책에는 조항이 `Resource: *` 결합되어 있으며 Security Hub는 해당 `Condition` 조항을 신고하고 있습니다. * 계정이 부트스트랩될 때 AWS CDK Pipelines에서 CodePipeline Artifact Bucket에 대해 생성한 AWS KMS 키가 아직 존재하지 않아 ARN을 참조할 수 없기 때문에 이 계정이 필요합니다. * 또한 Security Hub는 정책 설명의 해당 `Condition` 조항을 추론에 포함시키지 않습니다.

이 결과를 수정하려면 어떻게 해야 하나요? AWS KMS 키의 리소스 정책이 불필요하게 허용되지 않는 한, 현재 역할 정책으로는 배포 역할이 필요 이상으로 많은 데이터에 액세스하는 것을 허용하지 않습니다. 그래도 검색 결과를 없애고 싶다면 다음 두 가지 방법 중 하나로 부트스트랩 스택을 사용자 지정 (위에 설명된 프로세스 사용) 하면 됩니다.

- 계정 간 배포에 AWS CDK 파이프라인을 사용하지 않는 경우: 배포 역할에서 명령문을 삭제하거나 `Sid: PipelineCrossAccountArtifactsBucket`
- 계정 간 배포에 AWS CDK 파이프라인을 사용하는 경우: AWS CDK 파이프라인을 배포한 후 Artifact 버킷의 Key AWS KMS ARN을 찾아 명령문을 실제 Key ARN으로 `Resource: *` 바꾸십시오. `Sid: PipelineCrossAccountArtifactsBucket`

리소스

리소스는 AWS 서비스 애플리케이션에서 사용하도록 구성한 것입니다. 리소스는 의 기능입니다 AWS CloudFormation. AWS CloudFormation 템플릿에서 리소스와 해당 속성을 구성하여 리소스를 AWS

CloudFormation 프로비저닝하도록 배포할 수 있습니다. 를 사용하면 AWS Cloud Development Kit (AWS CDK)구문을 통해 리소스를 구성할 수 있습니다. 그런 다음 CDK 앱을 배포합니다. 여기에는 AWS CloudFormation 템플릿을 합성하고 리소스를 AWS CloudFormation 프로비저닝하기 위한 배포가 포함됩니다.

주제

구문을 사용하여 리소스를 구성합니다.

에 the section called “구조물” 설명된 대로 는 모든 리소스를 나타내는 구문이라고 하는 풍부한 클래스 AWS 구문 라이브러리를 AWS CDK 제공합니다. AWS

해당하는 구문을 사용하여 리소스의 인스턴스를 만들려면 범위에 첫 번째 인수로, 구문의 논리적 ID, 구성 속성 집합 (props) 을 전달해야 합니다. 예를 들어, 다음은 구성 라이브러리의 SQS.queue 구문을 사용하여 AWS KMS 암호화가 적용된 Amazon SQS 대기열을 생성하는 방법입니다. 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
});
```

일부 구성 소품은 선택 사항이며 대부분의 경우 기본값이 있습니다. 경우에 따라 모든 props는 선택 사항이며 마지막 인수는 완전히 생략할 수 있습니다.

Resource attributes

AWS 구성 라이브러리에 있는 대부분의 리소스는 속성을 노출하는데, 이러한 속성은 배포 시 다음을 통해 해결됩니다. AWS CloudFormation속성은 유형 이름을 접두사로 사용하는 리소스 클래스에 속성 형태로 표시됩니다. 다음 예제는 (queueUrlPython:queue_url) 속성을 사용하여 Amazon SQS 대기열의 URL을 가져오는 방법을 보여줍니다.

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
```

배포 시간 속성을 문자열로 AWS CDK 인코딩하는 방법에 the section called “토큰” 대한 자세한 내용은 을 참조하십시오.

리소스 참조

리소스를 구성할 때 다른 리소스의 속성을 참조해야 하는 경우가 많습니다. 예를 들어, 다음과 같습니다.

- Amazon Elastic Container Service (Amazon ECS) 리소스에는 해당 리소스가 실행되는 클러스터에 대한 참조가 필요합니다.
- Amazon CloudFront 배포에는 소스 코드가 들어 있는 Amazon Simple Storage 서비스 (Amazon S3) 버킷에 대한 참조가 필요합니다.

다음 방법 중 하나로 리소스를 참조할 수 있습니다.

- CDK 앱에 정의된 리소스를 동일한 스택이나 다른 스택에 전달하여
- 리소스의 고유 식별자 (예: ARN) 에서 생성된 AWS 계정에 정의된 리소스를 참조하는 프록시 객체를 전달함으로써

구성의 속성이 다른 리소스의 구성을 나타내는 경우 해당 구성의 유형은 해당 구성의 인터페이스 유형 유형입니다. 예를 들어 Amazon ECS 구문은 다음과 같은 `cluster` 유형의 `ecs.ICluster` 속성을 사용합니다. 또 다른 예로는 유형의 속성 `sourceBucket` (Python:`source_bucket`) 을 취하는 CloudFront 배포 구문을 들 수 `s3.IBucket` 있습니다.

동일한 AWS CDK 앱에 정의된 적절한 유형의 리소스 객체를 직접 전달할 수 있습니다. 다음 예제는 Amazon ECS 클러스터를 정의한 다음 이를 사용하여 Amazon ECS 서비스를 정의합니다.

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 });
```

다른 스택의 리소스 참조

리소스가 동일한 앱에 정의되어 있고 동일한 AWS 환경에 있는 한 다른 스택의 리소스를 참조할 수 있습니다. 일반적으로 다음과 같은 패턴이 사용됩니다.

- 구문에 대한 참조를 리소스를 생성하는 스택의 속성으로 저장합니다. (현재 구문의 스택에 대한 참조를 가져오려면 를 사용하십시오`Stack.of(this)`.)
- 리소스를 매개 변수 또는 속성으로 사용하는 스택의 생성자에 이 참조를 전달하십시오. 그런 다음 소비 스택은 해당 스택을 필요로 하는 모든 구문에 해당 스택을 속성으로 전달합니다.

다음 예제에서는 스택을 정의합니다`stack1`. 이 스택은 Amazon S3 버킷을 정의하고 버킷 구조에 대한 참조를 스택의 속성으로 저장합니다. 그런 다음 앱은 인스턴스화 시 버킷을 허용하는 두 번째 스택을 정의합니다. `stack2` `stack2`예를 들어 버킷을 데이터 AWS Glue 스토리지로 사용하는 테이블을 정의할 수 있습니다.

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});
```

리소스가 동일한 환경에 있지만 다른 스택에 있다고 AWS CDK 판단되면 생성 스택의 AWS CloudFormation 내보내기와 소비 스택의 Fn:: ImportValue 를 자동으로 합성하여 해당 정보를 한 스택에서 다른 스택으로 전송합니다.

종속성 교착 상태 해결

다른 스택의 한 스택에서 리소스를 참조하면 두 스택 사이에 종속성이 생깁니다. 이렇게 하면 올바른 순서로 배포될 수 있습니다. 스택을 배포한 후에는 이러한 종속성이 명확해집니다. 그런 다음 소비 스택에서 공유 리소스 사용을 제거하면 예기치 않은 배포 실패가 발생할 수 있습니다. 이 문제는 두 스택 사이에 또 다른 종속성이 있어 두 스택을 동일한 순서로 배포해야 하는 경우에 발생합니다. CDK 툴킷에서 단순히 프로덕션 스택을 선택하여 먼저 배포하는 경우에도 종속성 없이 발생할 수 있습니다. AWS CloudFormation 익스포트는 더 이상 필요하지 않으므로 프로덕션 스택에서 제거되지만, 익스포트된 리소스는 업데이트가 아직 배포되지 않았으므로 사용 스택에서 계속 사용됩니다. 따라서 프로듀서 스택 배포가 실패합니다.

이러한 교착 상태를 해결하려면 소비 스택에서 공유 리소스 사용을 제거하세요. (이렇게 하면 프로덕션 스택에서 자동 내보내기가 제거됩니다.) 그런 다음, 자동으로 생성된 내보내기와 정확히 동일한 논리 ID를 사용하여 동일한 익스포트를 생산 스택에 수동으로 추가합니다. 소비 스택에서 공유 리소스를 사용하지 않고 두 스택을 모두 배포하십시오. 그런 다음 수동 내보내기 (더 이상 필요하지 않은 경우 공유 리소스 포함) 를 제거하고 두 스택을 다시 배포하십시오. 스택의 `exportValue()` 방법은 이러한 목적으로 수동 내보내기를 생성할 수 있는 편리한 방법입니다. (링크된 메서드 참조의 예제를 참조하십시오.)

계정의 리소스 참조 AWS

AWS 계정에서 이미 사용 가능한 리소스를 앱에서 사용하고 싶다고 가정해 보겠습니다. AWS CDK 콘솔이나 AWS SDK를 통해 정의하거나 다른 애플리케이션에서 직접 정의하거나 다른 AWS CDK 애플리케이션에서 정의한 리소스일 수 있습니다. AWS CloudFormation리소스의 ARN (또는 다른 식별 속성 또는 속성 그룹) 을 프록시 객체로 전환할 수 있습니다. 프록시 객체는 리소스 클래스에서 정적 팩토리 메서드를 호출하여 리소스에 대한 참조 역할을 합니다.

이러한 프록시를 만들면 외부 리소스가 AWS CDK 앱의 일부가 되지 않습니다. 따라서 AWS CDK 앱에서 프록시를 변경해도 배포된 리소스에는 영향을 주지 않습니다. 하지만 프록시는 해당 유형의 리소스가 필요한 모든 AWS CDK 메서드에 전달될 수 있습니다.

다음 예제는 ARN arn:aws:s3: ::가 있는 기존 버킷과 특정 ID를 가진 기존 VPC를 기반으로 하는 Amazon my-bucket-name Virtual Private Cloud를 기반으로 하는 버킷을 참조하는 방법을 보여줍니다.

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"
});
```

메서드를 좀 더 자세히 살펴보겠습니다. `Vpc.fromLookup()` `ec2.Vpc`구조가 복잡하기 때문에 CDK 앱과 함께 사용할 VPC를 선택하는 방법은 여러 가지가 있습니다. 이 문제를 해결하기 위해 VPC 구조에는 합성 시 계정을 쿼리하여 원하는 Amazon VPC를 조회할 수 있는 `fromLookup` 정적 메서드 (Python:`from_lookup`) 가 있습니다. AWS

사용하려면 `Vpc.fromLookup()` 스택을 합성하는 시스템이 Amazon VPC를 소유한 계정에 액세스할 수 있어야 합니다. 이는 CDK 툴킷이 통합 시 계정을 쿼리하여 적합한 Amazon VPC를 찾기 때문입니다.

또한 명시적 계정 및 지역으로 정의된 스택에서만 **`Vpc.fromLookup()`** 작동합니다 (참조). the section called “환경” 환경에 구애받지 않는 스택에서 Amazon VPC를 AWS CDK 검색하려고 하면 CDK Toolkit은 VPC를 찾기 위해 어떤 환경을 쿼리해야 하는지 알지 못합니다.

계정에서 VPC를 고유하게 식별할 수 있을 만큼 충분한 `Vpc.fromLookup()` 속성을 제공해야 합니다. AWS 예를 들어 기본 VPC는 한 개만 있을 수 있으므로 VPC를 기본 VPC로 지정하는 것으로 충분합니다.

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 });
```

`tags`속성을 사용하여 태그별로 VPC를 쿼리할 수도 있습니다. Amazon VPC를 만들 때 AWS CloudFormation 또는 를 사용하여 태그를 추가할 수 있습니다. AWS CDK태그를 생성한 후 언제든지 AWS Management Console AWS CLI, 또는 AWS SDK를 사용하여 태그를 편집할 수 있습니다. 직접 추가한 태그 외에도 는 생성하는 모든 VPC에 다음 태그를 AWS CDK 자동으로 추가합니다.

- 이름 — VPC의 이름입니다.
- aws-cdk:서브넷 이름 — 서브넷의 이름입니다.
- aws-cdk:subnet-type — 서브넷 유형: 퍼블릭, 프라이빗 또는 격리형 서브넷 유형.

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" });
```

의 결과는 프로젝트 파일에 캐시됩니다. `Vpc.fromLookup()` `cdk.context.json` (the section called “컨텍스트”를 참조하세요.) 앱이 동일한 Amazon VPC를 계속 참조할 수 있도록 이 파일을 버전 관리에 커밋하십시오. 이는 나중에 VPC의 속성을 변경하여 다른 VPC가 선택되도록 하는 경우에도 작동합니다. 이는 CDK Pipelines와 같이 VPC를 정의하는 AWS 계정에 액세스할 수 없는 환경에 스택을 배포하는 경우 특히 중요합니다.

AWS CDK 앱에 정의된 유사한 리소스를 사용하는 곳 어디에서나 외부 리소스를 사용할 수 있지만 수정할 수는 없습니다. 예를 들어 외부에서 `addToResourcePolicy` (Python:`add_to_resource_policy`) 를 `s3.Bucket` 호출해도 아무 작업도 수행되지 않습니다.

리소스 물리적 이름

에 AWS CloudFormation 있는 리소스의 논리적 이름은 배포한 AWS Management Console 다음에 표시되는 리소스 이름과 다릅니다 AWS CloudFormation. 이 AWS CDK 호출은 이러한 최종 이름을 물리적 이름으로 부릅니다.

예를 AWS CloudFormation 들어, 물리적 이름을 가진 이전 예제의 논리적 ID를 `Stack2MyBucket4DD88B4F` 사용하여 Amazon S3 버킷을 생성할 수 `stack2mybucket4dd88b4f-iuv1rbv9z3to` 있습니다.

Name 속성을 사용하여 리소스를 나타내는 구성을 작성할 때 물리적 <resourceType>이름을 지정할 수 있습니다. 다음 예제는 물리적 이름을 사용하여 Amazon S3 버킷을 생성합니다`my-bucket-name`.

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

리소스에 물리적 이름을 할당하면 몇 가지 단점이 있습니다. AWS CloudFormation가장 중요한 것은 생성 후 변경할 수 없는 리소스 속성 변경과 같이 리소스 교체가 필요한 배포된 리소스에 대한 모든 변경 사항은 리소스에 물리적 이름이 할당된 경우 실패한다는 것입니다. 이 상태가 되면 AWS CloudFormation 스택을 삭제한 다음 AWS CDK 앱을 다시 배포하는 것만이 유일한 해결책입니다. 자세한 내용은 AWS CloudFormation 설명서를 참조하십시오.

환경 간 참조를 사용하여 AWS CDK 앱을 만드는 경우와 같이 일부 경우에는 가 제대로 AWS CDK 작동하려면 물리적 이름이 필요합니다. 이러한 경우 실제 이름을 직접 만들고 싶지 않다면 이름을 대신 지정하면 됩니다 AWS CDK . 이렇게 하려면 다음과 같이 특수 값을 `PhysicalName.GENERATE_IF_NEEDED` 사용하십시오.

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 });
```

고유한 리소스 식별자 전달

가능하면 이전 섹션에서 설명한 대로 참조를 통해 리소스를 전달해야 합니다. 하지만 속성 중 하나로 리소스를 참조하는 것 외에는 다른 선택의 여지가 없는 경우도 있습니다. 사용 사례의 예는 다음과 같습니다.

- 저수준 AWS CloudFormation 리소스를 사용하는 경우.
- 환경 변수를 통해 Lambda 함수를 참조하는 경우와 같이 AWS CDK 애플리케이션의 런타임 구성 요소에 리소스를 노출해야 하는 경우

이러한 식별자는 다음과 같이 리소스의 속성으로 사용할 수 있습니다.

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

Java AWS CDK 바인딩은 속성에 게터 메서드를 사용합니다.

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

C#

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

다음 예제는 생성된 버킷 이름을 AWS Lambda 함수에 전달하는 방법을 보여줍니다.

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
    }
});
```

리소스 간 권한 부여

상위 수준 구조는 권한 요구 사항을 명시하는 간단한 인텐트 기반 API를 제공하여 최소 권한 달성을 가능하게 합니다. 예를 들어, 많은 L2 구조는 IAM 권한 설명을 수동으로 생성하지 않고도 개체 (예: IAM 역할 또는 사용자) 에게 리소스를 사용할 수 있는 권한을 부여하는 데 사용할 수 있는 권한 부여 방법을 제공합니다.

다음 예제는 Lambda 함수의 실행 역할이 특정 Amazon S3 버킷에 객체를 읽고 쓸 수 있도록 권한을 생성합니다. Amazon S3 버킷이 AWS KMS 키로 암호화된 경우, 이 메서드는 Lambda 함수의 실행 역할에 키로 복호화할 수 있는 권한도 부여합니다.

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)
{
    // ...
}
```

권한 부여 메서드는 객체를 반환합니다. `iam.Grant` `Grant`객체의 `success` 속성을 사용하여 권한 부여가 효과적으로 적용되었는지 여부를 확인할 수 있습니다 (예: 외부 리소스에 적용되지 않았을 수 있음). `Grant`객체의 `assertSuccess` (Python:`assert_success`) 메서드를 사용하여 권한 부여가 성공적으로 적용되었는지 확인할 수도 있습니다.

특정 사용 사례에 특정 권한 부여 방법을 사용할 수 없는 경우 일반 권한 부여 방법을 사용하여 지정된 작업 목록으로 새 권한 부여를 정의할 수 있습니다.

다음 예제는 Lambda 함수에 Amazon DynamoDB 작업에 대한 액세스 권한을 부여하는 방법을 보여줍니다. `CreateBackup`

TypeScript

```
table.grant(func, 'dynamodb:CreateBackup');
```

JavaScript

```
table.grant(func, 'dynamodb:CreateBackup');
```

Python

```
table.grant(func, "dynamodb:CreateBackup")
```

Java

```
table.grant(func, "dynamodb:CreateBackup");
```

C#

```
table.Grant(func, "dynamodb:CreateBackup");
```

Lambda 함수와 같은 많은 리소스는 코드를 실행할 때 역할을 맡아야 합니다. 구성 속성을 사용하면 를 지정할 수 있습니다. `iam.IRole` 역할을 지정하지 않은 경우 함수는 해당 용도에 맞는 역할을 자동으로 생성합니다. 그런 다음 리소스에서 부여 메서드를 사용하여 역할에 명령문을 추가할 수 있습니다.

권한 부여 방법은 IAM 정책 처리를 위한 하위 수준 API를 사용하여 구축됩니다. 정책은 객체로 모델링됩니다. PolicyDocument 메서드 (Python:) 를 사용하여 역할 (또는 구문의 연결된 역할) 에 명령문을 직접 추가하거나 (Python:`add_to_role_policy`) `addToRolePolicy` 메서드를 사용하여 리소스의 `Bucket` 정책 `addToResourcePolicy` (예: 정책`add_to_resource_policy`) 에 명령문을 추가합니다.

리소스 메트릭 및 알람

많은 리소스에서 모니터링 대시보드 및 경보를 설정하는 데 사용할 수 있는 CloudWatch 지표를 내보냅니다. 상위 수준 구조에는 사용할 올바른 이름을 찾지 않고도 지표에 액세스할 수 있는 지표 메서드가 있습니다.

다음 예제는 Amazon SQS 대기열의 수가 100을 `ApproximateNumberOfMessagesNotVisible` 초과할 때 경보를 정의하는 방법을 보여줍니다.

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,
    // ..
});
```

특정 측정치에 대한 방법이 없는 경우 일반 측정치 방법을 사용하여 측정치 이름을 수동으로 지정할 수 있습니다.

CloudWatch 대시보드에 지표를 추가할 수도 있습니다. CloudWatch 섹션을 참조하세요.

네트워크 트래픽

대부분의 경우, 컴퓨팅 인프라가 지속성 계층에 액세스해야 하는 경우와 같이 애플리케이션이 작동하려면 네트워크에서 권한을 활성화해야 합니다. 연결을 설정하거나 수신 대기하는 리소스는 보안 그룹 규칙 또는 네트워크 ACL 설정을 포함하여 트래픽 흐름을 가능하게 하는 방법을 노출합니다.

IConnectable 리소스에는 네트워크 트래픽 규칙 `connections` 구성의 게이트웨이인 속성이 있습니다.

메서드를 사용하여 지정된 네트워크 경로를 통해 데이터가 흐르도록 할 수 있습니다. `allow` 다음 예제는 웹에 대한 HTTPS 연결과 Amazon EC2 Auto Scaling 그룹에서 들어오는 연결을 활성화합니다. `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());
```

특정 리소스에는 기본 포트가 연결되어 있습니다. 퍼블릭 포트의 로드 밸런서 리스너, 데이터베이스 엔진이 Amazon RDS 데이터베이스 인스턴스 연결을 수락하는 포트 등을 예로 들 수 있습니다. 이 경우 포트를 수동으로 지정하지 않고도 네트워크를 엄격하게 제어할 수 있습니다. 이렇게 하려면 `allowDefaultPortFrom` 및 `allowToDefaultPort` 메서드 (Python:`allow_default_port_from`,`allow_to_default_port`) 를 사용하십시오.

다음 예는 모든 IPV4 주소의 연결과 Auto Scaling 그룹의 연결을 활성화하여 데이터베이스에 액세스할 수 있도록 하는 방법을 보여줍니다.

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");
```

이벤트 처리

일부 리소스는 이벤트 소스로 작동할 수 있습니다. (Python:`add_event_notification`) `addEventNotification` 메서드를 사용하여 리소스에서 내보내는 특정 이벤트 유형에 이벤트 대상을 등록합니다. 이 외에도 `addXxxNotification` 메서드는 일반적인 이벤트 유형에 대한 핸들러를 등록하는 간단한 방법을 제공합니다.

다음 예제는 객체가 Amazon S3 버킷에 추가될 때 Lambda 함수를 트리거하는 방법을 보여줍니다.

TypeScript

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

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

JavaScript

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

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

Python

```
import aws_cdk.aws_s3_notifications as s3_nots
```

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

Java

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

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

C#

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

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

제거 정책

데이터베이스, Amazon S3 버킷, Amazon ECR 레지스트리와 같이 영구 데이터를 유지 관리하는 리소스에는 제거 정책이 있습니다. 제거 정책은 영구 객체를 포함하는 AWS CDK 스택이 삭제될 때 영구 객체를 삭제할지 여부를 나타냅니다. 제거 정책을 지정하는 값은 모듈의 `RemovalPolicy` 열거를 통해 사용할 수 있습니다. AWS CDK `core`

> **Note**
>
> 데이터를 영구적으로 저장하는 리소스 이외의 리소스에는 다른 용도로 사용되는 리소스가 `removalPolicy` 있을 수도 있습니다. 예를 들어, Lambda 함수 버전은 속성을 사용하여 `removalPolicy` 새 버전이 배포될 때 지정된 버전을 유지할지 여부를 결정합니다. 이는 Amazon S3 버킷 또는 DynamoDB 테이블의 제거 정책과 다른 의미와 기본값을 갖습니다.

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

AWS CloudFormation 제거 정책이 로 설정되어 있더라도 파일이 포함된 Amazon S3 버킷은 제거하지 않습니다. `DESTROY` 그렇게 하려고 하면 오류가 발생합니다. AWS CloudFormation 버킷을 AWS CDK 삭제하기 전에 버킷에서 모든 파일을 삭제하도록 하려면 버킷의 `autoDeleteObjects` 속성을 로 설정하십시오. `true`

다음은 `RemovalPolicy` of `DESTROY` 및 로 `autoDeleteOjbects` 설정된 Amazon S3 버킷을 생성하는 예제입니다`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
    });
}
```

`applyRemovalPolicy()`메서드를 통해 기본 AWS CloudFormation 리소스에 직접 제거 정책을 적용할 수도 있습니다. 이 메서드는 L2 리소스의 props에 `removalPolicy` 속성이 없는 일부 스테이트풀 리소스에서 사용할 수 있습니다. 예는 다음과 같습니다.

- AWS CloudFormation 스택
- Amazon Cognito 사용자 풀
- 아마존 DocumentDB 데이터베이스 인스턴스
- 아마존 EC2 볼륨
- 아마존 OpenSearch 서비스 도메인
- 아마존 FSx 파일 시스템
- Amazon SQS 대기열

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**
>
> AWS CDK는 로 RemovalPolicy 번역됩니다. AWS CloudFormationDeletionPolicy 하지만 기본값은 데이터를 보존하는 것인데, 이는 AWS CloudFormation 기본값과 반대입니다. AWS CDK

식별자

AWS Cloud Development Kit (AWS CDK) 앱을 빌드할 때는 다양한 유형의 식별자와 이름을 사용합니다. AWS CDK 효과적으로 사용하고 오류를 방지하려면 식별자의 유형을 이해하는 것이 중요합니다.

식별자는 생성되는 범위 내에서 고유해야 하며 애플리케이션에서 전체적으로 고유할 필요는 없습니다. AWS CDK

동일한 범위 내에서 값이 같은 식별자를 만들려고 하면 에서 AWS CDK 예외가 발생합니다.

주제

구성 ID

가장 일반적인 식별자는 구성 `id` 객체를 인스턴스화할 때 두 번째 인수로 전달되는 식별자입니다. 모든 식별자와 마찬가지로 이 식별자는 생성된 범위 내에서만 고유해야 합니다. 이 식별자는 구성 객체를 인스턴스화할 때 첫 번째 인수입니다.

> **Note**
>
> 스택은 에서 해당 스택을 참조할 때 사용하는 식별자이기도 합니다. `id` the section called "AWS CDK 툴킷"

앱에 식별자를 `MyBucket` 포함하는 구문이 두 개 있는 예를 살펴보겠습니다. 첫 번째는 식별자를 `Stack1` 사용하여 스택 범위에 정의됩니다. 두 번째는 스택 범위에서 식별자와 함께 `Stack2` 정의됩니다. 서로 다른 범위에서 정의되므로 충돌이 발생하지 않으며 동일한 앱에서 문제 없이 공존할 수 있습니다.

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");
    }
}
```

경로

AWS CDK 애플리케이션의 구문은 클래스를 기반으로 하는 계층 구조를 형성합니다. App 주어진 구문, 부모 구문, 최상위 구문 등에서 구문 트리의 루트까지 이어지는 ID 모음을 경로라고 합니다.

는 AWS CDK 일반적으로 템플릿의 경로를 문자열로 표시합니다. 레벨의 ID는 일반적으로 스택인 루트 App 인스턴스 바로 아래의 노드에서 시작하여 슬래시로 구분됩니다. 예를 들어, 이전 코드 예제에서 두 Amazon S3 버킷 리소스의 경로는 `Stack1/MyBucket` 및 `Stack2/MyBucket` 입니다.

다음 예제와 같이 프로그래밍 방식으로 모든 구문의 경로에 액세스할 수 있습니다. 이것은 `myConstruct` (또는 `my_construct` Python 개발자가 작성한 것처럼) 의 경로를 가져옵니다. ID는 생성되는 범위 내에서 고유해야 하므로 ID의 경로는 AWS CDK 응용 프로그램 내에서 항상 고유합니다.

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;
```

고유 ID

AWS CloudFormation 템플릿의 모든 논리적 ID가 고유해야 합니다. 따라서 는 응용 프로그램의 각 구문에 대해 고유한 식별자를 생성할 수 AWS CDK 있어야 합니다. 리소스에는 전체적으로 고유한 경로(스택에서 특정 리소스까지의 모든 범위 이름) 가 있습니다. 따라서 는 경로의 요소를 연결하고 8자리 해시를 추가하여 필요한 고유 식별자를 AWS CDK 생성합니다. (해시는 `A/B/C` 및 `A/BC` 와 같이 서로 다른 경로를 구분하는 데 필요합니다. 그러면 동일한 식별자가 됩니다. AWS CloudFormation AWS CloudFormation 식별자는 영숫자이며 슬래시나 기타 구분 문자를 포함할 수 없습니다. 는 이 문자열을 AWS CDK 구문의 고유 ID를 호출합니다.

일반적으로 AWS CDK 앱은 고유 ID에 대해 알 필요가 없습니다. 하지만 다음 예제와 같이 프로그래밍 방식으로 모든 구문의 고유 ID에 액세스할 수 있습니다.

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);
```

주소는 CDK 리소스를 고유하게 구분하는 또 다른 종류의 고유 식별자입니다. 경로의 SHA-1 해시에서 파생된 이 주소는 사람이 읽을 수 없습니다. 그러나 길이가 일정하고 비교적 짧기 때문에 (항상 42개의 16진수 문자) "기존" 고유 ID가 너무 길 수 있는 상황에서 유용합니다. 일부 구문은 고유 ID 대신 합성된 AWS CloudFormation 템플릿의 주소를 사용할 수 있습니다. 다시 말하지만, 앱은 일반적으로 구문의 주소를 알 필요가 없지만 다음과 같이 구문의 주소를 검색할 수 있습니다.

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;
```

논리적 ID

고유 ID는 리소스를 나타내는 AWS 구성에 대해 생성된 AWS CloudFormation 템플릿에서 리소스의 논리적 식별자 (또는 논리적 이름) 역할을 합니다.

예를 들어, 이전 예제에서 Amazon S3 버킷이 내에 `Stack2` 생성되어 `AWS::S3::Bucket` 리소스가 생성됩니다. 리소스의 논리적 ID는 결과 `Stack2MyBucket4DD88B4F` AWS CloudFormation 템플릿에 있습니다. (이 식별자 생성 방법에 대한 자세한 내용은 을 참조하십시오the section called "고유 ID".)

논리적 ID 안정성

리소스가 생성된 후에는 리소스의 논리적 ID를 변경하지 마십시오. AWS CloudFormation 논리적 ID로 리소스를 식별합니다. 따라서 리소스의 논리 ID를 변경하면 새 논리 ID로 AWS CloudFormation 새 리

소스가 생성된 다음 기존 리소스가 삭제됩니다. 리소스 유형에 따라 이로 인해 서비스가 중단되거나 데이터가 손실되거나 둘 다 발생할 수 있습니다.

토큰

토큰은 앱 수명 주기의 나중에만 확인할 수 있는 값을 나타냅니다. 예를 들어 CDK 앱에서 정의하는 Amazon Simple Storage Service (Amazon S3) 버킷의 이름은 템플릿이 합성될 때만 AWS CloudFormation 할당됩니다. 문자열인 `bucket.bucketName` 속성을 인쇄하면 다음과 같은 내용이 포함된 것을 볼 수 있습니다.

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

생성 당시에는 값이 아직 알려지지 않았지만 나중에 사용할 수 있게 될 토큰을 이 AWS CDK 인코딩하는 방식입니다. 이들은 이를 플레이스홀더 AWS CDK 토큰이라고 부릅니다. 이 경우에는 문자열로 인코딩된 토큰입니다.

이 문자열을 마치 버킷 이름인 것처럼 전달할 수 있습니다. 다음 예시에서는 버킷 이름을 AWS Lambda 함수에 대한 환경 변수로 지정합니다.

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
    }
});
```

AWS CloudFormation 템플릿이 최종적으로 합성되면 토큰은 AWS CloudFormation 내장 `{ "Ref": "MyBucket" }` 토큰으로 렌더링됩니다. 배포 시 이 내장 함수를 생성된 버킷의 실제 이름으로 AWS CloudFormation 대체합니다.

주제

- 토큰 및 토큰 인코딩
- 문자열로 인코딩된 토큰
- 목록으로 인코딩된 토큰
- 숫자로 인코딩된 토큰
- 지연 값
- JSON으로 변환

토큰 및 토큰 인코딩

토큰은 단일 메서드를 포함하는 IreResolvable 인터페이스를 구현하는 객체입니다. `resolve` 는 합성 중에 이 메서드를 AWS CDK 호출하여 템플릿의 최종 값을 생성합니다. AWS CloudFormation 토큰은 합성 프로세스에 참여하여 모든 유형의 임의 값을 생성합니다.

> **Note**
>
> `IResolvable`인터페이스로 직접 작업하는 경우는 거의 없습니다. 문자열로 인코딩된 버전의 토큰만 표시될 가능성이 높습니다.

다른 함수는 일반적으로 또는 같은 기본 유형의 인수만 받아들입니다. `string` `number` 이러한 경우 토큰을 사용하려면 CDK.Token 클래스의 정적 메서드를 사용하여 토큰을 세 가지 유형 중 하나로 인코딩할 수 있습니다.

- `Token.asString`문자열 인코딩 생성 (또는 토큰 객체 호출`.toString()`)
- `Token.asList`목록 인코딩 생성하기
- `Token.asNumber`숫자 인코딩 생성하기

이는 a일 수 있는 임의의 값을 취하여 지정된 유형의 프리미티브 값으로 인코딩합니다. `IResolvable`

> **Important**
>
> 이전 유형 중 하나라도 인코딩된 토큰일 수 있으므로 내용을 파싱하거나 읽을 때는 주의해야 합니다. 예를 들어 문자열을 구문 분석하여 값을 추출하려고 하는데 해당 문자열이 인코딩된 토큰인 경우 구문 분석이 실패합니다. 마찬가지로 배열의 길이를 쿼리하거나 숫자를 사용하여 수학 연산을 수행하려는 경우 먼저 해당 배열이 인코딩된 토큰이 아닌지 확인해야 합니다.

값에 확인되지 않은 토큰이 있는지 확인하려면 `Token.isUnresolved` (Python:`is_unresolved`) 메서드를 호출합니다.

다음 예제에서는 토큰일 수 있는 문자열 값이 10자를 넘지 않는지 확인합니다.

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");
```

이름이 토큰인 경우 검증이 수행되지 않으며 수명 주기의 나중 단계 (예: 배포 중) 에서 여전히 오류가 발생할 수 있습니다.

> **Note**
>
> 토큰 인코딩을 사용하여 형식 시스템을 이스케이프할 수 있습니다. 예를 들어 합성 시 숫자 값을 생성하는 토큰을 문자열 인코딩할 수 있습니다. 이러한 함수를 사용하는 경우 합성 후 템플릿이 사용 가능한 상태로 변환되는지 확인하는 것은 사용자의 책임입니다.

문자열로 인코딩된 토큰

문자열로 인코딩된 토큰은 다음과 같습니다.

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

다음 예제와 같이 일반 문자열처럼 전달되고 연결될 수 있습니다.

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";
```

다음 예제와 같이 해당 언어에서 문자열 보간을 지원하는 경우 문자열 보간을 사용할 수도 있습니다.

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";
```

다른 방법으로 문자열을 조작하지 마세요. 예를 들어 문자열의 하위 문자열을 가져오면 문자열 토큰이 손상될 수 있습니다.

목록으로 인코딩된 토큰

목록으로 인코딩된 토큰은 다음과 같습니다.

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

이러한 목록을 사용하여 수행할 수 있는 유일한 안전한 방법은 다른 구조에 직접 전달하는 것입니다. 문자열 목록 형식의 토큰은 연결할 수 없으며 토큰에서 요소를 가져올 수도 없습니다. 안전하게 조작할 수 있는 유일한 방법은 fn.select와 같은 AWS CloudFormation 내장 함수를 사용하는 것입니다.

숫자로 인코딩된 토큰

숫자로 인코딩된 토큰은 다음과 같은 작은 음수 부동 소수점 숫자 집합입니다.

```
-1.8881545897087626e+289
```

리스트 토큰과 마찬가지로 숫자 값을 수정할 수 없습니다. 이렇게 하면 숫자 토큰이 손상될 수 있기 때문입니다. 허용되는 유일한 작업은 값을 다른 구문으로 전달하는 것입니다.

지연 값

토큰은 AWS CloudFormation 매개변수와 같은 배포 시간 값을 나타내는 것 외에도 합성 시간의 지연 값을 나타내는 데에도 일반적으로 사용됩니다. 이러한 값은 합성이 완료되기 전에 최종 값이 결정되지만 값이 구성되는 시점에서는 결정되지 않습니다. 토큰을 사용하여 리터럴 문자열이나 숫자 값을 다른 구문에 전달하는 반면, 합성 시점의 실제 값은 아직 발생하지 않은 일부 계산에 따라 달라질 수 있습니다.

Lazy클래스의 정적 메서드 (예: Lazy.String 및 Lazy.Number) 를 사용하여 신시사이저 타임의 지연 값을 나타내는 토큰을 생성할 수 있습니다. 이러한 메서드는 컨텍스트 인수를 받아들이고 호출 시 최종 값을 반환하는 함수가 produce 속성인 객체를 받아들입니다.

다음 예에서는 생성 후 용량이 결정되는 Auto Scaling 그룹을 생성합니다.

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;
```

JSON으로 변환

임의의 데이터로 구성된 JSON 문자열을 생성하려고 하는데 데이터에 토큰이 포함되어 있는지 모르는 경우가 있습니다. 토큰 포함 여부와 상관없이 데이터 구조를 올바르게 JSON으로 인코딩하려면 메서드 스택을 사용하십시오. toJsonString, 다음 예제와 같습니다.

TypeScript

```
const stack = Stack.of(this);
const str = stack.toJsonString({
  value: bucket.bucketName
});
```

JavaScript

```
const stack = Stack.of(this);
const str = stack.toJsonString({
  value: bucket.bucketName
});
```

Python

```
stack = Stack.of(self)
string = stack.to_json_string(dict(value=bucket.bucket_name))
```

Java

```
Stack stack = Stack.of(this);
```

```
String stringVal = stack.toJsonString(java.util.Map.of(    // Map.of requires Java
 9+
        put("value", bucket.getBucketName())));
```

C#

```
var stack = Stack.Of(this);
var stringVal = stack.ToJsonString(new Dictionary<string, string>
{
    ["value"] = bucket.BucketName
});
```

파라미터

매개변수는 배포 시 제공되는 사용자 지정 값입니다. 매개변수는 의 기능입니다 AWS CloudFormation. AWS CloudFormation 템플릿을 AWS Cloud Development Kit (AWS CDK) 합성하므로 배포 시 파라미터에 대한 지원도 제공합니다.

주제

- 파라미터 정보
- 매개변수 정의
- 파라미터 사용
- 파라미터를 사용하여 배포하기

파라미터 정보

를 AWS CDK사용하여 매개변수를 정의한 다음 작성한 구문의 속성에 사용할 수 있습니다. 파라미터가 포함된 스택을 배포할 수도 있습니다.

AWS CDK 툴킷을 사용하여 AWS CloudFormation 템플릿을 배포할 때는 명령줄에 매개변수 값을 입력합니다. AWS CloudFormation 콘솔을 통해 템플릿을 배포하는 경우 매개변수 값을 입력하라는 메시지가 표시됩니다.

일반적으로 AWS CloudFormation 매개 변수를 와 함께 사용하지 않는 것이 좋습니다 AWS CDK. AWS CDK 앱에 값을 전달하는 일반적인 방법은 컨텍스트 값과 환경 변수입니다. 합성 시에는 매개변수 값을 사용할 수 없으므로 CDK 앱에서 흐름 제어 및 기타 용도로 매개변수 값을 쉽게 사용할 수 없습니다.

> **Note**
>
> CfnCondition구문을 사용하여 파라미터를 사용하여 흐름을 제어할 수 있지만, 이는 네이티브 명령문에 비해 어색하긴 하지만 말입니다. `if`

매개 변수를 사용하려면 작성 중인 코드가 배포 시점과 합성 시 어떻게 동작하는지 염두에 두어야 합니다. 이로 인해 AWS CDK 응용 프로그램을 이해하고 추론하기가 더 어려워지지만 대부분의 경우 별 도움이 되지 않습니다.

일반적으로 CDK 앱이 필요한 정보를 잘 정의된 방식으로 받아들이고 이를 직접 사용하여 CDK 앱에서 구문을 선언하도록 하는 것이 좋습니다. 이상적인 AWS CDK생성 AWS CloudFormation 템플릿은 구체적이며 배포 시 지정할 값이 남아 있지 않습니다.

그러나 AWS CloudFormation 매개변수가 고유하게 적합한 사용 사례가 있습니다. 예를 들어 인프라를 정의하고 배포하는 별도의 팀이 있는 경우 매개변수를 사용하여 생성된 템플릿을 더 광범위하게 유용하게 사용할 수 있습니다. 또한 AWS CloudFormation 매개 변수가 AWS CDK 지원되므로 AWS CloudFormation 템플릿을 사용하는 AWS 서비스 (예: Service Catalog) 에서 를 사용할 수 있습니다. AWS CDK 이러한 AWS 서비스는 매개 변수를 사용하여 배포되는 템플릿을 구성합니다.

매개변수 정의

CfnParameter클래스를 사용하여 매개변수를 정의합니다. 엄밀히 따지자면 둘 다 선택사항이긴 하지만 대부분의 매개 변수에 대해 최소한 한 가지 유형과 설명을 지정하는 것이 좋습니다. 설명은 AWS CloudFormation 콘솔에서 사용자에게 매개변수 값을 입력하라는 메시지가 표시될 때 나타납니다. 사용 가능한 유형에 대한 자세한 내용은 유형을 참조하십시오.

> **Note**
>
> 모든 범위에서 매개변수를 정의할 수 있습니다. 하지만 코드를 리팩토링할 때 논리 ID가 변경되지 않도록 스택 수준에서 매개변수를 정의하는 것이 좋습니다.

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"
});
```

파라미터 사용

`CfnParameter`인스턴스는 토큰을 통해 해당 값을 AWS CDK 앱에 노출합니다. 모든 토큰과 마찬가지로 파라미터의 토큰도 합성 시 확인됩니다. 하지만 이는 구체적인 값이 아니라 AWS CloudFormation 템플릿에 정의된 파라미터 (배포 시 확인됨) 에 대한 참조로 해석됩니다.

토큰은 `Token` 클래스의 인스턴스나 문자열, 문자열 목록 또는 숫자 인코딩으로 검색할 수 있습니다. 선택 사항은 매개 변수를 사용할 클래스 또는 메서드에 필요한 값의 종류에 따라 달라집니다.

TypeScript

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

JavaScript

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

Python

Property	kind of value
#	## class instance
`##_as_list`	The token represented as a string list
`##### ##`	The token represented as a number
`## ####`	The token represented as a string

Java

Property	kind of value
`getValue ()`	## class instance
`getValueAs### ()`	The token represented as a string list
`getValueAs## ()`	The token represented as a number
`getValueAs### ()`	The token represented as a string

C#

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

예를 들어, `Bucket` 정의에서 매개변수를 사용하려면

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
};
```

파라미터를 사용하여 배포하기

매개변수가 포함된 생성된 템플릿은 AWS CloudFormation 콘솔을 통해 일반적인 방식으로 배포할 수 있습니다. 각 매개변수의 값을 입력하라는 메시지가 표시됩니다.

AWS CDK 툴킷 (cdk명령줄 도구) 은 배포 시 매개변수 지정도 지원합니다. 명령줄에서 `--parameters` 플래그를 따라 입력합니다. 다음 예제와 같이 `uploadBucketName` 파라미터를 사용하는 스택을 배포할 수 있습니다.

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

여러 매개변수를 정의하려면 여러 `--parameters` 플래그를 사용하십시오.

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

여러 스택을 배포하는 경우 각 스택에 대해 각 매개 변수의 값을 다르게 지정할 수 있습니다. 이렇게 하려면 매개변수 이름 앞에 스택 이름과 콜론을 붙입니다.

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

기본적으로 는 이전 배포의 매개 변수 값을 AWS CDK 보존하고 명시적으로 지정하지 않은 경우 후속 배포에서 해당 값을 사용합니다. `--no-previous-parameters`플래그를 사용하면 모든 파라미터를 지정해야 합니다.

태그 지정

태그는 앱의 구문에 추가할 수 있는 정보용 키-값 요소입니다. AWS CDK 지정된 구문에 적용된 태그는 태그가 지정될 수 있는 모든 하위 구성요소에도 적용됩니다. 태그는 앱에서 합성된 AWS CloudFormation 템플릿에 포함되며 앱이 배포하는 AWS 리소스에 적용됩니다. 태그를 사용하여 다음과 같은 목적으로 리소스를 식별하고 분류할 수 있습니다.

- 관리 단순화
- 비용 할당
- 액세스 제어
- 사용자가 고안한 기타 모든 목적

> **Tip**
>
> 리소스에 태그를 사용하는 방법에 대한 자세한 내용은 AWS 백서의 리소스 태그 지정 AWS 모범 사례를 참조하십시오.AWS

주제

태그 사용

`Tags`클래스에는 지정된 구문에 태그를 추가하거나 지정된 구문에서 태그를 제거할 수 있는 정적 메서드가 `of()` 포함되어 있습니다.

- `Tags.of(SCOPE).add()`지정된 구문과 모든 하위 구성요소에 새 태그를 적용합니다.

- `Tags.of(SCOPE).remove()`지정된 구문과 그 하위 구문에서 태그를 제거합니다. 여기에는 하위 구문이 자체적으로 적용했을 수도 있는 태그가 포함됩니다.

> **Note**
>
> 태깅은 를 사용하여 the section called “속성” 구현됩니다. Aspect는 지정된 범위 내의 모든 구문에 작업 (예: 태깅) 을 적용하는 방법입니다.

다음 예제에서는 값 값이 있는 태그 키를 구문에 적용합니다.

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");
```

다음 예제에서는 구문에서 태그 키를 삭제합니다.

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");
```

`Stage`구문을 사용하는 경우 `Stage` 수준 또는 그 아래에 태그를 적용하십시오. 태그는 `Stage` 경계를 넘어 적용되지 않습니다.

태그 우선순위

는 태그를 재귀적으로 AWS CDK 적용하고 제거합니다. 충돌이 발생할 경우 우선 순위가 가장 높은 태깅 작업이 우선합니다. (우선 순위는 선택적 `priority` 속성을 사용하여 설정됩니다.) 두 작업의 우선 순위가 같으면 구성 트리의 맨 아래에 가장 가까운 태깅 작업이 우선합니다. 기본적으로 태그 적용의 우선 순위는 100입니다 (우선 순위가 50인 AWS CloudFormation 리소스에 직접 추가된 태그 제외). 태그 제거의 기본 우선 순위는 200입니다.

다음은 우선 순위가 300인 태그를 구문에 적용합니다.

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 });
```

선택적 속성

태그는 태그가 리소스에 적용되거나 리소스에서 제거되는 방식을 세밀하게 조정할 수 있도록 지원합니다 properties. 모든 속성은 선택 사항입니다.

applyToLaunchedInstances(Python:apply_to_launched_instances)

add () 에만 사용할 수 있습니다. 기본적으로 태그는 Auto Scaling 그룹에서 시작된 인스턴스에 적용됩니다. Auto Scaling 그룹에서 시작된 인스턴스를 무시하려면 이 속성을 false로 설정합니다.

includeResourceTypes/excludeResourceTypes(파이썬:include_resource_types/exclude_resource_types)

AWS CloudFormation 리소스 유형에 따라 리소스의 하위 집합에서만 태그를 조작하려면 이를 사용하십시오. 기본적으로 작업은 구성 하위 트리의 모든 리소스에 적용되지만 특정 리소스 유형을 포함하거나 제외하여 변경할 수 있습니다. 둘 다 지정된 경우 제외가 포함보다 우선합니다.

priority

이를 사용하여 다른 Tags.add() 및 Tags.remove() 작업과 관련하여 이 작업의 우선 순위를 설정할 수 있습니다. 값이 높을수록 낮은 값보다 우선합니다. 기본값은 추가 작업의 경우 100 (AWS CloudFormation 리소스에 직접 적용된 태그의 경우 50), 제거 작업의 경우 200입니다.

다음 예제에서는 값 값과 우선 순위가 100인 태그 태그 이름을 구문에 있는 유형의 AWS::Xxx::Yyy리소스에 적용합니다. Amazon EC2 Auto Scaling 그룹에서 시작된 인스턴스나 유형의 리소스에는 태그를

적용하지 않습니다. AWS::Xxx::Zzz (이는 임의적이지만 서로 다른 두 AWS CloudFormation 리소스 유형에 대한 자리 표시자입니다.)

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

```
Tags.Of(myConstruct).Add("tagname", "value", new TagProps
{
    ApplyToLaunchedInstances = false,
```

```
    IncludeResourceTypes = ["AWS::Xxx::Yyy"],
    ExcludeResourceTypes = ["AWS::Xxx::Zzz"],
    Priority = 100
});
```

다음 예제는 우선순위가 200인 태그 이름을 구문의 유형 AWS::Xxx::Yyy리소스에서는 제거하지만 유형의 리소스에서는 제거하지 않습니다. 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
});
```

예

다음 예제에서는 Stack 이름이 지정된 MarketingSystem 리소스 내에 생성된 모든 TheBest리소스에 값이 StackType있는 태그 키를 추가합니다. 그런 다음 Amazon EC2 VPC 서브넷을 제외한 모든 리소스에서 해당 데이터를 다시 제거합니다. 결과적으로 서브넷에만 태그가 적용됩니다.

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"]
});
```

다음 코드에서도 동일한 결과를 얻을 수 있습니다. 어떤 접근 방식 (포함 또는 제외) 이 의도를 더 명확하게 해주는지 생각해 보세요.

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"]
});
```

단일 구문에 태그 지정하기

`Tags.of(scope).add(key, value)`에서 구문에 태그를 추가하는 표준 방법입니다. AWS CDK주어진 범위 내에서 태그 지정 가능한 모든 리소스에 재귀적으로 태그를 지정하는 트리 워킹 동작은 거의 항상 원하는 방식입니다. 하지만 임의의 특정 구문 (또는 구문) 에 태그를 지정해야 하는 경우가 있습니다.

이러한 경우 중 하나는 태그가 지정되는 구문의 일부 속성에서 값이 파생된 태그를 적용하는 것입니다. 표준 태깅 접근 방식은 범위 내 모든 일치하는 리소스에 동일한 키와 값을 재귀적으로 적용합니다. 하지만 여기서는 태그가 지정된 구조마다 값이 다를 수 있습니다.

태그는 Aspect를 사용하여 구현되며, CDK는 사용자가 사용하여 지정한 범위 내에서 각 구문에 대해 태그의 `visit()` 메서드를 호출합니다. `Tags.of(scope)` `Tag.visit()`직접 호출하여 단일 구문에 태그를 적용할 수 있습니다.

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);
```

범위 내의 모든 구문에 태그를 지정할 수 있지만 태그의 값은 각 구문의 속성에서 파생되도록 할 수 있습니다. 이렇게 하려면 이전 예제와 같이 애스펙트를 작성하고 애스펙트의 `visit()` 메서드에 태그를 적용하세요. 그런 다음 를 사용하여 `Aspects.of(scope).add(aspect)` 원하는 범위에 애스펙트를 추가합니다.

다음 예제에서는 리소스 경로가 포함된 스택의 각 리소스에 태그를 적용합니다.

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**
>
> 우선 순위, 리소스 유형 등을 포함한 조건부 태깅 로직이 클래스에 내장되어 있습니다. `Tag` 임의의 리소스에 태그를 적용할 때 이러한 기능을 사용할 수 있습니다. 조건이 충족되지 않으면 태그가 적용되지 않습니다. 또한 `Tag` 클래스는 태그가 가능한 리소스에만 태그를 지정하므로 태그를 적용하기 전에 구문에 태깅이 가능한지 테스트할 필요가 없습니다.

자산

자산은 라이브러리와 앱에 번들로 묶을 수 있는 로컬 파일, 디렉터리 또는 Docker 이미지입니다. AWS CDK 예를 들어, 에셋은 함수의 핸들러 코드가 포함된 디렉터리일 수 있습니다. AWS Lambda 에셋은 앱이 작동하는 데 필요한 모든 아티팩트를 나타낼 수 있습니다.

다음 튜토리얼 동영상은 CDK 자산에 대한 포괄적인 개요를 제공하고 코드형 인프라 (IAc) 에서 CDK 자산을 사용하는 방법을 설명합니다.

CDK 자산 설명

특정 AWS 구조로 노출되는 API를 통해 자산을 추가합니다. 예를 들어 Lambda.Function 구문을 정의하면 코드 속성을 사용하여 자산 (디렉터리) 을 전달할 수 있습니다. `Function`자산을 사용하여 디렉터리의 콘텐츠를 번들로 묶어 함수 코드에 사용합니다. 마찬가지로 ecs도 마찬가지입니다. ContainerImage.FromAsset은 Amazon ECS 작업 정의를 정의할 때 로컬 디렉터리에서 빌드된 Docker 이미지를 사용합니다.

자산 세부 정보

앱의 자산을 참조할 때 애플리케이션에서 합성된 클라우드 어셈블리에는 AWS CDK CLI에 대한 지침이 포함된 메타데이터 정보가 포함됩니다. 지침에는 로컬 디스크에서 자산을 찾을 위치와 자산 유형에 따라 수행할 번들링 유형 (예: 압축할 디렉터리 (zip) 또는 빌드할 Docker 이미지 등이 포함됩니다.

AWS CDK 는 자산에 대한 소스 해시를 생성합니다. 이는 구성 시 에셋의 내용이 변경되었는지 여부를 확인하는 데 사용할 수 있습니다.

기본적으로 는 클라우드 어셈블리 디렉터리에 자산 사본을 AWS CDK 생성하며, 기본값은 소스 `cdk.out` 해시 아래에 있습니다. 이렇게 하면 클라우드 어셈블리가 독립적이므로 배포를 위해 다른 호스트로 이동해도 배포할 수 있습니다. 세부 정보는 the section called “클라우드 어셈블리”를 참조하세요.

앱 코드에서 직접 또는 라이브러리를 통해 자산을 참조하는 앱을 AWS CDK 배포하면 AWS CDK CLI는 먼저 자산을 준비하여 Amazon S3 버킷 또는 Amazon ECR 리포지토리에 게시합니다. (S3 버킷 또는 리포지토리는 부트스트랩 중에 생성됩니다.) 그래야만 스택에 정의된 리소스가 배포됩니다.

이 섹션에서는 프레임워크에서 사용할 수 있는 저수준 API에 대해 설명합니다.

자산 유형

는 다음과 같은 유형의 자산을 AWS CDK 지원합니다.

아마존 S3 자산

Amazon S3에 AWS CDK 업로드하는 로컬 파일 및 디렉터리입니다.

Docker 이미지

다음은 Amazon ECR에 AWS CDK 업로드한 도커 이미지입니다.

이러한 자산 유형은 다음 섹션에 설명되어 있습니다.

아마존 S3 자산

로컬 파일 및 디렉터리를 자산으로 정의하고, AWS-s3-assets 모듈을 통해 이를 AWS CDK 패키지하고 Amazon S3에 업로드할 수 있습니다.

다음 예제는 로컬 디렉터리 자산과 파일 자산을 정의합니다.

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")),
})
```

대부분의 경우 `aws-s3-assets` 모듈에서 API를 직접 사용할 필요는 없습니다. 와 같은 `aws-lambda` 자산을 지원하는 모듈에는 자산을 사용할 수 있는 편리한 메서드가 있습니다. Lambda 함수의 경우 fromAsset () 정적 메서드를 사용하면 로컬 파일 시스템에서 디렉토리 또는.zip 파일을 지정할 수 있습니다.

Lambda 함수 예제

일반적인 사용 사례는 핸들러 코드를 Amazon S3 자산으로 사용하여 Lambda 함수를 생성하는 것입니다.

다음 예제에서는 Amazon S3 자산을 사용하여 로컬 디렉터리에 `handler` Python 핸들러를 정의합니다. 또한 로컬 디렉토리 자산을 속성으로 사용하여 Lambda 함수를 생성합니다. `code` 다음은 핸들러의 Python 코드입니다.

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

기본 AWS CDK 앱의 코드는 다음과 같아야 합니다.

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
}
```

이 Function 메서드는 자산을 사용하여 디렉터리의 내용을 번들로 묶고 함수 코드에 사용합니다.

> **Tip**
>
> Java .jar 파일은 확장자가 다른 ZIP 파일입니다. Amazon S3에 있는 그대로 업로드되지만 Lambda 함수로 배포하면 포함된 파일이 추출되므로 원하지 않을 수 있습니다. 이를 방지하려면 .jar 파일을 디렉터리에 배치하고 해당 디렉터리를 자산으로 지정하십시오.

배포 시간 속성 예제

Amazon S3 자산 유형은 또한 라이브러리 및 앱에서 참조할 수 있는 배포 시간 속성을 노출합니다. AWS CDK AWS CDK CLI 명령은 자산 속성을 매개변수로 cdk synth AWS CloudFormation 표시합니다.

다음 예제는 배포 시간 속성을 사용하여 이미지 자산의 위치를 Lambda 함수에 환경 변수로 전달합니다. (파일 종류는 중요하지 않습니다. 여기에 사용된 PNG 이미지는 예시일 뿐입니다.)

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(),
  },
})
```

권한

AWS-s3-assets 모듈, IAM 역할, 사용자 또는 그룹을 통해 직접 Amazon S3 자산을 사용하고 런타임에 자산을 읽어야 하는 경우 Asset.GrantRead 메서드를 통해 해당 자산에 IAM 권한을 부여하십시오.

다음 예제는 IAM 그룹에 파일 자산에 대한 읽기 권한을 부여합니다.

TypeScript

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

const asset = new Asset(this, 'MyFile', {
  path: path.join(__dirname, 'my-image.png')
});

const group = new iam.Group(this, 'MyUserGroup');
asset.grantRead(group);
```

JavaScript

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

const asset = new Asset(this, 'MyFile', {
  path: path.join(__dirname, 'my-image.png')
});

const group = new iam.Group(this, 'MyUserGroup');
asset.grantRead(group);
```

Python

```
from aws_cdk.aws_s3_assets import Asset
import aws_cdk.aws_iam as iam
```

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

        asset = Asset(self, "MyFile",
            path=os.path.join(dirname, "my-image.png"))

        group = iam.Group(self, "MyUserGroup")
        asset.grant_read(group)
```

Java

```
import java.io.File;

import software.amazon.awscdk.Stack;
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.services.iam.Group;
import software.amazon.awscdk.services.s3.assets.Asset;

public class GrantStack extends Stack {
    public GrantStack(final App scope, final String id, final StackProps props) {
        super(scope, id, props);

        File startDir = new File(System.getProperty("user.dir"));

        Asset asset = Asset.Builder.create(this, "SampleAsset")
                .path(new File(startDir, "images/my-image.png").toString()).build();

        Group group = new Group(this, "MyUserGroup");
        asset.grantRead(group);   }
}
```

C#

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

var asset = new Asset(this, "MyFile", new AssetProps {
    Path = Path.Combine(Path.Combine(Directory.GetCurrentDirectory(), @"images\my-
image.png"))
});
```

```
var group = new Group(this, "MyUserGroup");
asset.GrantRead(group);
```

Go

```
import (
  "os"
  "path"

  "github.com/aws/aws-cdk-go/awscdk/v2"
  "github.com/aws/aws-cdk-go/awscdk/v2/awsiam"
  "github.com/aws/aws-cdk-go/awscdk/v2/awss3assets"
)

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

asset := awss3assets.NewAsset(stack, jsii.String("MyFile"), &awss3assets.AssetProps{
  Path: jsii.String(path.Join(dirName, "my-image.png")),
})

group := awsiam.NewGroup(stack, jsii.String("MyUserGroup"), &awsiam.GroupProps{})

asset.GrantRead(group)
```

Docker 이미지 자산

는 모듈을 통해 로컬 Docker 이미지를 자산으로 번들링할 수 있도록 AWS CDK 지원합니다. aws-ecr-assets

다음 예제는 로컬에서 빌드되어 Amazon ECR로 푸시되는 Docker 이미지를 정의합니다. 이미지는 로컬 Docker 컨텍스트 디렉터리 (Dockerfile 포함) 에서 빌드되고 AWS CDK CLI 또는 앱의 CI/CD 파이프라인을 통해 Amazon ECR에 업로드됩니다. 이미지는 앱에서 자연스럽게 참조될 수 있습니다. AWS CDK

TypeScript

```
import { DockerImageAsset } from 'aws-cdk-lib/aws-ecr-assets';
```

```
const asset = new DockerImageAsset(this, 'MyBuildImage', {
  directory: path.join(__dirname, 'my-image')
});
```

JavaScript

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

const asset = new DockerImageAsset(this, 'MyBuildImage', {
  directory: path.join(__dirname, 'my-image')
});
```

Python

```
from aws_cdk.aws_ecr_assets import DockerImageAsset

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

asset = DockerImageAsset(self, 'MyBuildImage',
    directory=os.path.join(dirname, 'my-image'))
```

Java

```
import software.amazon.awscdk.services.ecr.assets.DockerImageAsset;

File startDir = new File(System.getProperty("user.dir"));

DockerImageAsset asset = DockerImageAsset.Builder.create(this, "MyBuildImage")
            .directory(new File(startDir, "my-image").toString()).build();
```

C#

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

var asset = new DockerImageAsset(this, "MyBuildImage", new DockerImageAssetProps
{
    Directory = Path.Combine(Directory.GetCurrentDirectory(), "my-image")
});
```

Go

```
import (
  "os"
  "path"

  "github.com/aws/aws-cdk-go/awscdk/v2"
  "github.com/aws/aws-cdk-go/awscdk/v2/awsecrassets"
)

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

asset := awsecrassets.NewDockerImageAsset(stack, jsii.String("MyBuildImage"),
 &awsecrassets.DockerImageAssetProps{
  Directory: jsii.String(path.Join(dirName, "my-image")),
})
```

`my-image`디렉터리에는 Dockerfile이 포함되어야 합니다. AWS CDK CLI는 에서 `my-image` Docker 이미지를 빌드하여 Amazon ECR 리포지토리로 푸시하고 리포지토리 이름을 스택의 파라미터로 지정합니다. AWS CloudFormation Docker 이미지 자산 유형은 라이브러리와 앱에서 참조할 수 있는 배포 시간 속성을 노출합니다. AWS CDK AWS CDK CLI 명령은 자산 속성을 매개변수로 cdk synth AWS CloudFormation 표시합니다.

Amazon ECS 작업 정의 예제

일반적인 사용 사례는 Docker 컨테이너를 TaskDefinition실행하기 위해 Amazon ECS를 생성하는 것입니다. 다음 예제는 로컬에서 AWS CDK 빌드하여 Amazon ECR로 푸시하는 Docker 이미지 자산의 위치를 지정합니다.

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),
})
```

배포 시간 속성 예제

다음 예제는 배포 시간 속성을 `repository` 사용하고 시작 유형으로 Amazon ECS 작업 정의를 생성하는 `imageUri` 방법을 보여줍니다. AWS Fargate Amazon ECR 리포지토리 조회에는 URI가 아니라 이미지의 태그가 필요하므로 자산의 URI 끝에서 이미지를 잘라냅니다.

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()),
})
```

빌드 인수 예제

AWS CDK CLI가 배포 중에 이미지를 빌드할 때 `buildArgs` (Python:`build_args`) 속성 옵션을 통해 Docker 빌드 단계에 사용자 지정된 빌드 인수를 제공할 수 있습니다.

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"),
  },
})
```

권한

Docker 이미지 자산을 지원하는 모듈 (예: aws-ecs) 을 사용하는 경우 자산을 직접 또는 통해 사용할 때 에서 권한을 AWS CDK 관리합니다. ContainerImage fromEcrRepository(Python:`from_ecr_repository`). Docker 이미지 자산을 직접 사용하는 경우 소비 주체에게 이미지를 가져올 권한이 있는지 확인하세요.

대부분의 경우 Asset.Repository.GrantPull 메서드를 사용해야 합니다 (Python:. `grant_pull` 이렇게 하면 보안 주체의 IAM 정책이 수정되어 이 리포지토리에서 이미지를 가져올 수 있게 됩니다. 이미지를 가져오는 보안 주체가 동일한 계정에 속하지 않거나 계정에서 역할을 맡지 않는 AWS 서비스인 경우 (예: AWS CodeBuild) 보안 주체의 정책이 아닌 리소스 정책에 pull 권한을 부여해야 합니다. asset.repository를 사용하세요. addToResource적절한 기본 권한을 부여하는 정책 메서드 (Python:`add_to_resource_policy`).

AWS CloudFormation 리소스 메타데이터

> **Note**
>
> 이 섹션은 구성 작성자에게만 해당됩니다. 특정 상황에서는 도구가 특정 CFN 리소스가 로컬 자산을 사용하고 있음을 알아야 합니다. 예를 들어, AWS SAM CLI를 사용하여 디버깅 목적으로 Lambda 함수를 로컬에서 호출할 수 있습니다. 세부 정보는 the section called “AWS SAM 통합”를 참조하세요.

이러한 사용 사례를 활성화하기 위해 외부 도구는 리소스의 메타데이터 항목 세트를 참조합니다. AWS CloudFormation

- `aws:asset:path`— 자산의 로컬 경로를 가리킵니다.
- `aws:asset:property`— 자산이 사용되는 리소스 속성의 이름.

도구는 이 두 메타데이터 항목을 사용하여 특정 리소스에서 자산을 사용하고 있는지 식별하고 고급 로컬 경험을 제공할 수 있습니다.

이러한 메타데이터 항목을 리소스에 추가하려면 `asset.addResourceMetadata` (Python:`add_resource_metadata`) 메서드를 사용합니다.

권한

AWS 구성 라이브러리는 널리 구현되는 몇 가지 일반적인 관용구를 사용하여 액세스 및 권한을 관리합니다. IAM 모듈은 이러한 관용구를 사용하는 데 필요한 도구를 제공합니다.

AWS CDK 변경 사항을 AWS CloudFormation 배포하는 데 사용합니다. 모든 배포에는 배포를 시작하는 행위자 (개발자 또는 자동화된 시스템) 가 참여합니다. AWS CloudFormation 이 과정에서 행위자는 하나 이상의 IAM ID (사용자 또는 역할) 를 수임하고 선택적으로 역할을 전달합니다. AWS CloudFormation

를 사용하여 사용자로 AWS IAM Identity Center 인증하는 경우 Single Sign-On 공급자는 사전 정의된 IAM 역할로 사용할 수 있는 권한을 부여하는 단기 세션 자격 증명을 제공합니다. IAM Identity Center 인증에서 AWS 자격 증명을 AWS CDK 얻는 방법을 알아보려면 SDK 및 도구 참조 안내서의 IAM Identity Center 인증 이해를 참조하십시오.AWS

보안 주체

IAM 보안 주체는 API를 호출할 수 있는 사용자, 서비스 또는 애플리케이션을 대표하는 인증된 AWS 엔티티입니다. AWS AWS 구성 라이브러리는 여러 가지 유연한 방법으로 주체를 지정하여 리소스에 대한 액세스 권한을 부여할 수 있도록 지원합니다. AWS

보안 컨텍스트에서 “주체”라는 용어는 특히 사용자와 같은 인증된 개체를 가리킵니다. 그룹 및 역할과 같은 개체는 사용자 (및 기타 인증된 개체) 를 나타내는 것이 아니라 권한 부여를 위해 사용자를 간접적으로 식별합니다.

예를 들어, IAM 그룹을 생성하는 경우 그룹 (및 그룹 구성원) 에게 Amazon RDS 테이블에 대한 쓰기 액세스 권한을 부여할 수 있습니다. 하지만 그룹 자체는 단일 엔티티를 나타내지 않기 때문에 보안 주체가 아니며 그룹에 로그인할 수도 없습니다.

CDK의 IAM 라이브러리에서는 주체를 직접 또는 간접적으로 식별하는 클래스가 `IPrincipal`인터페이스를 구현하므로 액세스 정책에서 이러한 객체를 서로 바꿔서 사용할 수 있습니다. 하지만 모든 보안 측면에서 보안 주체가 되는 것은 아닙니다. 이러한 객체에는 다음이 포함됩니다.

1. `Role`,`User`, 및 같은 IAM 리소스 `Group`
2. 서비스 주체 () `new iam.ServicePrincipal('service.amazonaws.com')`
3. 연합 주체 () `new iam.FederatedPrincipal('cognito-identity.amazonaws.com')`
4. 계정 보안 주체 (`new iam.AccountPrincipal('0123456789012')`)
5. 표준 사용자 주체 () `new iam.CanonicalUserPrincipal('79a59d[...]7ef2be')`
6. AWS Organizations `new iam.OrganizationPrincipal('org-id')`주도자 ()

7. 임의 ARN 보안 주체 () `new iam.ArnPrincipal(res.arn)`
8. 여러 보안 주체를 신뢰할 수 있는지 여부 `iam.CompositePrincipal(principal1, principal2, ...)`

권한 부여

Amazon S3 버킷 또는 Amazon DynamoDB 테이블과 같이 액세스 가능한 리소스를 나타내는 모든 구조에는 다른 엔티티에 대한 액세스 권한을 부여하는 메서드가 있습니다. 이러한 모든 메서드는 grant로 시작하는 이름을 갖습니다.

예를 들어 Amazon S3 버킷에는 엔티티에서 버킷으로의 읽기 `grantRead` 및 읽기/쓰기 액세스를 각각 활성화하는 메서드와 `grantReadWrite` (Python:`grant_read`,`grant_read_write`) 가 있습니다. 엔티티는 이러한 작업을 수행하는 데 필요한 Amazon S3 IAM 권한을 정확히 알 필요가 없습니다.

권한 부여 메서드의 첫 번째 인수는 항상 iGrantable 유형입니다. 이 인터페이스는 권한을 부여받을 수 있는 엔티티를 나타냅니다. 즉, IAM 객체 `RoleUser`, 및 `Group` 와 같은 역할이 있는 리소스를 나타냅니다.

다른 엔티티에도 권한을 부여할 수 있습니다. 예를 들어, 이 주제 후반부에서는 Amazon S3 버킷에 대한 CodeBuild 프로젝트 액세스 권한을 부여하는 방법을 보여줍니다. 일반적으로 관련 역할은 액세스 권한이 부여된 개체의 `role` 속성을 통해 부여됩니다.

실행 역할을 사용하는 리소스 (예:) 도 `IGrantable` 구현되므로 역할에 대한 액세스 권한을 부여하는 대신 직접 액세스 권한을 부여할 수 있습니다. `lambda.Function` 예를 들어 Amazon S3 `bucket` 버킷이고 Lambda 함수인 경우 다음 코드는 함수에 버킷에 대한 읽기 액세스 권한을 부여합니다. `function`

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

스택을 배포하는 동안 권한을 적용해야 하는 경우가 있습니다. 이러한 경우 중 하나는 AWS CloudFormation 사용자 지정 리소스에 다른 리소스에 대한 액세스 권한을 부여하는 경우입니다. 사용자 지정 리소스는 배포 중에 호출되므로 배포 시 지정된 권한이 있어야 합니다.

또 다른 경우는 전달한 역할에 올바른 정책이 적용되었는지 서비스가 확인하는 경우입니다. (정책을 설정하는 것을 잊지 않도록 하기 위해 이 작업을 수행하는 AWS 서비스가 많습니다.) 이러한 경우 권한이 너무 늦게 적용되면 배포가 실패할 수 있습니다.

다른 리소스가 생성되기 전에 권한 부여의 권한을 강제로 적용하려면 다음과 같이 권한 부여 자체에 종속성을 추가할 수 있습니다. 권한 부여 메서드의 반환 값은 일반적으로 삭제되지만 실제로 모든 권한 부여 메서드는 객체를 반환합니다. `iam.Grant`

TypeScript

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

JavaScript

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

Python

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

Java

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

C#

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

역할

IAM 패키지에는 IAM 역할을 나타내는 `Role` 구조가 포함되어 있습니다. 다음 코드는 Amazon EC2 서비스를 신뢰하는 새 역할을 생성합니다.

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
});
```

역할의 addToPolicy 메서드 (Python:add_to_policy) 를 호출하고 추가할 규칙을 PolicyStatement 정의하는 a를 전달하여 역할에 권한을 추가할 수 있습니다. 명령문은 역할의 기본 정책에 추가되며, 없는 경우 새 정책이 생성됩니다.

다음 예제는 승인된 서비스를 조건으로 작업 ec2:SomeAction 및 리소스 bucket 및 s3:AnotherAction otherRole (Python:other_role) 의 역할에 Deny 정책 설명을 추가합니다 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"
        }
    }
}));
```

위 예제에서는 (addToPolicyPython:add_to_policy) PolicyStatement 호출을 사용하여 새 인라인을 만들었습니다. 기존 정책 설명문이나 수정한 설명을 전달할 수도 있습니다. PolicyStatement객체에는 주도자, 리소스, 조건 및 작업을 추가할 수 있는 다양한 메서드가 있습니다.

제대로 작동하는 데 역할이 필요한 구문을 사용하는 경우 다음 중 하나를 수행할 수 있습니다.

- 구성 객체를 인스턴스화할 때 기존 역할을 전달하세요.
- 해당 구문을 통해 적절한 서비스 주체를 신뢰하여 새 역할을 자동으로 생성하도록 하십시오. 다음 예제에서는 이러한 구조를 사용합니다. 바로 CodeBuild 프로젝트입니다.

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
});
```

객체가 만들어지면 역할 (전달된 역할이든 구문을 통해 생성되는 기본 역할이든) 을 속성으로 사용할 수 `role` 있습니다. 하지만 외부 리소스에서는 이 속성을 사용할 수 없습니다. 따라서 이러한 구문에는 `addToRolePolicy` (Python:`add_to_role_policy`) 메서드가 있습니다.

이 메서드는 구문이 외부 리소스인 경우 아무 작업도 수행하지 않고 그렇지 않으면 `role` 속성의 `addToPolicy` (Python:`add_to_policy`) 메서드를 호출합니다. 이렇게 하면 정의되지 않은 케이스를 명시적으로 처리하는 수고를 덜 수 있습니다.

다음 예제는 다음을 보여줍니다.

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
}));
```

리소스 정책

Amazon S3 버킷 및 IAM 역할과 같은 일부 리소스에도 리소스 정책이 있습니다. AWS이러한 구문에는 a `PolicyStatement` 를 인수로 취하는 `addToResourcePolicy` 메서드 (Python:`add_to_resource_policy`) 가 있습니다. 리소스 정책에 추가되는 모든 정책 설명은 하나 이상의 보안 주체를 지정해야 합니다.

다음 예제에서 Amazon S3 버킷은 역할에 자신에게 `s3:SomeAction` 권한을 `bucket` 부여합니다.

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 }
}));
```

외부 IAM 객체 사용

앱 외부에서 IAM 사용자, 보안 주체, 그룹 또는 역할을 정의한 경우 해당 IAM 객체를 AWS CDK 앱에서 사용할 수 있습니다. AWS CDK 이렇게 하려면 ARN 또는 이름을 사용하여 참조를 생성하십시오. (사용

자, 그룹 및 역할에는 이름을 사용하십시오.) 그런 다음 반환된 참조를 사용하여 권한을 부여하거나 앞에서 설명한 대로 정책 설명을 구성할 수 있습니다.

- 사용자의 경우 `User.fromUserArn()` 또는 으로 `User.fromUserName()` 전화하십시오. `User.fromUserAttributes()`도 사용할 수 있지만 현재는 과 동일한 기능을 제공합니다`User.fromUserArn()`.
- 주도자의 경우 객체를 인스턴스화하십시오. `ArnPrincipal`
- 그룹의 경우 또는 를 호출하십시오. `Group.fromGroupArn()` `Group.fromGroupName()`
- 역할을 원하시면 `Role.fromRoleArn()` 또는 전화로 `Role.fromRoleName()` 문의하세요.

정책 (관리형 정책 포함) 은 다음 방법을 사용하여 비슷한 방식으로 사용할 수 있습니다. IAM 정책이 필요한 모든 곳에서 이러한 객체에 대한 참조를 사용할 수 있습니다.

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

> **Note**
>
> 외부 AWS 리소스에 대한 모든 참조와 마찬가지로 CDK 앱에서는 외부 IAM 객체를 수정할 수 없습니다.

런타임 컨텍스트

컨텍스트 값은 앱, 스택 또는 구성과 연결할 수 있는 키-값 페어입니다. 파일 (일반적으로 프로젝트 디렉토리 또는 프로젝트 디렉터리) `cdk.json` 또는 `cdk.context.json` 명령줄에서 앱에 제공할 수 있습니다.

CDK 툴킷은 합성 중에 컨텍스트를 사용하여 AWS 계정에서 검색된 값을 캐시합니다. 값에는 계정의 가용 영역 또는 현재 Amazon EC2 인스턴스에 사용할 수 있는 Amazon 머신 이미지 (AMI) ID가 포함됩니다. 이러한 값은 AWS 계정에서 제공되므로 CDK 애플리케이션 실행 간에 변경될 수 있습니다. 따라서 의도하지 않은 변경의 원인이 될 수 있습니다. CDK 툴킷의 캐싱 동작은 새 값을 수락하기로 결정할 때까지 CDK 앱에서 이러한 값을 “고정”시킵니다.

컨텍스트 캐싱이 없는 다음과 같은 시나리오를 상상해 보세요. Amazon EC2 인스턴스의 AMI로 “최신 Amazon Linux”를 지정하고 이 AMI의 새 버전이 출시되었다고 가정해 보겠습니다. 그러면 다음 번에 CDK 스택을 배포할 때 이미 배포된 인스턴스가 오래된 (“잘못된”) AMI를 사용하게 되므로 업그레이드해야 합니다. 업그레이드하면 기존 인스턴스가 모두 새 인스턴스로 교체되는데, 이는 예상치 못한 일이고 원치 않는 일이었을 수 있습니다.

대신 CDK는 계정의 사용 가능한 AMI를 프로젝트 `cdk.context.json` 파일에 기록하고 저장된 값을 향후 합성 작업에 사용합니다. 이렇게 하면 AMI 목록이 더 이상 잠재적 변경의 원인이 되지 않습니다. 또한 스택이 항상 동일한 템플릿에 합성되도록 할 수 있습니다. AWS CloudFormation

모든 컨텍스트 값이 사용자 환경의 캐시된 값인 것은 아닙니다. AWS the section called “기능 플래그”컨텍스트 값이기도 합니다. 앱이나 구문에 사용할 고유한 컨텍스트 값을 만들 수도 있습니다.

컨텍스트 키는 문자열입니다. 값은 숫자, 문자열, 배열, 객체 등 JSON에서 지원하는 모든 유형일 수 있습니다.

> **Tip**
>
> 구문이 고유한 컨텍스트 값을 생성하는 경우 라이브러리의 패키지 이름을 키에 통합하여 다른 패키지의 컨텍스트 값과 충돌하지 않도록 하세요.

많은 컨텍스트 값이 특정 AWS 환경과 연관되어 있으며, 특정 CDK 앱을 둘 이상의 환경에 배포할 수 있습니다. 이러한 값의 키에는 AWS 계정과 지역이 포함되므로 서로 다른 환경의 값이 충돌하지 않습니다.

다음 컨텍스트 키는 계정과 지역을 AWS CDK포함하여 에서 사용하는 형식을 보여줍니다.

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

> **Important**
>
> 캐시된 컨텍스트 값은 사용자가 작성할 수 있는 구문을 포함하여 AWS CDK 와 해당 구문을 통해 관리됩니다. 파일을 수동으로 편집하여 캐시된 컨텍스트 값을 추가하거나 변경하지 마십시오. 하지만 `cdk.context.json` 가끔 어떤 값이 캐시되는지 검토해 보는 것이 유용할 수 있습니다. 캐시된 값을 나타내지 않는 컨텍스트 값은 의 `context` 키 아래에 저장해야 합니다. `cdk.json` 이렇게 하면 캐시된 값을 지워도 삭제되지 않습니다.

컨텍스트 값의 소스

컨텍스트 값은 다음과 같은 6가지 방법으로 AWS CDK 앱에 제공할 수 있습니다.

- 현재 AWS 계정에서 자동으로.
- cdk명령에 대한 --context 옵션을 통해 (이 값은 항상 문자열입니다.)
- 프로젝트 `cdk.context.json` 파일에서
- 프로젝트 `cdk.json` 파일의 `context` 키에.
- `~/.cdk.json`파일 `context` 키에.
- AWS CDK 앱에서 `construct.node.setContext()` 메서드를 사용합니다.

프로젝트 `cdk.context.json` 파일은 AWS 계정에서 검색된 컨텍스트 값을 AWS CDK 캐시하는 곳입니다. 이렇게 하면 예를 들어 새 가용 영역이 도입될 때 배포에 예상치 못한 변경 사항이 발생하는 것을 방지할 수 있습니다. 는 나열된 다른 파일에는 컨텍스트 데이터를 쓰지 AWS CDK 않습니다.

> **Important**
>
> 이는 애플리케이션 상태의 일부이므로 나머지 앱 소스 코드와 함께 소스 제어에 `cdk.context.json` 커밋되어야 하기 때문입니다. `cdk.json` 그렇지 않으면 다른 환경 (예: CI 파이프라인) 에 배포하면 일관되지 않은 결과가 발생할 수 있습니다.

컨텍스트 값은 해당 값을 생성한 구문으로 범위가 지정되며, 자식 구문에는 표시되지만 부모나 형제 구조에는 표시되지 않습니다. AWS CDK 툴킷 (cdk명령) 으로 설정된 컨텍스트 값은 파일 또는 옵션에서 자동으로 설정할 수 있습니다. --context 이러한 소스의 컨텍스트 값은 구문에 암시적으로 설정됩니다. App 따라서 앱 내 모든 스택의 모든 구조에서 볼 수 있습니다.

앱은 `construct.node.tryGetContext` 메서드를 사용하여 컨텍스트 값을 읽을 수 있습니다. 요청된 항목을 현재 구조나 그 상위 구조에서 찾을 수 없는 경우 결과는 다음과 같습니다`undefined`. (또는 None Python에서와 같이 사용자 언어와 동일한 결과가 나올 수도 있습니다.)

컨텍스트 메서드

는 AWS CDK 앱이 환경에서 컨텍스트 정보를 얻을 수 있도록 하는 여러 컨텍스트 메서드를 AWS CDK 지원합니다. AWS [예를 들어 Stack.availabilityZones 메서드를 사용하여 특정 AWS 계정 및 지역에서 사용할 수 있는 가용 영역 목록을 가져올 수 있습니다.]

컨텍스트 메서드는 다음과 같습니다.

HostedZone.fromLookup

계정의 호스팅 영역을 가져옵니다.

스택. 가용 영역

지원되는 가용 영역을 가져옵니다.

StringParameter.valueFromLookup

현재 지역의 Amazon EC2 Systems Manager 파라미터 스토어에서 값을 가져옵니다.

VPC. FromLookup

계정의 기존 Amazon 가상 사설 클라우드를 가져옵니다.

LookupMachineImage

Amazon Virtual Private Cloud에서 NAT 인스턴스와 함께 사용할 머신 이미지를 찾습니다.

필수 컨텍스트 값을 사용할 수 없는 경우 AWS CDK 앱은 CDK 툴킷에 컨텍스트 정보가 누락되었음을 알립니다. 그런 다음 CLI는 현재 AWS 계정에서 정보를 쿼리하고 결과 컨텍스트 정보를 파일에 저장합니다. `cdk.context.json` 그런 다음 컨텍스트 값을 사용하여 AWS CDK 앱을 다시 실행합니다.

컨텍스트 보기 및 관리

cdk context명령을 사용하여 `cdk.context.json` 파일의 정보를 보고 관리할 수 있습니다. 이 정보를 보려면 옵션 없이 cdk context 명령을 사용하십시오. 출력은 다음과 같아야 합니다.

```
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.
```

컨텍스트 값을 제거하려면 값에 해당하는 키 또는 번호를 지정하여 실행합니다cdk context --reset. 다음 예제에서는 이전 예제의 두 번째 키에 해당하는 값을 제거합니다. 이 값은 유럽 (아일랜드) 지역의 가용 영역 목록을 나타냅니다.

```
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.
```

따라서 Amazon Linux AMI를 최신 버전으로 업데이트하려면 위 예제를 사용하여 컨텍스트 값의 제어된 업데이트를 수행하고 이를 재설정하십시오. 그런 다음 앱을 다시 합성하고 배포하십시오.

```
cdk synth
```

앱에 저장된 컨텍스트 값을 모두 지우려면 다음과 같이 cdk context --clear 실행하세요.

```
cdk context --clear
```

에 저장된 컨텍스트 값만 재설정하거나 지울 `cdk.context.json` 수 있습니다. 다른 컨텍스트 AWS CDK 값에는 영향을 주지 않습니다. 따라서 이러한 명령을 사용하여 컨텍스트 값이 재설정되지 않도록 하려면 값을 에 복사할 수 `cdk.json` 있습니다.

AWS CDK 툴킷 `--context` 플래그

`--context`(`-c`간단히 말해서) 옵션을 사용하면 합성 또는 배포 중에 런타임 컨텍스트 값을 CDK 앱에 전달할 수 있습니다.

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

컨텍스트 값을 여러 개 지정하려면 --context 옵션을 여러 번 반복하여 매번 키-값 쌍을 하나씩 제공하십시오.

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

여러 스택을 합성하는 경우 지정된 컨텍스트 값이 모든 스택에 전달됩니다. 개별 스택에 서로 다른 컨텍스트 값을 제공하려면 값에 다른 키를 사용하거나 or 명령을 여러 개 사용하십시오. cdk synth cdk deploy

명령줄에서 전달되는 컨텍스트 값은 항상 문자열입니다. 값이 일반적으로 다른 유형인 경우 값을 변환하거나 파싱할 코드를 준비해야 합니다. 문자열이 아닌 컨텍스트 값이 다른 방식 (예: in) 으로 제공될 수 있습니다. `cdk.context.json` 이러한 종류의 값이 예상대로 작동하는지 확인하려면 값을 변환하기 전에 값이 문자열인지 확인하십시오.

예

다음은 컨텍스트를 사용하여 기존 Amazon VPC를 사용하는 AWS CDK 예제입니다.

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import { Construct } from 'constructs';

export class ExistsVpcStack extends cdk.Stack {

  constructor(scope: Construct, id: string, props?: cdk.StackProps) {

    super(scope, id, props);

    const vpcid = this.node.tryGetContext('vpcid');
    const vpc = ec2.Vpc.fromLookup(this, 'VPC', {
      vpcId: vpcid,
    });

    const pubsubnets = vpc.selectSubnets({subnetType: ec2.SubnetType.PUBLIC});

    new cdk.CfnOutput(this, 'publicsubnets', {
      value: pubsubnets.subnetIds.toString(),
    });
  }
}
```

JavaScript

```
const cdk = require('aws-cdk-lib');
const ec2 = require('aws-cdk-lib/aws-ec2');

class ExistsVpcStack extends cdk.Stack {

  constructor(scope, id, props) {
```

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

    const vpcid = this.node.tryGetContext('vpcid');
    const vpc = ec2.Vpc.fromLookup(this, 'VPC', {
      vpcId: vpcid
    });

    const pubsubnets = vpc.selectSubnets({subnetType: ec2.SubnetType.PUBLIC});

    new cdk.CfnOutput(this, 'publicsubnets', {
      value: pubsubnets.subnetIds.toString()
    });
  }
}

module.exports = { ExistsVpcStack }
```

Python

```
import aws_cdk as cdk
import aws_cdk.aws_ec2 as ec2
from constructs import Construct

class ExistsVpcStack(cdk.Stack):

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

        super().__init__(scope, id, **kwargs)

        vpcid = self.node.try_get_context("vpcid")
        vpc = ec2.Vpc.from_lookup(self, "VPC", vpc_id=vpcid)

        pubsubnets = vpc.select_subnets(subnetType=ec2.SubnetType.PUBLIC)

        cdk.CfnOutput(self, "publicsubnets",
            value=pubsubnets.subnet_ids.to_string())
```

Java

```
import software.amazon.awscdk.CfnOutput;

import software.amazon.awscdk.services.ec2.Vpc;
import software.amazon.awscdk.services.ec2.VpcLookupOptions;
```

```
import software.amazon.awscdk.services.ec2.SelectedSubnets;
import software.amazon.awscdk.services.ec2.SubnetSelection;
import software.amazon.awscdk.services.ec2.SubnetType;
import software.constructs.Construct;

public class ExistsVpcStack extends Stack {
    public ExistsVpcStack(Construct context, String id) {
        this(context, id, null);
    }

    public ExistsVpcStack(Construct context, String id, StackProps props) {
        super(context, id, props);

        String vpcId = (String)this.getNode().tryGetContext("vpcid");
        Vpc vpc = (Vpc)Vpc.fromLookup(this, "VPC", VpcLookupOptions.builder()
                .vpcId(vpcId).build());

        SelectedSubnets pubSubNets = vpc.selectSubnets(SubnetSelection.builder()
                .subnetType(SubnetType.PUBLIC).build());

        CfnOutput.Builder.create(this, "publicsubnets")
                .value(pubSubNets.getSubnetIds().toString()).build();

    }
}
```

C#

```
using Amazon.CDK;
using Amazon.CDK.AWS.EC2;
using Constructs;

class ExistsVpcStack : Stack
{
    public ExistsVpcStack(Construct scope, string id, StackProps props) :
 base(scope, id, props)
    {
        var vpcId = (string)this.Node.TryGetContext("vpcid");
        var vpc = Vpc.FromLookup(this, "VPC", new VpcLookupOptions
        {
            VpcId = vpcId
        });
```

```
        SelectedSubnets pubSubNets = vpc.SelectSubnets([new SubnetSelection
        {
            SubnetType = SubnetType.PUBLIC
        }]);

        new CfnOutput(this, "publicsubnets", new CfnOutputProps {
            Value = pubSubNets.SubnetIds.ToString()
        });
    }
}
```

를 cdk diff 사용하여 명령줄에서 컨텍스트 값을 전달했을 때의 효과를 확인할 수 있습니다.

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

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

결과 컨텍스트 값은 다음과 같이 볼 수 있습니다.

```
cdk context -j
```

```
{
  "vpc-provider:account=123456789012:filter.vpc-id=vpc-0cb9c31031d0d3e22:region=us-
east-1": {
    "vpcId": "vpc-0cb9c31031d0d3e22",
    "availabilityZones": [
      "us-east-1a",
      "us-east-1b"
    ],
    "privateSubnetIds": [
      "subnet-03ecfc033225be285",
      "subnet-0cded5da53180ebfa"
    ],
    "privateSubnetNames": [
      "Private"
    ],
    "privateSubnetRouteTableIds": [
```

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

기능 플래그

AWS CDK 는 기능 플래그를 사용하여 릴리스에서 잠재적으로 문제가 될 수 있는 동작을 활성화합니다. 플래그는 `cdk.json` (또는`~/.cdk.json`) 에 the section called “컨텍스트” 값으로 저장됩니다. cdk context --reset또는 cdk context --clear 명령으로 제거되지는 않습니다.

기능 플래그는 기본적으로 비활성화됩니다. 플래그를 지정하지 않은 기존 프로젝트는 이후 AWS CDK 릴리스에서도 이전과 마찬가지로 계속 작동합니다. cdk init포함 플래그를 사용하여 만든 새 프로젝트는 프로젝트를 만든 릴리스에서 사용할 수 있는 모든 기능을 사용할 수 있게 합니다. 이전 동작에서 원하는 플래그를 `cdk.json` 비활성화하려면 편집하세요. 업그레이드 후 플래그를 추가하여 새 동작을 활성화할 수도 있습니다. AWS CDK

모든 현재 기능 플래그 목록은 의 AWS CDK GitHub 저장소에서 찾을 수 있습니다. `FEATURE_FLAGS.md` 해당 `CHANGELOG` 릴리스에 추가된 새 기능 플래그에 대한 설명은 해당 릴리스의 를 참조하십시오.

v1 동작으로 되돌리기

CDK v2에서는 일부 기능 플래그의 기본값이 v1과 관련하여 변경되었습니다. 이를 다시 설정하여 특정 v1 동작으로 `false` 되돌릴 수 있습니다. AWS CDK `cdk diff`명령을 사용하여 합성된 템플릿의 변경 사항을 검사하여 이러한 플래그가 필요한지 확인하세요.

`@aws-cdk/core:newStyleStackSynthesis`

잘 알려진 이름을 가진 부트스트랩 리소스를 가정하는 새로운 스택 합성 방법을 사용하세요. 최신 부트스트래핑이 필요하지만 CDK Pipeline을 통한 CI/CD 및 계정 간 배포가 즉시 가능합니다.

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

애플리케이션에서 여러 Amazon API Gateway API 키를 사용하고 이를 사용량 계획에 연결하는 경우

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

애플리케이션에서 Amazon RDS 데이터베이스 인스턴스 또는 데이터베이스 클러스터를 사용하고 이에 대한 식별자를 명시적으로 지정하는 경우

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

애플리케이션이 배포와 함께 TLS_V1_2_2019 보안 정책을 사용하는 경우 Amazon CloudFront CDK v2는 기본적으로 보안 정책 TLSv1.2_2021을 사용합니다.

`@aws-cdk/core:stackRelativeExports`

애플리케이션에서 여러 스택을 사용하고 한 스택의 리소스를 다른 스택에서 참조하는 경우 내보내기를 구성하는 데 절대 경로를 사용할지 상대 경로를 사용할지가 결정됩니다. AWS CloudFormation

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

로 `false` 설정된 경우 CDK는 Lambda 함수의 변경 여부를 감지할 때 메타데이터를 포함합니다. 중복 버전은 허용되지 않으므로 메타데이터만 변경된 경우 배포가 실패할 수 있습니다. 애플리케이션의 모든 Lambda 함수를 한 번 이상 변경한 경우 이 플래그를 되돌릴 필요가 없습니다.

이러한 플래그를 되돌리기 위한 구문은 다음과 같습니다. `cdk.json`

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

```
}
```

속성

Aspect는 주어진 범위 내의 모든 구문에 연산을 적용하는 방법입니다. 애스펙트는 태그를 추가하는 등의 방법으로 구문을 수정할 수 있습니다. 또는 모든 버킷이 암호화되었는지 확인하는 등 구문의 상태를 확인할 수도 있습니다.

한 구문과 동일한 범위에 있는 모든 구문에 애스펙트를 적용하려면 다음 `Aspects.of(SCOPE).add()`예제와 같이 새 어스펙트를 호출해야 합니다.

TypeScript

```
Aspects.of(myConstruct).add(new SomeAspect(...));
```

JavaScript

```
Aspects.of(myConstruct).add(new SomeAspect(...));
```

Python

```
Aspects.of(my_construct).add(SomeAspect(...))
```

Java

```
Aspects.of(myConstruct).add(new SomeAspect(...));
```

C#

```
Aspects.Of(myConstruct).add(new SomeAspect(...));
```

Go

```
awscdk.Aspects_Of(stack).Add(awscdk.NewTag(...))
```

AWS CDK 에서는 속성을 사용하여 리소스에 태그를 지정하지만 프레임워크는 다른 용도로도 사용될 수 있습니다. 예를 들어, 이를 사용하여 상위 수준 구성으로 정의된 AWS CloudFormation 리소스를 확인하거나 변경할 수 있습니다.

특징 세부 정보

어스펙트는 방문자 패턴을 사용합니다. 애스펙트는 다음 인터페이스를 구현하는 클래스입니다.

TypeScript

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

JavaScript

JavaScript 언어 기능으로는 인터페이스가 없습니다. 따라서 애스펙트는 단순히 조작할 노드를 받아들이는 `visit` 메서드가 있는 클래스의 인스턴스일 뿐입니다.

Python

Python에는 언어 기능으로서의 인터페이스가 없습니다. 따라서 애스펙트는 단순히 연산할 노드를 받아들이는 `visit` 메서드가 있는 클래스의 인스턴스일 뿐입니다.

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)
}
```

를 `Aspects.of(SCOPE).add(...)` 호출하면 구문이 내부 애스펙트 목록에 애스펙트를 추가합니다. 를 사용하여 목록을 얻을 수 `Aspects.of(SCOPE)` 있습니다.

준비 단계에서 는 구문과 각 하위 구성요소의 객체 `visit` 메서드를 하향식 순서로 AWS CDK 호출합니다.

`visit`메서드는 구문의 모든 내용을 자유롭게 변경할 수 있습니다. 강력한 형식의 언어에서는 구조별 속성이나 메서드에 액세스하기 전에 수신된 구문을 보다 구체적인 형식으로 캐스팅하십시오.

어스펙트는 독립적이고 정의 후에는 변경할 수 없기 때문에 `Stage` 구문 경계를 넘어 `Stages` 전파되지 않습니다. `Stage`구성 자체 (또는 더 낮은 구문) 에 애스펙트를 적용하면 해당 구문 내부의 구문을 방문할 수 있습니다. `Stage`

예

다음 예제는 스택에서 생성된 모든 버킷에 버전 관리가 활성화되어 있는지 확인합니다. 이 애스펙트는 검증에 실패한 구문에 오류 주석을 추가합니다. 이로 인해 synth 작업이 실패하고 생성된 클라우드 어셈블리를 배포할 수 없게 됩니다.

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());
```

시작하기 AWS CDK

첫 번째 CDK 앱을 AWS CDK CLI 설치하고 AWS Cloud Development Kit (AWS CDK) 생성하여 시작하십시오.

주제

필수 조건

권장 리소스

시작하기 전에 다음 사항에 대한 기본적인 이해를 하는 것이 좋습니다. AWS CDK

- 에 대한 소개 AWS CDK. 자세한 내용은 이게 뭐야 AWS CDK? 단원을 참조하십시오.
- 핵심 개념 뒤에 숨어 AWS CDK있습니다. 자세한 내용은 AWS CDK 개념를 참조하세요.
- AWS 서비스 이를 통해 관리하고 싶은 것들 AWS CDK.
- AWS Identity and Access Management. 자세한 내용은 IAM이란 무엇입니까? 를 참조하십시오. 그리고 IAM 아이덴티티 센터란 무엇입니까?
- AWS CloudFormation AWS CloudFormation 서비스를 AWS CDK 활용하여 CDK에서 생성된 리소스를 프로비저닝하기 때문입니다. 자세한 내용은 AWS CloudFormation이란 무엇입니까?를 참조하십시오.
- 에서 사용할 지원되는 프로그래밍 언어입니다. AWS CDK

로컬 환경을 준비하세요.

기본 언어에 관계없이 모든 AWS CDK 개발자는 Node.js14.15.0 이상이 필요합니다. 지원되는 모든 프로그래밍 언어는 에서 실행되는 동일한 백엔드를 사용합니다. Node.js 장기 지원이 가능한 버전을 사용하는 것이 좋습니다. 조직에 따라 권장 사항이 다를 수 있습니다.

> **Important**
>
> Node.js 버전 13.0.0부터 13.6.0까지는 종속성과의 호환성 문제로 AWS CDK 인해 과 호환되지 않습니다.

기타 사전 요구 사항은 응용 프로그램을 개발하는 데 사용하는 언어에 따라 다르며 다음과 같습니다. AWS CDK

TypeScript

- TypeScript 3.8 이상 () `npm -g install typescript`

JavaScript

추가 요구 사항 없음

Python

- Python 3.7 이상 (및 포함) `pip` `virtualenv`

Java

- 자바 개발 키트 (JDK) 8 (일명 1.8) 이상
- 아파치 메이븐 3.5 이상

자바 IDE 권장 (이 안내서의 일부 예제에서는 Eclipse를 사용합니다). IDE는 Maven 프로젝트를 가져올 수 있어야 합니다. 프로젝트가 Java 1.8을 사용하도록 설정되어 있는지 확인하세요. JAVA_HOME 환경 변수를 JDK를 설치한 경로로 설정합니다.

C#

.NET 코어 3.1 이상 또는 .NET 6.0 이상.

비주얼 스튜디오 2019 (모든 에디션) 또는 비주얼 스튜디오 코드를 권장합니다.

Go

1.1.8 이상 버전을 선택하세요.

자세한 내용은 해당 언어의 사전 요구 사항 섹션을 참조하십시오.

- the section called “에서 TypeScript”
- the section called “에서 JavaScript”
- the section called “Python에서”
- the section called “자바에서”
- the section called “C #에서”
- the section called “In Go”

> **타사 언어 지원 중단**
>
> 각 언어 버전은 해당 기간 EOL (End Of Life) 까지만 지원되며 사전 통지로 변경될 수 있습니다.

1단계: 만들기 AWS 계정

를 AWS처음 사용하는 경우 관리 사용자를 AWS 계정 등록하고 생성해야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 설정을 참조하십시오.

와 상호 작용할 때는 AWS 보안 자격 증명을 지정하여 자신이 누구인지 AWS, 요청한 리소스에 액세스할 권한이 있는지 여부를 확인합니다. AWS 보안 자격 증명을 사용하여 요청을 인증하고 권한을 부여합니다. 자세히 알아보려면 IAM 사용 AWS 설명서의 보안 자격 증명을 참조하십시오.

2단계: 프로그래밍 방식 액세스 구성

로컬 AWS CDK 환경에서 를 사용하여 개발할 때는 를 사용하여 리소스와 상호 AWS 서비스 작용하고 AWS 리소스를 AWS CDK CLI 관리해야 합니다. 를 AWS CDK CLI 사용하려면 프로그래밍 방식 액세스를 구성해야 합니다. 프로그래밍 방식 액세스를 구성하는 다양한 방법에 대해 자세히 알아보려면 AWS SDK 및 도구 참조 가이드의 인증 및 액세스를 참조하십시오.

고용주로부터 인증 방법을 부여받지 않은 신규 사용자의 경우 를 사용하는 것이 좋습니다. AWS IAM Identity Center이 방법에는 AWS Command Line Interface (AWS CLI) 를 설치하여 구성에 사용하고 AWS 액세스 포털에 로그인하는 방법이 포함됩니다. IAM Identity Center를 사용하여 프로그래밍 액세스를 구성하려면 AWS SDK 및 도구 참조 안내서의 IAM ID 센터 인증을 참조하십시오. 완료 후에는 환경에 다음 요소가 포함되어야 합니다.

- 는 AWS CLI애플리케이션을 실행하기 전에 AWS 액세스 포털 세션을 시작하는 데 사용합니다.

- 에서 참조할 수 있는 구성 값 집합이 포함된 [default] 프로필이 있는 공유 AWSconfig 파일입니다. AWS CDK이 파일의 위치를 찾으려면 AWS SDK 및 도구 참조 가이드에서 공유 파일의 위치를 참조하세요.
- 공유 config 파일은 region 설정을 지정합니다. 이렇게 하면 AWS 요청에 대한 기본 AWS 리전 AWS CDK 용도가 설정됩니다.
- 는 요청을 보내기 전에 프로필의 SSO 토큰 공급자 구성을 AWS CDK 사용하여 자격 증명을 획득합니다. AWSsso_role_name값은 IAM Identity Center 권한 집합에 연결된 IAM 역할로, 애플리케이션에서 AWS 서비스 사용되는 사용자에 대한 액세스를 허용해야 합니다.

  다음 샘플 config 파일은 SSO 토큰 공급자 구성으로 설정된 기본 프로필을 보여줍니다. 프로필의 sso_session 설정은 이름이 지정된 sso-session 섹션을 참조합니다. sso-session섹션에는 AWS 액세스 포털 세션을 시작하기 위한 설정이 포함되어 있습니다.

```
[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
```

AWS 액세스 포털 세션 시작

AWS 서비스액세스하기 전에 IAM Identity Center 인증을 사용하여 자격 증명을 AWS CDK 확인하려면 활성 AWS 액세스 포털 세션이 필요합니다. 구성된 세션 길이에 따라 액세스는 결국 만료되며 인증 오류가 발생합니다. AWS CDK 에서 다음 명령을 AWS CLI 실행하여 AWS 액세스 포털에 로그인합니다.

```
aws sso login
```

SSO 토큰 공급자 구성에서 기본 프로필 대신 명명된 프로필을 사용하는 경우 명령은 입니다aws sso login --profile *NAME*. 또한 --profile 옵션이나 AWS_PROFILE 환경 변수를 사용하여 cdk 명령을 실행할 때 이 프로필을 지정하십시오.

이미 활성 세션이 있는지 테스트하려면 다음 AWS CLI 명령을 실행합니다.

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

이 명령에 대한 응답은 공유 `config` 파일에 구성된 IAM Identity Center 계정 및 권한 집합을 보고해야 합니다.

> **Note**
>
> 이미 활성 AWS 액세스 포털 세션이 있고 실행 `aws sso login` 중인 경우에는 자격 증명을 제공하지 않아도 됩니다.
> 로그인 과정에서 데이터에 AWS CLI 대한 접근을 허용하라는 메시지가 표시될 수 있습니다.
> AWS CLI 는 Python용 SDK를 기반으로 구축되었으므로 권한 메시지에는 다양한 `botocore` 이름이 포함될 수 있습니다.

3단계: 설치 AWS CDKCLI

다음 Node Package Manager 명령을 사용하여 AWS CDK CLI 전역적으로 설치합니다.

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

> **Note**
>
> 권한 오류가 발생하고 시스템에 대한 관리자 액세스 권한이 있는 경우 시도해 보십시오`sudo npm install -g aws-cdk`.

다음 명령을 실행하여 성공적으로 설치되었는지 확인합니다. 버전 번호를 AWS CDK CLI 출력해야 합니다.

```
cdk --version
```

오류 메시지가 표시되면 다음을 AWS CDK CLI 실행하여 를 제거해 보십시오.

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

그런 다음 단계를 반복하여 를 다시 설치합니다. AWS CDK CLI

여전히 오류가 발생하면 현재 프로젝트와 글로벌 node-modules 폴더에서 node-modules 폴더를 삭제하십시오. 이 폴더를 찾으려면 를 실행하십시오npm config get prefix.

AWS CDK CLI는 이전 단계에서 구성한 소스로부터 보안 자격 증명을 가져옵니다.

> **Note**
>
> CDK 툴킷 v2는 기존 CDK v1 프로젝트와 함께 작동합니다. 하지만 새 CDK v1 프로젝트는 초기화할 수 없습니다. the section called “새 사전 요구 사항”그렇게 할 수 있어야 하는지 확인해 보세요.

4단계: 환경 부트스트랩

리소스를 배포하려는 각 AWS 환경은 부트스트랩해야 합니다.

부트스트랩하려면 다음을 실행하세요.

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

> **Tip**
>
> AWS 계좌 번호가 없는 경우 에서 확인할 수 있습니다. AWS Management Console또는 앱을 AWS CLI 설치한 경우 다음 명령을 실행하면 계좌 번호를 포함한 기본 계정 정보가 표시됩니다.
>
> ```
> aws sts get-caller-identity
> ```
>
> 로컬 AWS 구성에서 이름이 지정된 프로필을 만든 경우 --profile 옵션을 사용하여 특정 프로필의 계정 정보를 표시할 수 있습니다. 다음 예는 prod 프로필의 계정 정보를 표시하는 방법을 보여줍니다.
>
> ```
> aws sts get-caller-identity --profile prod
> ```
>
> 기본 지역을 표시하려면 를 사용하십시오aws configure get.
>
> ```
> aws configure get region
> aws configure get region --profile prod
> ```

옵션 AWS CDK 도구

AWS Toolkit for Visual Studio Code는 Visual Studio Code용 오픈 소스 플러그인으로, 응용 프로그램을 만들고 디버그하고 배포하는 데 도움이 됩니다. AWS이 툴킷은 응용 프로그램 개발을 AWS CDK 위한 통합 환경을 제공합니다. 여기에는 AWS CDK 프로젝트를 나열하고 AWS CDK 애플리케이션의 다양한 구성 요소를 탐색할 수 있는 CDK Explorer 기능이 포함되어 있습니다. 플러그인을 설치하고 탐색기 사용에 대해 자세히 알아보십시오. AWS CDK

다음 단계

이제 를 설치했으니 이를 사용하여 첫 번째 AWS CDK 앱을 빌드하십시오. AWS CDK CLI

선호하는 프로그래밍 AWS CDK 언어로 를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오지원되는 프로그래밍 언어로 작업하기 AWS CDK.

AWS CDK 는 오픈 소스 프로젝트입니다. 기여하려면 기여를 AWS Cloud Development Kit (AWS CDK) 참조하십시오.

자세히 알아보기

에 대한 자세한 내용은 AWS CDK다음을 참조하십시오.

- CDK 워크숍 — 심층 실무 워크숍.
- API 레퍼런스 — 사용할 수 있는 구문을 살펴보세요. AWS 서비스
- 구성 허브 — CDK 커뮤니티에서 구문을 찾아보세요.
- AWS CDK 예제 — 프로젝트의 코드 예제를 살펴보세요. AWS CDK

첫 번째 AWS CDK 앱

첫 번째 CDK 앱을 AWS Cloud Development Kit (AWS CDK) 빌드하여 사용을 시작하세요.

이 자습서를 시작하기 전에 다음을 완료하는 것이 좋습니다.

- 이게 뭐야 AWS CDK?에 대한 소개는 를 참조하십시오 AWS CDK.
- 의 핵심 AWS CDK 개념 개념에 대해 알아보려면 을 참조하십시오 AWS CDK.
- 에서 사전 요구 사항 및 AWS CDK 설정 단계를 살펴보십시오. 시작하기 AWS CDK

주제

이 자습서 소개

이 자습서에서는 간단한 AWS CDK 앱을 만들고 배포합니다. 이 앱에는 단일 Amazon Simple Storage Service (Amazon S3) 버킷 리소스가 있는 스택 하나가 포함되어 있습니다. 이 자습서를 통해 다음 내용을 배우게 됩니다.

- AWS CDK 프로젝트의 구조.
- AWS CDK 앱 생성 방법.
- AWS 구성 라이브러리를 사용하여 앱, 스택, AWS 리소스를 정의하는 방법
- CDK를 사용하여 CDK CLI 앱을 합성, 비교, 배포, 삭제하는 방법.
- CDK 앱을 수정하고 재배포하여 배포된 리소스를 업데이트하는 방법.

표준 AWS CDK 개발 워크플로는 다음 단계로 구성됩니다.

1. AWS CDK 앱 만들기 - 여기서는 에서 제공하는 템플릿을 사용합니다 AWS CDK CLI.
2. 스택 및 리소스 정의 - 구문을 사용하여 앱 내의 스택과 AWS 리소스를 정의합니다.
3. 앱 빌드 — 이 단계는 선택 사항입니다. 필요한 경우 이 단계를 AWS CDK CLI 자동으로 수행합니다. 구문 및 유형 오류를 식별하려면 이 단계를 수행하는 것이 좋습니다.
4. 스택 합성 — 이 단계에서는 앱의 각 스택에 대한 AWS CloudFormation 템플릿을 생성합니다. 이 단계는 정의된 AWS 리소스의 논리적 오류를 식별하는 데 유용합니다.

5. 앱 배포 — 리소스를 프로비저닝하는 AWS CloudFormation 데 사용하여 AWS 환경에 배포합니다. 배포 과정에서 앱과 관련된 모든 권한 문제를 식별할 수 있습니다.

일반적인 워크플로를 통해 이전 단계를 반복하여 앱을 수정하거나 디버깅합니다.

AWS CDK 프로젝트에는 버전 관리를 사용하는 것이 좋습니다.

1단계: 앱 만들기

CDK 앱은 자체 로컬 모듈 종속 항목이 있는 자체 디렉터리에 있어야 합니다. 개발 머신에서 새 디렉터리를 생성하세요. 다음은 새 `hello-cdk` 디렉터리를 생성하는 예제입니다.

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

> **Important**
>
> 프로젝트 디렉터리의 `hello-cdk` 이름을 여기에 표시된 대로 정확히 지정해야 합니다. AWS CDK 프로젝트 템플릿은 디렉터리 이름을 사용하여 생성된 코드에 있는 항목의 이름을 지정합니다. 다른 이름을 사용하면 이 자습서의 코드가 작동하지 않습니다.

그런 다음 새 디렉터리에서 cdk init 명령을 사용하여 앱을 초기화합니다. `--language`옵션을 사용하여 app 템플릿과 선호하는 프로그래밍 언어를 지정합니다. 다음은 그 예제입니다.

TypeScript

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

JavaScript

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

Python

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

앱을 만든 후 다음 두 명령도 입력합니다. 이는 앱의 Python 가상 환경을 활성화하고 AWS CDK 핵심 종속성을 설치합니다.

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

Java

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

IDE를 사용하는 경우 이제 프로젝트를 열거나 가져올 수 있습니다. 예를 들어 Eclipse에서는 [파일] > [가져오기] > [Maven] > [기존 Maven 프로젝트] 를 선택합니다. 프로젝트 설정이 Java 8 (1.8) 을 사용하도록 설정되어 있는지 확인하십시오.

C#

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

Visual Studio를 사용하는 경우 `src` 디렉터리에서 솔루션 파일을 여십시오.

Go

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

앱을 만든 후에는 다음 명령도 입력하여 앱에 필요한 AWS Construct Library 모듈을 설치합니다.

```
go get
```

이 cdk init 명령어는 `hello-cdk` 디렉터리 내에 여러 개의 파일과 폴더를 만들어 AWS CDK 앱의 소스 코드를 정리하는 데 도움이 됩니다. 이를 총칭하여 AWS CDK 프로젝트라고 합니다. 잠시 시간을 내어 CDK 프로젝트를 살펴보세요.

Git설치가 완료되면 사용하여 생성한 각 프로젝트도 cdk init 저장소로 초기화됩니다. Git

2단계: 앱 빌드

대부분의 프로그래밍 환경에서는 코드를 변경한 후 빌드하거나 컴파일합니다. CDK가 이 단계를 CLI 자동으로 AWS CDK 수행하므로 에서는 이 작업을 수행할 필요가 없습니다. 하지만 구문 및 유형 오류를 포착하려는 경우에는 여전히 수동으로 빌드할 수 있습니다. 다음은 그 예제입니다.

TypeScript

```
npm run build
```

JavaScript

빌드 단계는 필요하지 않습니다.

Python

빌드 단계는 필요하지 않습니다.

Java

```
mvn compile -q
```

또는 Eclipse에서 Control-B를 누르세요 (다른 Java IDE는 다를 수 있음).

C#

```
dotnet build src
```

또는 비주얼 스튜디오에서 F6을 누릅니다.

Go

```
go build
```

3단계: 앱의 스택 나열

앱에 스택을 나열하여 앱이 올바르게 생성되었는지 확인하세요. 다음을 실행합니다.

```
cdk ls
```

출력이 `HelloCdkStack` 표시되어야 합니다. 이 출력이 보이지 않으면 프로젝트의 작업 디렉터리가 올바른지 확인하고 다시 시도하세요. 그래도 스택이 보이지 않으면 the section called “1단계: 앱 만들기” 반복해서 다시 시도하세요.

4단계: Amazon S3 버킷 추가

현재 CDK 앱에는 단일 스택이 포함되어 있습니다. 다음으로 스택 내에 Amazon Simple Storage 서비스 (Amazon S3) 버킷 리소스를 정의합니다. 이렇게 하려면 AWS 구성 라이브러리에서 `Bucket` L2 구문을 가져와서 사용해야 합니다.

`Bucket`구성을 가져오고 Amazon S3 버킷 리소스를 정의하여 CDK 앱을 수정합니다. 다음은 그 예제입니다.

TypeScript

lib/hello-cdk-stack.ts에서:

```
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

lib/hello-cdk-stack.js에서:

```
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

hello_cdk/hello_cdk_stack.py에서:

```
import aws_cdk as cdk
import aws_cdk.aws_s3 as s3
```

```
class HelloCdkStack(cdk.Stack):

    def __init__(self, scope: cdk.App, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

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

Java

src/main/java/com/myorg/HelloCdkStack.java에서:

```
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#

src/HelloCdk/HelloCdkStack.cs에서:

```
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

hello-cdk.go에서:

```
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
}
```

구조를 좀 더 자세히 살펴보겠습니다. Bucket 모든 구문과 마찬가지로 이 Bucket 클래스는 세 개의 매개변수를 사용합니다.

- 범위 — Stack 클래스를 Bucket 구문의 부모로 정의합니다. AWS 리소스를 정의하는 모든 구문은 스택 범위 내에서 생성됩니다. 구문 내에 구문을 정의하여 계층 (트리) 을 만들 수 있습니다. 여기서는 대부분의 경우 범위가 this (selfinPython) 입니다.
- Id — AWS CDK 앱 Bucket 내 논리적 ID입니다. 이 ID와 스택 내 버킷 위치에 기반한 해시를 더하면 배포 중에 버킷을 고유하게 식별할 수 있습니다. AWS CDK 또한 앱에서 구조를 업데이트하고 배포된 리소스를 업데이트하기 위해 재배포할 때도 이 ID를 참조합니다. 여기서 논리 ID는 MyFirstBucket 다음과 같습니다. 버킷은 bucketName 속성과 함께 지정된 이름을 가질 수도 있습니다. 이는 논리적 ID와 다릅니다.
- props — 버킷의 속성을 정의하는 값 번들. 여기서는 versioned 속성을 로 true 정의하여 버킷에 있는 파일의 버전 관리를 활성화합니다.

  Props는 에서 지원하는 언어에 따라 다르게 표시됩니다. AWS CDK

  - TypeScript In 및 JavaScript, props 는 단일 인수이며 원하는 속성이 포함된 객체를 전달합니다.
  - Python에서 props는 키워드 인자로 전달됩니다.
  - Java에서는 props를 전달하기 위한 빌더가 제공됩니다. 두 가지가 있습니다. 하나는 BucketProps for이고 다른 하나는 Bucket for로 구성과 props 객체를 한 번에 빌드할 수 있게 해줍니다. 이 코드는 후자를 사용합니다.
  - C #에서는 개체 이니셜라이저를 사용하여 BucketProps 개체를 인스턴스화하고 이를 세 번째 매개 변수로 전달합니다.

  구문의 props가 선택사항인 경우 매개 변수를 완전히 생략할 수 있습니다. props

모든 구문은 이와 같은 세 개의 인수를 사용하므로 새로운 인수를 익혀도 방향을 잡기가 쉽습니다. 예상할 수 있듯이, 원하는 구문을 서브클래스화하여 필요에 맞게 확장하거나 기본값을 변경하려는 경우에도 사용할 수 있습니다.

5단계: 템플릿 합성 AWS CloudFormation

다음과 같이 앱용 AWS CloudFormation 템플릿을 합성합니다.

```
cdk synth
```

앱에 둘 이상의 스택이 포함된 경우 합성할 스택을 지정해야 합니다. 앱에 단일 스택이 포함되어 있으므로 CDK는 합성할 스택을 CLI 자동으로 감지합니다.

`cdk synth`실행하지 않으면 배포 시 CLI CDK가 이 단계를 자동으로 수행합니다. 하지만 각 배포 전에 이 단계를 실행하는 것이 좋습니다.

> **Tip**
>
> 와 같은 `--app is required ...` 오류가 발생하는 경우 CDK CLI 명령을 실행 중인 디렉터리를 확인하세요. 기본 앱 디렉터리에 있어야 합니다.

`cdk synth`명령어는 앱을 실행합니다. 그러면 앱의 각 스택에 대한 AWS CloudFormation 템플릿이 만들어집니다. CLICDK는 명령줄에 템플릿의 YAML 형식 버전을 표시하고 디렉터리에 템플릿의 JSON 형식 버전을 저장합니다. `cdk.out` 다음은 템플릿에 정의된 버킷을 보여주는 명령줄 출력 스니펫입니다. AWS CloudFormation

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

> **Note**
>
> 생성된 모든 템플릿에는 기본적으로 `AWS::CDK::Metadata` 리소스가 포함되어 있습니다. AWS CDK 팀은 이 메타데이터를 사용하여 AWS CDK 사용에 대한 통찰력을 얻고 개선 방법을 찾습니다. 버전 보고를 거부하는 방법을 비롯한 자세한 내용은 을 참조하십시오버전 보고.

생성된 템플릿은 AWS CloudFormation 콘솔 또는 기타 AWS CloudFormation 배포 도구를 통해 배포할 수 있습니다. CDK를 사용하여 CLI 배포할 수도 있습니다. 다음 단계에서는 CLI CDK를 사용하여 배포합니다.

6단계: 스택 배포

CDK를 AWS CloudFormation 사용하기 위해 CDK 스택을 CLI 배포하려면 다음을 실행하세요.

```
cdk deploy
```

> **Important**
>
> 배포 전에 환경을 한 번 부트스트래핑해야 합니다. AWS 지침은 환경 부트스트랩을 참조하십시오.

마찬가지로 앱에는 단일 스택이 포함되어 있으므로 AWS CDK 스택을 지정할 필요가 없습니다. `cdk synth`

코드에 보안 문제가 있는 경우 CLI CDK는 요약을 출력합니다. 배포를 계속하려면 이를 확인해야 합니다. 이 자습서의 앱에는 이러한 영향이 없습니다.

실행 후 스택이 `cdk deploy` 배포되면 CDK에 진행 정보가 CLI 표시됩니다. 완료되면 AWS CloudFormation 콘솔로 이동하여 `HelloCdkStack` 스택을 확인할 수 있습니다. Amazon S3 콘솔로 이동하여 `MyFirstBucket` 리소스를 볼 수도 있습니다.

축하합니다! 를 사용하여 첫 번째 스택을 배포했습니다 AWS CDK. 다음으로 앱을 수정하고 다시 배포하여 리소스를 업데이트합니다.

7단계: 앱 수정

이 단계에서는 스택이 삭제될 때 자동으로 삭제되도록 구성하여 Amazon S3 버킷을 수정합니다. 이 수정에는 버킷 `RemovalPolicy` 속성 변경이 포함됩니다. 또한 버킷을 삭제하기 전에 버킷에서 객체를

CLI 삭제하도록 CDK를 구성하도록 `autoDeleteObjects` 속성을 구성합니다. 기본적으로 객체가 포함된 Amazon S3 버킷은 삭제하지 AWS CloudFormation 않습니다.

다음 예제를 사용하여 리소스를 수정하십시오.

TypeScript

`lib/hello-cdk-stack.ts` 업데이트

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

JavaScript

`lib/hello-cdk-stack.js` 업데이트

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

Python

`hello_cdk/hello_cdk_stack.py` 업데이트

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

Java

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

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

```
        .build();
```

C#

src/HelloCdk/HelloCdkStack.cs 업데이트

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

Go

hello-cdk.go 업데이트

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

현재 코드 변경으로 배포된 Amazon S3 버킷 리소스가 직접 업데이트되지는 않았습니다. 코드는 리소스의 원하는 상태를 정의합니다. 배포된 리소스를 수정하려면 CLI CDK를 사용하여 원하는 상태를 새 AWS CloudFormation 템플릿에 합성합니다. 그런 다음 새 AWS CloudFormation 템플릿을 변경 세트로 배포합니다. 변경 세트는 원하는 새 상태에 도달하는 데 필요한 변경만 수행합니다.

이러한 변경 내용을 보려면 cdk diff 명령을 사용하십시오. 다음을 실행합니다.

```
cdk diff
```

CDK는 AWS 계정 계정에 HelloCdkStack 스택용 최신 AWS CloudFormation 템플릿을 CLI 쿼리합니다. 그런 다음 최신 템플릿을 앱에서 방금 합성한 템플릿과 비교합니다. 이 명령의 출력은 다음과 같습니다.

```
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
```

이 차이점에는 네 개의 섹션이 있습니다.

- IAM 설명 변경 및 IAM 정책 변경 — Amazon S3 버킷에 `AutoDeleteObjects` 속성을 설정했기 때문에 이러한 권한 변경이 적용됩니다. 자동 삭제 기능은 버킷 자체가 삭제되기 전에 사용자 지정 리소스를 사용하여 버킷의 객체를 삭제합니다. IAM 객체는 버킷에 대한 사용자 지정 리소스의 코드 액세스 권한을 부여합니다.
- 파라미터 - 는 이러한 항목을 AWS CDK 사용하여 사용자 지정 리소스의 AWS Lambda 함수 자산을 찾습니다.
- 리소스 - 이 스택의 신규 및 변경된 리소스입니다. 앞서 언급한 IAM 객체, 사용자 지정 리소스 및 관련 Lambda 함수가 추가되는 것을 볼 수 있습니다. 또한 `DeletionPolicy` 버킷과 `UpdateReplacePolicy` 속성이 업데이트되고 있는 것을 확인할 수 있습니다. 이렇게 하면 스택과 함께 버킷을 삭제하고 새 스택으로 교체할 수 있습니다.

AWS CDK 앱에서 지정했지만 결과 `RemovalPolicy` AWS CloudFormation 템플릿에 `DeletionPolicy` 속성이 추가된 것을 확인할 수 있습니다. 이는 에서 속성에 다른 이름을 AWS CDK 사용하기 때문입니다. AWS CDK 기본값은 스택이 삭제될 때 버킷을 유지하는 것이고 AWS

CloudFormation 기본값은 스택을 삭제하는 것입니다. 자세한 설명은 the section called “제거 정책” 섹션을 참조하세요.

를 실행하여 새 AWS CloudFormation 템플릿을 볼 수 있습니다cdk synth. CDK 앱을 약간 변경하면 이제 새 AWS CloudFormation 템플릿에 원래 AWS CloudFormation 템플릿에 비해 많은 추가 코드 줄이 포함됩니다.

다음으로, 다음을 실행하여 앱을 배포하세요.

```
cdk deploy
```

Diff에서 이미 확인한 보안 정책 변경 사항을 AWS CDK 알려줍니다. 변경 사항을 승인하고 업데이트된 스택을 y 배포하려면 Enter를 입력하십시오. CLICDK는 원하는 대로 변경할 수 있도록 스택을 배포합니다. 다음은 출력의 예제입니다.

```
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
```

8단계: 앱 리소스 삭제

이제 이 자습서를 완료했으므로 배포된 AWS CloudFormation 스택과 이와 관련된 모든 리소스를 삭제할 수 있습니다. 이는 불필요한 비용을 최소화하고 환경을 깨끗하게 유지하기 위한 좋은 방법입니다. 다음을 실행합니다.

```
cdk destroy
```

y를 입력하여 변경 사항을 승인하고 스택을 삭제하십시오.

> **Note**
>
> 버킷을 변경하지 않으면 스택 삭제는 성공적으로 완료되지만 버킷은 분리되어 스택과 더 이상 연결되지 않게 됩니다. `RemovalPolicy`

다음 단계

축하합니다! 이 자습서를 완료하고 AWS CDK 를 사용하여 에서 리소스를 성공적으로 생성, 수정 및 삭제했습니다. AWS 클라우드이제 를 사용할 준비가 되었습니다 AWS CDK.

선호하는 프로그래밍 AWS CDK 언어로 를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오지원되는 프로그래밍 언어로 작업하기 AWS CDK.

추가 리소스는 다음을 참조하십시오.

- CDK 워크숍을 통해 좀 더 복잡한 프로젝트와 관련된 심도 있는 투어를 즐겨보세요.
- the section called “환경”,,,, the section called “자산” the section called “권한” the section called “컨텍스트”the section called “파라미터”, 등의 the section called “구문 사용자 지정하기” 개념을 더 자세히 살펴보세요.
- 선호하는 AWS 서비스에 사용할 수 있는 CDK 구조를 살펴보려면 API 레퍼런스를 참고하세요.
- Construct Hub를 방문하여 다른 사람들이 만든 AWS 구문을 찾아보세요.
- 사용 예를 살펴보세요. AWS CDK

AWS CDK 는 오픈 소스 프로젝트입니다. 기여하려면 기여를 참조하십시오. AWS Cloud Development Kit (AWS CDK)

AWS CDK v1에서 v2로 마이그레이션 AWS CDK

버전 2는 선호하는 프로그래밍 언어로 인프라를 코드로 쉽게 작성할 수 있도록 설계되었습니다. AWS Cloud Development Kit (AWS CDK) 이 항목에서는 v1과 v2 사이의 변경 사항에 대해 설명합니다. AWS CDK

> **Tip**
>
> AWS CDK v1과 함께 배포된 스택을 식별하려면 awscdk-v1-stack-finder 유틸리티를 사용하십시오.

v1에서 CDK v2로의 주요 변경 사항은 다음과 같습니다. AWS CDK

- AWS CDK v2는 코어 라이브러리를 포함한 AWS 구성 라이브러리의 안정적인 부분을 단일 패키지로 통합합니다. `aws-cdk-lib` 개발자는 더 이상 자신이 사용하는 개별 AWS 서비스에 대한 추가 패키지를 설치할 필요가 없습니다. 또한 이 단일 패키지 접근 방식을 사용하면 다양한 CDK 라이브러리 패키지의 버전을 동기화할 필요가 없습니다.

  에서 사용 가능한 리소스를 정확히 나타내는 L1 (CFNxxxx) 구문은 항상 안정적인 것으로 간주되므로 에 AWS CloudFormation포함됩니다. `aws-cdk-lib`
- 아직 커뮤니티와 협력하여 새로운 L2 또는 L3 구조를 개발하고 있는 실험용 모듈은 포함되지 않습니다. `aws-cdk-lib` 대신 개별 패키지로 배포됩니다. 실험용 패키지의 이름은 `alpha` 접미사와 시맨틱 버전 번호로 지정됩니다. 시맨틱 버전 번호는 호환되는 AWS Construct Library의 첫 번째 버전과 일치하며 접미사도 붙습니다. `alpha` 구문은 안정 버전으로 지정된 `aws-cdk-lib` 후 안으로 이동하므로 기본 구성 라이브러리는 엄격한 시맨틱 버전 관리를 준수할 수 있습니다.

  안정성은 서비스 수준에서 지정됩니다. 예를 들어, 이 글을 쓰는 시점에는 L1 구조만 있는 AppFlow Amazon용 L2 구문을 하나 이상 생성하기 시작하면 이 구문은 라는 모듈에 먼저 나타납니다. `@aws-cdk/aws-appflow-alpha` 그러다가 새 구조가 고객의 `aws-cdk-lib` 기본적인 요구를 충족한다고 판단되는 시점으로 넘어갑니다.

  모듈이 안정으로 지정되어 `aws-cdk-lib` 통합되면 다음 글머리 기호에 설명된 “BetAn” 규칙을 사용하여 새 API가 추가됩니다.

  가 출시될 때마다 각 실험 모듈의 새 버전이 릴리스됩니다. AWS CDK하지만 대부분의 경우 동기화된 상태로 유지할 필요가 없습니다. 언제든지 `aws-cdk-lib` 업그레이드하거나 실험용 모듈을 사용

할 수 있습니다. 단, 관련된 두 개 이상의 실험 모듈이 서로 종속되어 있는 경우에는 동일한 버전이어야 합니다.

- 새 기능이 추가되는 안정 모듈의 경우 작업이 진행되는 동안 새 API (완전히 새로운 구조이든 기존 구문의 새로운 메서드나 속성이든 관계 없음) 에는 `Beta1` 접미사가 붙습니다. (주요 변경 사항이 필요한 경우 뒤에 `Beta2`Beta3, 등이 붙습니다.) API가 안정으로 지정되면 접미사가 없는 API 버전이 추가됩니다. 그러면 최신 메서드 (베타 또는 최종) 를 제외한 모든 메서드는 더 이상 사용되지 않습니다.

  예를 들어, 구문에 새 `grantPower()` 메서드를 추가하면 처음에는 로 표시됩니다. `grantPowerBeta1()` 주요 변경 사항이 필요한 경우 (예: 새 필수 매개 변수 또는 속성) 메서드의 다음 버전 이름이 `grantPowerBeta2()` 지정되는 식입니다. 작업이 완료되고 API가 완성되면 메서드 `grantPower()` (접미사 없음) 가 추가되고 BetAn 메서드는 더 이상 사용되지 않습니다.

  모든 베타 API는 다음 메이저 버전 (3.0) 릴리스까지 구성 라이브러리에 남아 있으며 시그니처는 변경되지 않습니다. 사용할 경우 지원 중단 경고가 표시되므로 최대한 빨리 API의 최종 버전으로 옮겨야 합니다. 그러나 향후 AWS CDK 2.x 릴리스에서는 애플리케이션이 손상되지 않습니다.

- `Construct`클래스가 관련 형식과 함께 에서 별도의 라이브러리로 추출되었습니다. AWS CDK 이는 Construct Programming Model을 다른 도메인에 적용하려는 노력을 지원하기 위한 것입니다. 직접 구문을 작성하거나 관련 API를 사용하는 경우 `constructs` 모듈을 종속 항목으로 선언하고 임포트를 약간 변경해야 합니다. CDK 앱 라이프사이클에 연결하는 것과 같은 고급 기능을 사용하는 경우 추가 변경이 필요할 수 있습니다. 자세한 내용은 RFC를 참조하십시오.
- AWS CDK v1.x 및 해당 구성 라이브러리에서 더 이상 사용되지 않는 속성, 메서드, 형식이 CDK v2 API에서 완전히 제거되었습니다. 지원되는 대부분의 언어에서 이러한 API는 v1.x에서 경고를 생성하므로 이미 대체 API로 마이그레이션했을 수 있습니다. CDK v1.x에서 더 이상 사용되지 않는 API의 전체 목록은 에서 확인할 수 있습니다. GitHub
- AWS CDK v1.x에서 기능 플래그에 의해 제한되었던 동작은 CDK v2에서 기본적으로 활성화됩니다. 이전 기능 플래그는 더 이상 필요하지 않으며 대부분의 경우 지원되지 않습니다. 아주 특정한 상황에서 CDK v1 동작으로 되돌릴 수 있는 몇 가지 기능이 아직 남아 있습니다. 자세한 설명은 the section called “기능 플래그 업데이트” 섹션을 참조하세요.
- CDK v2를 사용하면 배포하는 환경을 최신 부트스트랩 스택을 사용하여 부트스트랩해야 합니다. 레거시 부트스트랩 스택 (v1의 기본값) 은 더 이상 지원되지 않습니다. 또한 CDK v2에는 새 버전의 최신 스택이 필요합니다. 기존 환경을 업그레이드하려면 다시 부트스트랩하세요. 더 이상 최신 부트스트랩 스택을 사용하기 위해 기능 플래그 또는 환경 변수를 설정할 필요가 없습니다.

> **Important**
>
> 최신 부트스트랩 템플릿은 에서 암시하는 권한을 목록에 있는 모든 AWS 계정에 효과적으로 부여합니다. `--cloudformation-execution-policies` `--trust` 기본적으로 이렇게 하면 부트스트랩된 계정의 모든 리소스에 대한 읽기 및 쓰기 권한이 확장됩니다. 마음에 드는 정책 및 신뢰할 수 있는 계정으로 부트스트래핑 스택을 구성해야 합니다.

새 사전 요구 사항

v2의 요구 사항 대부분은 AWS CDK v1.x와 동일합니다. AWS CDK 추가 요구 사항은 여기에 나열되어 있습니다.

- TypeScript 개발자의 경우 TypeScript 3.8 이상이 필요합니다.
- CDK v2와 함께 사용하려면 새 버전의 CDK 툴킷이 필요합니다. 이제 CDK v2를 정식 버전으로 사용할 수 있게 되었으므로 CDK 툴킷을 설치할 때는 v2가 기본 버전입니다. CDK v1 프로젝트와 이전 버전과 호환되므로 CDK v1 프로젝트를 만들려는 경우가 아니면 이전 버전을 계속 설치하지 않아도 됩니다. 업그레이드하려면 다음을 실행하십시오`npm install -g aws-cdk`.

AWS CDK v2 개발자 프리뷰에서 업그레이드

CDK v2 개발자 프리뷰를 사용하는 경우 프로젝트에 릴리스 후보 버전 (예:) 이 종속되어 AWS CDK있는 것입니다. `2.0.0-rc1` 이를 로 `2.0.0` 업데이트한 다음 프로젝트에 설치된 모듈을 업데이트하십시오.

TypeScript

`npm install` 또는 `yarn install`

JavaScript

`npm install` 또는 `yarn install`

Python

```
python -m pip install -r requirements.txt
```

Java

```
mvn package
```

C#

```
dotnet restore
```

Go

```
go get
```

종속 항목을 업데이트한 후 CDK Toolkit을 릴리스 버전으로 `npm update -g aws-cdk` 업데이트하도록 실행하십시오.

v1에서 AWS CDK CDK v2로 마이그레이션

앱을 AWS CDK v2로 마이그레이션하려면 먼저 에서 기능 플래그를 업데이트하세요. `cdk.json` 그런 다음 앱이 작성된 프로그래밍 언어에 맞게 필요에 따라 앱의 종속성 및 가져오기를 업데이트하세요.

최신 v1으로 업데이트

많은 고객이 한 번에 이전 버전의 AWS CDK v1을 최신 버전의 v2로 업그레이드하는 것을 목격하고 있습니다. 물론 가능하긴 하지만, 수년에 걸친 변경에 따른 업그레이드 (안타깝게도 모두 현재 진행 중인 에볼루션 테스트 양이 같지는 않을 수도 있음) 뿐만 아니라 새 기본값과 다른 코드 구성을 사용하여 버전 간에 업그레이드해야 할 수도 있습니다.

가장 안전하게 업그레이드하고 예상치 못한 변경의 원인을 더 쉽게 진단하려면 먼저 최신 v1 버전으로 업그레이드한 다음 나중에 v2로 전환하는 두 단계를 구분하는 것이 좋습니다.

기능 플래그 업데이트

다음 v1 기능 플래그가 있는 `cdk.json` 경우 제거하세요. v2에서는 기본적으로 모두 활성화되기 때문입니다. AWS CDK 이전 효과가 인프라에 중요한 영향을 미치는 경우 소스 코드를 변경해야 합니다. 자세한 내용은 GitHub 의 플래그 목록을 참조하십시오.

- `@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`

특정 AWS CDK v1 동작으로 되돌리기 위해 몇 가지 v1 기능 플래그를 설정할 수 있습니다. 전체 참조는 아래 목록을 참조하십시오the section called “v1 동작으로 되돌리기”. `false` GitHub

두 유형의 플래그 모두에 대해 `cdk diff` 명령을 사용하여 합성된 템플릿의 변경 사항을 검사하여 이러한 플래그에 대한 변경 사항이 인프라에 영향을 미치는지 확인하십시오.

CDK 툴킷 호환성

CDK v2를 사용하려면 v2 이상의 CDK 툴킷이 필요합니다. 이 버전은 CDK v1 앱과 이전 버전과 호환됩니다. 따라서 v1을 사용하든 v2를 사용하든 상관없이 전 세계적으로 설치된 단일 버전의 CDK Toolkit을 모든 AWS CDK 프로젝트에 사용할 수 있습니다. 단, CDK 툴킷 v2는 CDK v2 프로젝트만 생성하는 경우는 예외입니다.

v1과 v2 CDK 프로젝트를 모두 생성해야 하는 경우 CDK 툴킷 v2를 전역적으로 설치하지 마십시오. (이미 설치한 경우 제거:.) `npm remove -g aws-cdk` CDK 툴킷을 호출하려면 를 사용하여 CDK 툴킷 v1 또는 v2를 원하는 npx 대로 실행하십시오.

```
npx aws-cdk@1.x init app --language typescript
npx aws-cdk@2.x init app --language typescript
```

> **Tip**
>
> cdk및 명령을 사용하여 원하는 버전의 CDK 툴킷을 호출할 수 있도록 cdk1 명령줄 별칭을 설정합니다.
>
> 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 $*
```

종속성 업데이트 및 가져오기

앱의 종속성을 업데이트한 다음 새 패키지를 설치합니다. 마지막으로 코드에서 임포트를 업데이트하세요.

TypeScript

애플리케이션

CDK 앱의 경우 다음과 `package.json` 같이 업데이트하십시오. v1 스타일의 개별 안정 모듈에 대한 종속성을 제거하고 애플리케이션에 `aws-cdk-lib` 필요한 최소 버전 (여기서는 2.0.0) 을 설정합니다.

실험적 구조는 이름이 로 끝나고 알파 버전 번호가 있는 독립적으로 버전이 지정된 별도의 패키지로 제공됩니다. `alpha` 알파 버전 번호는 `aws-cdk-lib` 호환되는 첫 번째 릴리스에 해당합니다. 여기서는 v2.0.0-alpha.1에 고정했습니다`aws-codestar`.

```
{
  "dependencies": {
    "aws-cdk-lib": "^2.0.0",
    "@aws-cdk/aws-codestar-alpha": "2.0.0-alpha.1",
    "constructs": "^10.0.0"
  }
}
```

라이브러리 구축하기

구성 라이브러리의 `aws-cdk-lib` 경우, 애플리케이션에 필요한 최저 버전 (여기서는 2.0.0) 을 설정하고 다음과 `package.json` 같이 업데이트하십시오.

참고로 이는 피어 종속성과 개발 종속성 둘 다로 `aws-cdk-lib` 나타납니다.

```
{
```

```
  "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**
>
> v2 호환 라이브러리를 출시할 때는 라이브러리 버전 번호에 메이저 버전 범프를 적용해야 합니다. 이는 라이브러리 소비자를 위한 주요 변경 사항이기 때문입니다. 단일 라이브러리로 CDK v1과 v2를 모두 지원하는 것은 불가능합니다. 아직 v1을 사용하는 고객을 계속 지원하려면 이전 릴리스를 병렬로 유지 관리하거나 v2용 새 패키지를 만들 수 있습니다. AWS CDK v1 고객을 얼마나 오랫동안 계속 지원할지는 여러분에게 달려 있습니다. 2022년 6월 1일에 유지 보수에 들어갔다가 2023년 6월 end-of-life 1일에 도달할 예정인 CDK v1 자체의 라이프사이클에서 힌트를 얻을 수 있을 것입니다. 자세한 내용은 유지 관리 정책을 참조하십시오AWS CDK .

라이브러리와 앱 모두

`npm install`또는 `yarn install` 를 실행하여 새 종속성을 설치합니다.

새 `constructs` 모듈, 최상위 수준 등의 코어 유형, `App` 그리고 `Stack` 네임스페이스에서 사용하는 서비스의 안정적인 Construct Library 모듈에서 가져오도록 임포트를 `Construct` 변경하세요. `aws-cdk-lib` `aws-cdk-lib`

```
import { Construct } from 'constructs';
import { App, Stack } from 'aws-cdk-lib';                // core constructs
import { aws_s3 as s3 } from 'aws-cdk-lib';              // stable module
import * as codestar from '@aws-cdk/aws-codestar-alpha';  // experimental module
```

JavaScript

`package.json`다음과 같이 업데이트하세요. v1 스타일의 개별 안정 모듈에 대한 종속성을 제거하고 애플리케이션에 `aws-cdk-lib` 필요한 최소 버전 (여기서는 2.0.0) 을 설정합니다.

실험적 구조는 이름이 로 끝나고 알파 버전 번호가 있는 독립적으로 버전이 지정된 별도의 패키지로 제공됩니다. alpha 알파 버전 번호는 aws-cdk-lib 호환되는 첫 번째 릴리스에 해당합니다. 여기서는 v2.0.0-alpha.1에 고정했습니다aws-codestar.

```
{
  "dependencies": {
    "aws-cdk-lib": "^2.0.0",
    "@aws-cdk/aws-codestar-alpha": "2.0.0-alpha.1",
    "constructs": "^10.0.0"
  }
}
```

또는 를 실행하여 새 종속성을 설치합니다. npm install yarn install

다음과 같이 앱 가져오기를 변경하세요.

- 새 Construct constructs 모듈에서 가져오기
- 코어 유형 (예: App 및Stack) 을 최상위 수준에서 가져옵니다. aws-cdk-lib
- 아래의 네임스페이스에서 AWS 구조 라이브러리 모듈을 가져옵니다. 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

setup.py다음과 같이 install_requires 정의를 requirements.txt 업데이트하십시오. v1 스타일의 개별 안정 모듈에 대한 종속성을 제거합니다.

실험적 구조는 이름이 로 끝나고 알파 버전 번호가 있는 별도의 독립 버전 패키지로 제공됩니다. alpha 알파 버전 번호는 aws-cdk-lib 호환되는 첫 번째 릴리스에 해당합니다. 여기서는 v2.0.0alpha1에 고정했습니다aws-codestar.

```
install_requires=[
     "aws-cdk-lib>=2.0.0",
     "constructs>=10.0.0",
     "aws-cdk.aws-codestar-alpha>=2.0.0alpha1",
     # ...
],
```

> **Tip**
>
> 를 사용하여 앱의 가상 환경에 이미 설치된 다른 버전의 AWS CDK 모듈을 모두 제거하세요. `pip uninstall` 그런 다음 를 사용하여 새 종속 항목을 설치합니다. `python -m pip install -r requirements.txt`

다음과 같이 앱 가져오기를 변경하세요.

- 새 `Construct` `constructs` 모듈에서 가져오기
- 코어 유형 (예: `App` 및`Stack`) 을 최상위 수준에서 가져옵니다. `aws_cdk`
- 아래의 네임스페이스에서 AWS 구조 라이브러리 모듈을 가져옵니다. `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

에서 `pom.xml` 안정적인 모듈의 모든 `software.amazon.awscdk` 종속성을 제거하고 `software.constructs` (for) 및 에 대한 종속성으로 바꿉니다. `Construct` `software.amazon.awscdk`

실험용 구문은 이름이 로 끝나고 알파 버전 번호가 있는 독립적으로 버전이 지정된 별도의 패키지로 제공됩니다. `alpha` 알파 버전 번호는 `aws-cdk-lib` 호환되는 첫 번째 릴리스에 해당합니다. 여기서는 v2.0.0-alpha.1에 고정했습니다`aws-codestar`.

```
<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>
```

를 실행하여 `mvn package` 새 종속성을 설치합니다.

코드를 변경하여 다음을 수행하세요.

- 새 `Construct` `software.constructs` 라이브러리에서 가져오기
- 에서 `Stack` 및 와 `App` 같은 핵심 클래스 가져오기 `software.amazon.awscdk`
- 에서 서비스 구문 가져오기 `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#

C# CDK 애플리케이션의 종속성을 업그레이드하는 가장 간단한 방법은 파일을 수동으로 편집하는 것입니다. `.csproj` 모든 안정적인 `Amazon.CDK.*` 패키지 참조를 제거하고 및 패키지에 대한 참조로 바꾸십시오. `Amazon.CDK.Lib` `Constructs`

실험용 구문은 이름이 로 끝나고 알파 버전 번호가 있는 독립적으로 버전이 지정된 별도의 패키지로 제공됩니다. `alpha` 알파 버전 번호는 `aws-cdk-lib` 호환되는 첫 번째 릴리스에 해당합니다. 여기서는 v2.0.0-alpha.1에 고정했습니다`aws-codestar`.

```
<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" />
```

를 실행하여 `dotnet restore` 새 종속성을 설치합니다.

소스 파일의 가져오기를 다음과 같이 변경하십시오.

```
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

종속성을 최신 버전으로 업데이트하고 프로젝트 `.mod` 파일을 go get 업데이트하라는 문제.

배포하기 전에 마이그레이션된 앱을 테스트하세요.

스택을 배포하기 전에 리소스에 예상치 못한 변경 사항이 있는지 확인하는 `cdk diff` 데 사용합니다. 논리적 ID 변경 (리소스 대체 원인) 은 예상되지 않습니다.

예상 변경 사항에는 다음이 포함되지만 이에 국한되지는 않습니다.

- `CDKMetadata`리소스 변경.
- 자산 해시가 업데이트되었습니다.
- 새로운 스타일의 스택 합성과 관련된 변경사항. 앱이 v1의 레거시 스택 신시사이저를 사용한 경우에 적용됩니다. (CDK v2는 레거시 스택 신시사이저를 지원하지 않습니다.)
- 규칙 추가. `CheckBootstrapVersion`

예상치 못한 변경은 일반적으로 AWS CDK v2로 업그레이드 자체로 인해 발생하지 않습니다. 일반적으로 이러한 오류는 이전에 기능 플래그로 인해 변경된 지원 중단된 동작의 결과입니다. 이는 약 1.85.x 이전 버전의 CDK에서 업그레이드할 때 나타나는 증상입니다. 최신 v1.x 릴리스로 업그레이드해도 동일한 변경 사항을 확인할 수 있습니다. 일반적으로 다음과 같이 하면 이 문제를 해결할 수 있습니다.

1. 앱을 최신 v1.x 릴리스로 업그레이드하세요.
2. 기능 플래그 제거
3. 필요에 따라 코드를 수정하세요.
4. Deploy

5. v2로 업그레이드

> **Note**
>
> 2단계 업그레이드 후 업그레이드된 앱을 배포할 수 없는 경우 문제를 신고하세요.

앱에 스택을 배포할 준비가 되면 테스트할 수 있도록 먼저 사본을 배포해 보세요. 가장 쉬운 방법은 다른 지역에 배포하는 것입니다. 하지만 스택의 ID를 변경할 수도 있습니다. 테스트 후에는 를 사용하여 테스트 사본을 삭제해야 합니다. cdk destroy

문제 해결

TypeScript **`'from' expected`**또는 임포트 중 **`';' expected`** 오류가 발생했습니다.

TypeScript 3.8 이상으로 업그레이드하십시오.

'cdk 부트스트랩'을 실행합니다.

다음과 같은 오류가 표시되는 경우

```
#  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 v2에는 업데이트된 부트스트랩 스택이 필요하며, 모든 v2 배포에는 부트스트랩 리소스가 필요합니다. (v1을 사용하면 부트스트래핑 없이 간단한 스택을 배포할 수 있습니다.) 자세한 내용은 the section called “부트스트래핑” 섹션을 참조하세요.

v1 스택 찾기

CDK 애플리케이션을 v1에서 v2로 마이그레이션할 때 v1을 사용하여 만든 배포된 AWS CloudFormation 스택을 확인하는 것이 좋습니다. 이렇게 하려면 다음 명령어를 실행하세요.

```
npx awscdk-v1-stack-finder
```

사용에 대한 자세한 내용은 awscdk-v1-stack-finder README를 참조하십시오.

기존 리소스 및 AWS CloudFormation 템플릿을 다음으로 마이그레이션하십시오. AWS CDK

CDK Migrate 기능은 프리뷰 릴리즈 AWS CDK 중이며 변경될 수 있습니다.

AWS Cloud Development Kit (AWS CDK) 명령줄 인터페이스 (AWS CDK CLI) 를 사용하여 배포된 AWS 리소스, 배포된 AWS CloudFormation 스택 및 로컬 AWS CloudFormation 템플릿을 로 마이그레이션할 수 있습니다. AWS CDK

주제

마이그레이션 작동 방식

AWS CDK CLI`cdk migrate`명령을 사용하여 다음 소스에서 마이그레이션하십시오.

- 배포된 AWS 리소스.
- 배포된 AWS CloudFormation 스택.
- 로컬 AWS CloudFormation 템플릿.

배포된 AWS 리소스

AWS CloudFormation 스택과 연결되지 않은 특정 환경 (AWS 계정 및 AWS 리전) 에서 배포된 AWS 리소스를 마이그레이션할 수 있습니다.

는 IaC 생성기 서비스를 AWS CDK CLI 활용하여 사용자 AWS 환경의 리소스를 스캔하여 리소스 세부 정보를 수집합니다. IaC generator에 대한 자세한 내용은 사용 설명서의 기존 리소스용 템플릿 생성을 참조하십시오.AWS CloudFormation

리소스 세부 정보를 수집한 후 는 마이그레이션된 리소스가 포함된 단일 스택을 포함하는 새 CDK 앱을 AWS CDK CLI 만듭니다.

배포된 스택 AWS CloudFormation

단일 AWS CloudFormation 스택을 새 AWS CDK 앱으로 마이그레이션할 수 있습니다. AWS CDK CLI그러면 스택의 AWS CloudFormation 템플릿을 검색하고 새 CDK 앱을 생성합니다. CDK 앱은 AWS CloudFormation 마이그레이션된 스택이 포함된 단일 스택으로 구성됩니다.

로컬 템플릿 AWS CloudFormation

로컬 AWS CloudFormation 템플릿에서 마이그레이션할 수 있습니다. 로컬 템플릿에는 배포된 리소스가 포함될 수도 있고 포함되지 않을 수도 있습니다. AWS CDK CLI그러면 리소스가 포함된 단일 스택이 포함된 새 CDK 앱이 생성됩니다.

마이그레이션 후에는 CDK 앱을 관리, 수정, 배포하여 리소스를 프로비저닝하거나 업데이트할 수 있습니다. AWS CloudFormation

CDK 마이그레이션의 이점

지금까지 리소스를 마이그레이션하는 AWS CDK 작업은 수동 프로세스였기 때문에 AWS CDK 시작까지 AWS CloudFormation 시간과 전문 지식이 필요합니다. CDK Migrate를 AWS CDK CLI 사용하면 짧은 시간 안에 마이그레이션 작업의 대부분을 수월하게 처리할 수 있습니다. CDK Migrate를 사용하면 새로운 애플리케이션과 기존 애플리케이션을 개발하고 관리하는 작업을 빠르게 시작할 수 있습니다. AWS CDK AWS

고려 사항

일반적인 고려 사항

CDK 마이그레이션과 CDK 임포트 비교

이 `cdk import` 명령은 배포된 리소스를 새 CDK 앱 또는 기존 CDK 앱으로 가져올 수 있습니다. 가져올 때 각 리소스를 앱에서 L1 구조로 수동으로 정의해야 합니다. 새 CDK 앱이나 기존 CDK 앱

으로 한 번에 하나 이상의 리소스를 가져오는 `cdk import` 데 사용하는 것이 좋습니다. 자세한 내용은 기존 리소스를 스택으로 가져오기 섹션을 참조하세요.

이 `cdk migrate` 명령은 배포된 리소스, 배포된 AWS CloudFormation 스택 또는 로컬 AWS CloudFormation 템플릿에서 새 CDK 앱으로 마이그레이션됩니다. `cdk import`마이그레이션하는 동안 사용자는 AWS CDK CLI 리소스를 새 CDK 앱으로 가져옵니다. AWS CDK CLI또한 각 리소스에 대한 L1 구문을 자동으로 생성합니다. 지원되는 마이그레이션 소스에서 새 앱으로 가져올 `cdk migrate` 때 사용하는 것이 좋습니다. AWS CDK

CDK Migrate는 L1 구문만 생성합니다.

새로 만든 CDK 앱에는 L1 구조만 포함됩니다. 마이그레이션 후 앱에 상위 수준 구문을 추가할 수 있습니다.

CDK Migrate는 단일 스택을 포함하는 CDK 앱을 만듭니다.

새로 만든 CDK 앱에는 단일 스택이 포함됩니다.

배포된 리소스를 마이그레이션할 때 마이그레이션된 모든 리소스는 새 CDK 앱의 단일 스택에 포함됩니다.

AWS CloudFormation 스택을 마이그레이션할 때 새 CDK 앱에서는 단일 스택만 단일 AWS CloudFormation 스택으로 마이그레이션할 수 있습니다.

에셋 마이그레이션

AWS Lambda 코드와 같은 프로젝트 에셋은 새 CDK 앱으로 직접 마이그레이션되지 않습니다. 마이그레이션 후에는 CDK 앱에 포함할 자산 값을 지정할 수 있습니다.

스테이트풀 리소스 마이그레이션

데이터베이스 및 Amazon Simple Storage Service (Amazon S3) 버킷과 같은 상태 저장 리소스를 마이그레이션할 때는 새 리소스를 생성하는 대신 기존 리소스를 마이그레이션하는 것이 가장 좋습니다.

스테이트풀 리소스를 마이그레이션하고 보존하려면 다음을 수행하십시오.

- 스테이트풀 리소스가 가져오기를 지원하는지 확인하세요. 자세한 내용은 AWS CloudFormation 사용 설명서의 리소스 유형 지원을 참조하십시오.
- 마이그레이션 후에는 새 CDK 앱에서 마이그레이션된 리소스의 논리 ID가 배포된 리소스의 논리 ID와 일치하는지 확인하십시오.
- AWS CloudFormation 스택에서 마이그레이션하는 경우 새 CDK 앱의 스택 이름이 스택과 일치하는지 확인하십시오. AWS CloudFormation

- 동일한 AWS 계정과 AWS 리전 마이그레이션된 리소스를 사용하여 CDK 앱을 배포하십시오.

템플릿에서 마이그레이션할 때 고려할 사항 AWS CloudFormation

CDK Migrate는 단일 템플릿 마이그레이션을 지원합니다.

AWS CloudFormation 템플릿을 마이그레이션할 때 마이그레이션할 템플릿 하나를 선택할 수 있습니다. 중첩 템플릿은 지원되지 않습니다.

내장 함수가 포함된 템플릿 마이그레이션

내장 함수를 사용하는 AWS CloudFormation 템플릿에서 마이그레이션할 때 는 클래스와 함께 논리를 AWS CDK CLI CDK 앱으로 마이그레이션하려고 시도합니다. Fn 자세히 알아보려면 API 레퍼런스의 클래스 Fn을 참조하십시오.AWS Cloud Development Kit (AWS CDK)

배포된 리소스에서 마이그레이션할 때 고려할 사항

스캔 제한

환경에서 리소스를 스캔할 때 IaC Generator에는 검색할 수 있는 데이터에 대한 특정 제한과 스캔 시 할당량 제한이 있습니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 고려 사항을 참조하십시오.

필수 조건

`cdk migrate`명령을 사용하기 전에 다음을 수행하십시오.

1. 를 사용하여 인증을 설정합니다 AWS. 지침은 2단계: 프로그래밍 방식 액세스 구성을 참조하세요.
2. AWS CDK CLI 설치 또는 업그레이드 설치 지침은 3단계: 설치 AWS CDKCLI을 확인하십시오.

CDK 마이그레이션 시작하기

시작하려면 원하는 디렉터리에서 AWS CDK CLI `cdk migrate` 명령을 실행하세요. 수행 중인 마이그레이션 유형에 따라 필수 및 선택적 옵션을 제공하십시오.

함께 `cdk migrate` 사용할 수 있는 옵션의 전체 목록 및 설명은 을 참조하십시오cdk migrate 명령 참조.

제공할 수 있는 몇 가지 중요한 옵션은 다음과 같습니다.

스택 이름

유일한 필수 옵션은 입니다`--stack-name`. 이 옵션을 사용하여 마이그레이션 후 AWS CDK 앱 내에 생성될 스택의 이름을 지정합니다. 스택 이름은 배포 시 AWS CloudFormation 스택 이름으로도 사용됩니다.

언어

새 CDK 앱의 프로그래밍 언어를 지정하는 `--language` 데 사용합니다.

AWS 계정 및 AWS 리전

는 기본 소스에서 AWS 계정 및 AWS 리전 정보를 AWS CDK CLI 검색합니다. 자세한 정보는 2단계: 프로그래밍 방식 액세스 구성을 참조하세요. `--account`와 `--region` `cdk migrate` 옵션을 사용하여 다른 값을 제공할 수 있습니다.

새 CDK 프로젝트의 출력 디렉터리

기본적으로 AWS CDK CLI 는 작업 디렉토리에 새 CDK 프로젝트를 만들고 입력한 값을 사용하여 프로젝트 폴더의 이름을 지정합니다. `--stack-name` 같은 이름의 폴더가 현재 존재하면 이 AWS CDK CLI 폴더를 덮어씁니다.

옵션을 사용하여 새 CDK 프로젝트 폴더에 다른 출력 경로를 지정할 수 있습니다. `--output-path`

마이그레이션 소스

마이그레이션할 소스를 지정하는 옵션을 제공하십시오.

- `--from-path`— 로컬 AWS CloudFormation 템플릿에서 마이그레이션합니다.
- `--from-scan`— AWS 계정 및 계정에 배포된 리소스에서 AWS 리전마이그레이션합니다.
- `--from-stack`— AWS CloudFormation 스택에서 마이그레이션합니다.

마이그레이션 원본에 따라 `cdk migrate` 명령을 사용자 지정하는 추가 옵션을 제공할 수 있습니다.

AWS CloudFormation 스택에서 마이그레이션하세요.

배포된 AWS CloudFormation 스택에서 마이그레이션하려면 `--from-stack` 옵션을 제공하십시오. 배포된 AWS CloudFormation 스택의 이름을 입력합니다`--stack-name`. 다음은 그 예제입니다.

```
$ cdk migrate --from-stack --stack-name "myCloudFormationStack"
```

AWS CDK CLI는 다음을 수행합니다.

1. 배포된 스택의 AWS CloudFormation 템플릿을 검색하십시오.
2. `cdk init`를 실행하여 새 CDK 앱을 초기화합니다.
3. 마이그레이션된 스택이 포함된 CDK 앱 내에 스택을 생성하십시오. AWS CloudFormation

배포된 AWS CloudFormation 스택에서 마이그레이션할 때 배포된 리소스 논리 ID와 배포된 AWS CloudFormation 스택 이름을 새 CDK 앱의 마이그레이션된 리소스 및 스택과 AWS CDK CLI 일치시키려고 시도합니다.

마이그레이션 후에는 CDK 앱을 정상적으로 관리하고 수정할 수 있습니다. 배포할 때 스택 이름이 AWS CloudFormation 일치하므로 배포를 AWS CloudFormation 스택 업데이트로 식별합니다. AWS CloudFormation 논리적 ID가 일치하는 리소스가 업데이트됩니다. 배포에 대한 자세한 내용은 을 참조하십시오CDK 앱 관리 및 배포.

템플릿에서 AWS CloudFormation 마이그레이션

CDK Migrate는 또는 형식의 AWS CloudFormation 템플릿에서 마이그레이션할 수 있도록 지원합니다. `JSON` `YAML`

로컬 AWS CloudFormation 템플릿에서 마이그레이션하려면 `--from-path` 옵션을 사용하고 로컬 템플릿의 경로를 제공하십시오. 또한 필수 `--stack-name` 옵션을 제공해야 합니다. 다음은 그 예제입니다.

```
$ cdk migrate --from-path "./template.json" --stack-name "myCloudFormationStack"
```

AWS CDK CLI다음 작업을 수행합니다.

1. 로컬 AWS CloudFormation 템플릿을 검색하세요.
2. `cdk init`를 실행하여 새 CDK 앱을 초기화합니다.
3. 마이그레이션된 템플릿이 포함된 CDK 앱 내에 스택을 생성합니다. AWS CloudFormation

마이그레이션 후에는 CDK 앱을 정상적으로 관리하고 수정할 수 있습니다. 배포 시 다음과 같은 옵션을 사용할 수 있습니다.

- AWS CloudFormation 스택 업데이트 - 로컬 AWS CloudFormation 템플릿이 이전에 배포된 경우 배포된 AWS CloudFormation 스택을 업데이트할 수 있습니다.
- 새 AWS CloudFormation 스택 배포 - 로컬 AWS CloudFormation 템플릿이 배포되지 않았거나 이전에 배포한 템플릿에서 새 스택을 생성하려는 경우 새 AWS CloudFormation 스택을 배포할 수 있습니다.

AWS SAM 템플릿에서 마이그레이션

AWS Serverless Application Model (AWS SAM) 템플릿에서 마이그레이션하려면 먼저 템플릿을 템플릿으로 변환하거나 배포하여 AWS CloudFormation 스택을 생성해야 합니다. AWS CloudFormation

AWS SAM 템플릿을 로 AWS CloudFormation변환하려면 AWS SAM CLI `sam validate --debug` 명령을 사용할 수 있습니다. 이 명령을 실행하기 전에 `false` `samconfig.toml` 파일에서 `lint` 로 설정해야 할 수도 있습니다.

AWS CloudFormation 스택으로 변환하려면 를 사용하여 AWS SAM 템플릿을 배포하십시오 AWS SAM CLI. 그런 다음 배포된 스택에서 마이그레이션하십시오.

배포된 리소스에서 마이그레이션하세요

배포된 AWS 리소스에서 마이그레이션하려면 `--from-scan` 옵션을 제공하십시오. 필수 `--stack-name` 옵션도 제공해야 합니다. 다음은 그 예제입니다.

```
$ cdk migrate --from-scan --stack-name "myCloudFormationStack"
```

AWS CDK CLI다음 작업을 수행합니다.

1. 계정에서 자원 및 재산 세부 정보를 검색하십시오. — IaC 생성기를 AWS CDK CLI 사용하여 계정을 스캔하고 세부 정보를 수집합니다.
2. AWS CloudFormation 템플릿 생성 — 스캔 후 IaC AWS CDK CLI 생성기를 사용하여 템플릿을 생성합니다. AWS CloudFormation
3. 새 CDK 앱 초기화 및 템플릿 이전 — AWS CDK CLI `cdk init` 실행하여 새 AWS CDK 앱을 초기화하고 AWS CloudFormation 템플릿을 단일 스택으로 CDK 앱에 마이그레이션합니다.

필터 사용

기본적으로 AWS CDK CLI 는 전체 AWS 환경을 스캔하고 리소스를 IaC Generator의 최대 할당량 한도까지 마이그레이션합니다. 계정에서 새 CDK 앱으로 리소스를 마이그레이션하는 기준을 지정하는 필터를 제공할 수 있습니다. AWS CDK CLI 자세한 내용은 `--filter` 섹션을 참조하세요.

IaC 생성기로 리소스 스캔하기

계정의 리소스 수에 따라 스캔하는 데 몇 분 정도 걸릴 수 있습니다. 스캔 프로세스 중에 진행률 표시줄이 표시됩니다.

지원되는 리소스 유형

AWS CDK CLIIaC 생성기가 지원하는 리소스를 마이그레이션합니다. 전체 목록은 AWS CloudFormation 사용 설명서의 리소스 유형 지원을 참조하십시오.

쓰기 전용 속성 해결

지원되는 일부 리소스에는 쓰기 전용 속성이 포함되어 있습니다. 이러한 속성을 작성하여 속성을 구성할 수는 있지만 IaC 생성기로 읽거나 AWS CloudFormation 값을 가져올 수는 없습니다. 예를 들어 데이터베이스 암호를 지정하는 데 사용되는 속성은 보안상의 이유로 쓰기 전용일 수 있습니다.

마이그레이션 중에 리소스를 스캔할 때 IaC generator는 쓰기 전용 속성을 포함할 수 있는 리소스를 탐지하여 다음 유형 중 하나로 분류합니다.

- `MUTUALLY_EXCLUSIVE_PROPERTIES`— 이는 상호 교환이 가능하고 유사한 용도로 사용되는 특정 리소스의 쓰기 전용 속성입니다. 리소스를 구성하려면 상호 배타적인 속성 중 하나가 필요합니다. 예를 들어, `AWS::Lambda::Function` 리소스의 `S3BucketImageUri`, 및 `ZipFile` 속성은 상호 배타적인 쓰기 전용 속성입니다. 둘 중 하나를 사용하여 함수 자산을 지정할 수 있지만 반드시 하나를 사용해야 합니다.
- `MUTUALLY_EXCLUSIVE_TYPES`— 여러 구성 유형을 허용하는 필수 쓰기 전용 속성입니다. 예를 들어, `AWS::ApiGateway::RestApi` 리소스의 `Body` 속성은 객체 또는 문자열 유형을 받아들입니다.
- `UNSUPPORTED_PROPERTIES`— 다른 두 범주에 속하지 않는 쓰기 전용 속성입니다. 이들은 선택적 속성이거나 개체 배열을 허용하는 필수 속성입니다.

쓰기 전용 속성과 IaC Generator가 배포된 리소스를 검색하고 AWS CloudFormation 템플릿을 만들 때 쓰기 전용 속성을 관리하는 방법에 대한 자세한 내용은 사용 설명서의 IaC 생성기 및 쓰기 전용 속성을 참조하십시오.AWS CloudFormation

마이그레이션 후에는 새 CDK 앱에서 쓰기 전용 속성 값을 지정해야 합니다. AWS CDK CLI는 CDK 프로젝트 ReadMe 파일에 경고 섹션을 추가하여 IaC 생성기로 식별된 쓰기 전용 속성을 문서화합니다. 다음은 그 예제입니다.

```
# 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
```

- 경고는 관련 리소스의 논리적 ID를 식별하는 제목 아래에 정리되어 있습니다.
- 경고는 유형별로 분류됩니다. 이러한 유형은 IaC 생성기에서 직접 가져온 것입니다.

쓰기 전용 속성을 해결하려면

1. CDK 프로젝트 파일의 경고 섹션에서 해결해야 할 쓰기 전용 속성을 식별하십시오. ReadMe 여기에서 쓰기 전용 속성을 포함할 수 있는 CDK 앱의 리소스를 기록하고 발견된 쓰기 전용 속성 유형을 식별할 수 있습니다.

a. 의 경우MUTUALLY_EXCLUSIVE_PROPERTIES, 앱에서 어떤 상호 배타적 속성을 구성할지 결정하세요. AWS CDK

b. 에 대해 MUTUALLY_EXCLUSIVE_TYPES 속성을 구성하는 데 사용할 허용 유형을 결정하십시오.

c. 의 UNSUPPORTED_PROPERTIES 경우 속성이 선택사항인지 필수인지 확인하십시오. 그런 다음 필요에 따라 구성합니다.

2. IaC 생성기 및 쓰기 전용 속성의 지침을 사용하여 경고 유형의 의미를 참조할 수 있습니다.

3. CDK 앱에서는 해결해야 할 쓰기 전용 속성 값도 앱 섹션에 지정됩니다. Props 여기에 올바른 값을 입력하세요. 속성 설명 및 지침은 AWS CDK API 참조를 참조할 수 있습니다.

   다음은 해결해야 할 쓰기 전용 속성 2개가 있는 마이그레이션된 CDK 앱 내 Props 섹션 예시입니다.

```
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;
}
```

쓰기 전용 속성 값을 모두 확인했으면 배포를 준비할 준비가 된 것입니다.

마이그레이션.json 파일

는 AWS CDK CLI 마이그레이션 중에 프로젝트에 migrate.json 파일을 생성합니다. AWS CDK 이 파일에는 배포된 리소스에 대한 참조 정보가 들어 있습니다. CDK 앱을 처음 배포하면 AWS CDK CLI는 이 파일을 사용하여 배포된 리소스를 참조하고, 리소스를 새 AWS CloudFormation 스택과 연결하고, 파일을 삭제합니다.

CDK 앱 관리 및 배포

로 마이그레이션할 AWS CDK때 새 CDK 앱을 즉시 배포할 준비가 되지 않을 수 있습니다. 이 주제에서는 새 CDK 앱을 관리하고 배포할 때 고려해야 할 조치 항목에 대해 설명합니다.

배포 준비

배포하기 전에 CDK 앱을 준비해야 합니다.

앱 합성하기

`cdk synth`명령어를 사용하여 CDK 앱의 스택을 템플릿으로 합성합니다. AWS CloudFormation

배포된 AWS CloudFormation 스택 또는 템플릿에서 마이그레이션한 경우 합성된 템플릿을 마이그레이션된 템플릿과 비교하여 리소스 및 속성 값을 확인할 수 있습니다.

`cdk synth`에 대한 자세한 내용은 스택 합성을(를) 참조하세요.

비교 수행

배포된 AWS CloudFormation 스택에서 마이그레이션한 경우 cdk diff 명령어를 사용하여 새 CDK 앱의 스택과 비교할 수 있습니다.

cdk diff에 대한 자세한 내용은 을 참조하십시오. 스택 비교

환경을 부트스트랩하세요

환경에서 처음 배포하는 경우 AWS 환경을 준비하는 `cdk bootstrap` 데 사용하십시오. 자세한 내용은 부트스트래핑 섹션을 참조하세요.

CDK 앱 배포

CDK 앱을 배포하면 에서 AWS CloudFormation 서비스를 AWS CDK CLI 활용하여 리소스를 프로비저닝합니다. 리소스는 CDK 앱에서 단일 스택으로 번들링되며 단일 스택으로 배포됩니다. AWS CloudFormation

마이그레이션한 위치에 따라 배포하여 새 AWS CloudFormation 스택을 만들거나 기존 스택을 업데이트할 수 있습니다. AWS CloudFormation

배포하여 새 AWS CloudFormation 스택을 생성하십시오.

배포된 리소스에서 마이그레이션한 경우 배포 시 에서 AWS CDK CLI 자동으로 새 AWS CloudFormation 스택을 생성합니다. 배포된 리소스는 새 AWS CloudFormation 스택에 포함됩니다.

배포되지 않은 로컬 AWS CloudFormation 템플릿에서 마이그레이션한 경우 배포 시 새 AWS CloudFormation 스택이 AWS CDK CLI 자동으로 생성됩니다.

배포된 AWS CloudFormation 스택 또는 이전에 배포된 로컬 AWS CloudFormation 템플릿에서 마이그레이션한 경우 배포하여 새 AWS CloudFormation 스택을 만들 수 있습니다. 새 스택을 생성하려면 다음과 같이 하십시오.

- 새 AWS 환경에 배포하세요. 이는 다른 계정을 사용하거나 다른 AWS 계정에 배포하는 것으로 구성됩니다. AWS 리전
- 마이그레이션된 스택 또는 템플릿의 동일한 AWS 환경에 새 스택을 배포하려면 CDK 앱의 스택 이름을 새 값으로 수정해야 합니다. 또한 CDK 앱 리소스의 모든 논리적 ID를 수정해야 합니다. 그런 다음 동일한 환경에 배포하여 새 스택과 새 리소스를 만들 수 있습니다.

배포하여 기존 AWS CloudFormation 스택을 업데이트하십시오.

배포된 AWS CloudFormation 스택 또는 이전에 배포된 로컬 AWS CloudFormation 템플릿에서 마이그레이션한 경우 배포하여 기존 AWS CloudFormation 스택을 업데이트할 수 있습니다.

CDK 앱의 스택 이름이 AWS CloudFormation 배포된 스택의 스택 이름과 일치하는지 확인하고 동일한 AWS 환경에 배포하십시오.

지원되는 프로그래밍 언어로 작업하기 AWS CDK

AWS Cloud Development Kit (AWS CDK) 를 사용하여 지원되는 프로그래밍 언어로 AWS 클라우드 인프라를 정의할 수 있습니다.

주제

AWS 구성 라이브러리 가져오기

AWS CDK 여기에는 서비스별로 AWS 구성된 AWS 구성 컬렉션인 구성 라이브러리가 포함됩니다. 라이브러리의 안정적인 구조는 TypeScript 패키지 이름으로 호출되는 단일 모듈로 제공됩니다. `aws-cdk-lib` 실제 패키지 이름은 언어마다 다릅니다.

TypeScript

Install	`npm ## aws-cdk-lib`
가져오기	`const cdk = ## (''); aws-cdk-lib`

JavaScript

Install	`npm ## aws-cdk-lib`
가져오기	`const cdk = ## (''); aws-cdk-lib`

Python

Install	`### -m pip ## aws-cdk-lib`
가져오기	`aws_cdk# cdk# ####`

Java

다음에 추가 `pom.xml`	Group `#####.amazon.awscdk` ; artifact `aws-cdk-lib`
가져오기	`##### ####. Amazon.awscdk.app;` (for example)

C#

Install	`## ## ### Amazon.cdk.lib`
가져오기	`Amazon.cdk ##`

`construct`기본 클래스와 지원 코드는 모듈에 있습니다. `constructs` API가 아직 개선 중인 실험적 구문은 별도의 모듈로 배포됩니다.

API 레퍼런스 AWS CDK

AWS CDK API 참조는 라이브러리의 구성 (및 기타 구성 요소) 에 대한 자세한 문서를 제공합니다. 지원되는 각 프로그래밍 언어에 대한 API 참조 버전이 제공됩니다.

각 모듈의 참조 자료는 다음 섹션으로 구분되어 있습니다.

- 개요: 개념과 예를 포함하여 에서 서비스를 사용하기 위해 알아야 할 소개 자료입니다. AWS CDK
- 구성: 하나 이상의 구체적인 AWS 리소스를 나타내는 라이브러리 클래스. 기본값이 정상인 고급 인터페이스를 제공하는 '큐레이션된' (L2) 리소스 또는 패턴 (L3 리소스) 입니다.
- 클래스: 모듈의 구문에 사용되는 기능을 제공하는 비구체 클래스입니다.
- 구조체: 속성 (구문의 `props` 인수) 및 옵션과 같은 복합 값의 구조를 정의하는 데이터 구조 (속성 번들).

- 인터페이스: 이름이 모두 “I”로 시작하는 인터페이스는 해당 구문이나 다른 클래스의 절대 최소 기능을 정의합니다. CDK는 구문 인터페이스를 사용하여 AWS CDK 앱 외부에서 정의되고 다음과 같은 메서드에서 참조하는 AWS 리소스를 나타냅니다. `Bucket.fromBucketArn()`
- 열거형: 특정 구성 파라미터를 지정하는 데 사용되는 명명된 값의 컬렉션입니다. 열거된 값을 사용하면 CDK가 합성 중에 해당 값의 유효성을 검사할 수 있습니다.
- CloudFormation 리소스: 이름이 “Cfn”으로 시작하는 이러한 L1 구문은 사양에 정의된 리소스를 정확히 나타냅니다. CloudFormation CDK가 릴리스될 때마다 해당 사양에서 자동으로 생성됩니다. 각 L2 또는 L3 구조는 하나 이상의 리소스를 캡슐화합니다. CloudFormation
- CloudFormation 속성 유형: 각 리소스의 속성을 정의하는 명명된 값의 모음입니다. CloudFormation

인터페이스와 구성 클래스 비교

AWS CDK 는 인터페이스를 프로그래밍 개념으로 잘 알고 있더라도 명확하지 않을 수 있는 특정한 방식으로 인터페이스를 사용합니다.

는 다음과 같은 `Bucket.fromBucketArn()` 방법을 사용하여 CDK 애플리케이션 외부에 정의된 리소스를 사용할 수 있도록 AWS CDK 지원합니다. 외부 리소스는 수정할 수 없으며 클래스를 사용하여 CDK 앱에 정의된 리소스에서 사용할 수 있는 모든 기능을 사용하지 못할 수도 있습니다. `Bucket` 따라서 인터페이스는 외부 리소스를 포함하여 특정 AWS 리소스 유형에 대해 CDK에서 사용할 수 있는 최소한의 기능을 나타냅니다.

CDK 앱에서 리소스를 인스턴스화할 때는 항상 다음과 같은 구체적인 클래스를 사용해야 합니다. `Bucket` 자체 구문 중 하나에 허용되는 인수 유형을 지정할 때는 인터페이스 유형을 사용하세요. 예를 들어 외부 리소스를 처리할 준비가 되어 `IBucket` 있다면 (즉, 변경할 필요가 없는 경우). CDK로 정의한 구문이 필요한 경우 사용할 수 있는 가장 일반적인 유형을 지정하십시오.

일부 인터페이스는 구문이 아니라 특정 클래스와 관련된 속성 또는 옵션 번들의 최소 버전입니다. 이러한 인터페이스는 부모 클래스로 전달할 인수를 서브클래싱하여 받아들일 때 유용할 수 있습니다. 하나 이상의 추가 속성이 필요한 경우 이 인터페이스 또는 좀 더 구체적인 유형에서 속성을 구현하거나 파생시키는 것이 좋습니다.

> **Note**
>
> 에서 지원하는 일부 프로그래밍 언어에는 인터페이스 기능이 AWS CDK 없습니다. 이러한 언어에서 인터페이스는 일반 클래스일 뿐입니다. 이름으로 구분할 수 있습니다. 이름은 이니셜 “I”의 패턴을 따르고 그 뒤에 다른 구문의 이름 (예:`IBucket`) 이 옵니다. 동일한 규칙이 적용됩니다.

종속성 관리

AWS CDK 앱 또는 라이브러리의 종속성은 패키지 관리 도구를 사용하여 관리합니다. 이러한 도구는 일반적으로 프로그래밍 언어와 함께 사용됩니다.

일반적으로 는 해당 언어의 표준 또는 공식 패키지 관리 도구 (있는 경우) 를 AWS CDK 지원합니다. 그렇지 AWS CDK 않으면 에서 해당 언어의 가장 인기 있거나 널리 지원되는 언어를 지원하게 됩니다. 다른 도구를 사용할 수도 있습니다. 특히 지원되는 도구와 함께 작동하는 경우에는 더욱 그렇습니다. 하지만 다른 도구에 대한 공식 지원은 제한적입니다.

AWS CDK 는 다음 패키지 관리자를 지원합니다.

언어	지원되는 패키지 관리 도구
TypeScript/JavaScript	NPM (노드 패키지 관리자) 또는 원사
Python	PIP (파이썬용 패키지 인스톨러)
Java	Maven
C#	NuGet
Go	Go 모듈

AWS CDK CLI`cdk init`명령을 사용하여 새 프로젝트를 만들면 CDK 코어 라이브러리 및 안정적인 구문에 대한 종속성이 자동으로 지정됩니다.

지원되는 프로그래밍 언어의 종속성 관리에 대한 자세한 내용은 다음을 참조하십시오.

- 의 종속성 관리 TypeScript.
- 의 종속성 관리 JavaScript.
- 에서 종속성 관리 Python.
- 의 종속성 관리 Java.
- 에서 종속성 관리 C#.
- 종속성 관리: Go.

다른 AWS CDKTypeScript 언어와의 비교

TypeScript AWS CDK 애플리케이션 개발에 지원되는 첫 번째 언어였습니다. 따라서 상당한 양의 예제 CDK 코드가 작성됩니다. TypeScript 다른 언어로 개발하는 경우 선택한 언어와 AWS CDK 코드 구현 방식을 비교하는 것이 유용할 수 있습니다. TypeScript 이렇게 하면 문서 전체에서 예제를 사용하는 데 도움이 될 수 있습니다.

모듈 가져오기

TypeScript/JavaScript

TypeScript 전체 네임스페이스 또는 네임스페이스에서 개별 객체 가져오기를 지원합니다. 각 네임스페이스에는 지정된 서비스에 사용할 수 있는 구문 및 기타 클래스가 포함되어 있습니다. AWS

```
// Import main CDK library as cdk
import * as cdk from 'aws-cdk-lib';   // ES6 import preferred in TS
const cdk = require('aws-cdk-lib');   // Node.js require() preferred in JS

// Import specific core CDK classes
import { Stack, App } from 'aws-cdk-lib';
const { Stack, App } = require('aws-cdk-lib');

// Import AWS S3 namespace as s3 into current namespace
import { aws_s3 as s3 } from 'aws-cdk-lib';   // TypeScript
const s3 = require('aws-cdk-lib/aws-s3');     // JavaScript

// Having imported cdk already as above, this is also valid
const s3 = cdk.aws_s3;

// Now use s3 to access the S3 types
const bucket = s3.Bucket(...);

// Selective import of s3.Bucket
import { Bucket } from 'aws-cdk-lib/aws-s3';        // TypeScript
const { Bucket } = require('aws-cdk-lib/aws-s3');   // JavaScript

// Now use Bucket to instantiate an S3 bucket
const bucket = Bucket(...);
```

Python

마찬가지로 TypeScript, Python은 네임스페이스 모듈 가져오기와 선택적 가져오기를 지원합니다. 파이썬의 네임스페이스는 aws_cdk와 비슷합니다. xxx, 여기서 xxx는 AWS 서비스 이름을 나타냅니다 (예: Amazon S3의 경우 s3). (이 예제에서는 Amazon S3가 사용됩니다.)

```
# 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

Java의 가져오기는 TypeScript 's와 다르게 작동합니다. 각 import 문은 지정된 패키지에서 단일 클래스 이름을 가져오거나 해당 패키지에 정의된 모든 클래스 (사용*) 를 가져옵니다. 클래스는 가져온 경우 클래스 이름을 단독으로 사용하거나 패키지를 포함한 정규화된 클래스 이름을 사용하여 액세스할 수 있습니다.

라이브러리 이름은 AWS 구성 라이브러리의 이름을 따서 명명되었습니다 (기본 라이브러리는 다음과 같습니다 `software.amazon.awscdk.services.xxx`software.amazon.awscdk). AWS CDK 패키지의 Maven 그룹 ID는 입니다`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#

C #에서는 지시문을 사용하여 형식을 가져옵니다. `using` 두 가지 스타일이 있습니다. 하나는 일반 이름을 사용하여 지정된 네임스페이스의 모든 유형에 액세스할 수 있도록 합니다. 다른 방법으로는 별칭을 사용하여 네임스페이스 자체를 참조할 수 있습니다.

패키지 이름은 AWS Construct Library 패키지에서 `Amazon.CDK.AWS.xxx` 따온 것과 같습니다. (핵심 모듈은 입니다`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

각 AWS 구조 라이브러리 모듈은 Go 패키지로 제공됩니다.

```
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(...)
```

구문 인스턴스화

AWS CDK 구조체 클래스는 지원되는 모든 언어에서 같은 이름을 가집니다. 대부분의 언어는 new 키워드를 사용하여 클래스를 인스턴스화합니다 (Python과 Go는 그렇지 않음). 또한 대부분의 언어에서 키워드는 현재 this 인스턴스를 가리킵니다. (Python은 self 관례에 따라 사용합니다.) 현재 인스턴스에 대한 참조를 생성하는 모든 구문의 scope 매개 변수로 전달해야 합니다.

AWS CDK 구문의 세 번째 인수는 구문을 만드는 데 필요한 속성이 들어 있는 객체입니다. props 이 인수는 선택사항일 수 있지만 필요한 경우 지원되는 언어가 관용적인 방식으로 처리합니다. 속성 이름도 언어의 표준 이름 지정 패턴에 맞게 조정됩니다.

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

Python은 클래스를 인스턴스화할 때 new 키워드를 사용하지 않습니다. properties 인수는 키워드 인수를 사용하여 표현되고 인수의 이름은 를 사용하여 지정됩니다. `snake_case`

props 값 자체가 속성 묶음인 경우 해당 속성의 이름을 딴 클래스로 표현되며, 이 클래스는 하위 속성에 대한 키워드 인수를 허용합니다.

Python에서 현재 인스턴스는 규칙에 따라 이름이 지정된 첫 번째 인수로 메서드에 `self` 전달됩니다.

```
# 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

자바에서 props 인수는 이름이 지정된 클래스로 표현됩니다 `XxxxProps` (예: `Bucket` 구문의 props의 `BucketProps` 경우). 빌더 패턴을 사용하여 props 인수를 빌드합니다.

각 XxxxProps 클래스에는 빌더가 있습니다. 다음 예제와 같이 각 구문에는 props와 구문을 한 번에 빌드하는 편리한 빌더도 있습니다.

소품의 이름은 in을 사용하여 지정하는 것과 TypeScript 같습니다. camelCase

```
// Instantiate default Bucket
Bucket bucket = Bucket(self, "MyBucket");

// Instantiate Bucket with bucketName and versioned properties
Bucket bucket = Bucket.Builder.create(self, "MyBucket")
                      .bucketName("my-bucket").versioned(true)
                      .build();

# Instantiate Bucket with websiteRedirect, which has its own sub-properties
Bucket bucket = Bucket.Builder.create(self, "MyBucket")
                      .websiteRedirect(new websiteRedirect.Builder()
                          .hostName("aws.amazon.com").build())
                      .build();
```

C#

C #에서 props는 이름이 지정된 클래스 XxxxProps (예: 구문의 props) 에 객체 이니셜라이저를 사용하여 지정합니다. BucketProps Bucket

Props의 이름은 를 사용한다는 점을 제외하면 과 비슷합니다. TypeScript PascalCase

구문을 인스턴스화할 때 var 키워드를 사용하면 편리하므로 클래스 이름을 두 번 입력할 필요가 없습니다. 하지만 로컬 코드 스타일 가이드는 다를 수 있습니다.

```
// 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

Go에서 구문을 만들려면 함수를 호출해야 합니다. `NewXxxxxx` 여기서 `Xxxxxxx` 는 구문의 이름입니다. 구문의 속성은 구조체로 정의됩니다.

Go에서 모든 구문 매개변수는 숫자, 불리언, 문자열과 같은 값을 포함한 포인터입니다. 와 같은 편의 함수를 사용하여 이러한 `jsii.String` 포인터를 생성하세요.

```
// 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"),
 }})
```

멤버에 접근하기

일반적으로 구문 및 기타 AWS CDK 클래스의 속성이나 속성을 참조하고 이러한 값을 입력으로 사용하여 다른 구문을 빌드할 수 있습니다. 앞서 설명한 메서드 이름 지정 차이는 여기에도 적용됩니다. 또한 Java에서는 멤버에 직접 액세스할 수 없습니다. 대신 getter 메서드가 제공됩니다.

TypeScript/JavaScript

이름은 다음과 같습니다. `camelCase`

```
bucket.bucketArn
```

Python

이름은`snake_case`.

```
bucket.bucket_arn
```

Java

각 속성에 대해 getter 메서드가 제공됩니다. 이러한 이름은 다음과 같습니다`camelCase`.

```
bucket.getBucketArn()
```

C#

이름은 입니다. `PascalCase`

```
bucket.BucketArn
```

Go

이름은`PascalCase`.

```
bucket.BucketArn
```

열거형 상수

열거형 상수는 클래스로 범위가 지정되며, 모든 언어에서 밑줄이 있는 대문자 이름을 가집니다 (라고도 함). `SCREAMING_SNAKE_CASE` Go를 제외한 지원되는 모든 언어에서 클래스 이름에도 동일한 대/소문자가 사용되므로 이들 언어에서는 정규화된 열거형 이름도 동일합니다.

```
s3.BucketEncryption.KMS_MANAGED
```

Go에서 열거형 상수는 모듈 네임스페이스의 속성이며 다음과 같이 작성됩니다.

```
awss3.BucketEncryption_KMS_MANAGED
```

오브젝트 인터페이스

AWS CDK 는 TypeScript 개체 인터페이스를 사용하여 클래스가 예상 메서드 및 속성 집합을 구현함을 나타냅니다. 이름이 로 `I` 시작되므로 객체 인터페이스를 인식할 수 있습니다. 구체적인 클래스는 `implements` 키워드를 사용하여 구현하는 인터페이스를 나타냅니다.

TypeScript/JavaScript

> **Note**
>
> JavaScript 인터페이스 기능이 없습니다. `implements`키워드와 그 뒤에 오는 클래스 이름은 무시해도 됩니다.

```
import { IAspect, IConstruct } from 'aws-cdk-lib';

class MyAspect implements IAspect {
  public visit(node: IConstruct) {
    console.log('Visited', node.node.path);
  }
}
```

Python

Python에는 인터페이스 기능이 없습니다. 하지만 클래스를 로 AWS CDK 꾸미면 인터페이스 구현을 표시할 수 `@jsii.implements(interface)` 있습니다.

```
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 구조체는 구현하는 인터페이스를 명시적으로 선언할 필요가 없습니다. Go 컴파일러는 구조체에서 사용할 수 있는 메서드와 속성을 기반으로 구현을 결정합니다. 예를 들어 다음 코드에서는 구문을 취하는 `Visit` 메서드를 제공하기 때문에 이 `IAspect` 인터페이스를 `MyAspect` 구현합니다.

```
type MyAspect struct {
}

func (a MyAspect) Visit(node constructs.IConstruct) {
 fmt.Println("Visited", *node.Node().Path())
}
```

In과 함께 AWS CDK 일하기 TypeScript

TypeScript 에서 완전히 지원되는 클라이언트 언어이며 안정적인 것으로 AWS Cloud Development Kit (AWS CDK) 간주됩니다. 을 AWS CDK TypeScript 사용하여 작업하면 Microsoft의 TypeScript 컴파일러 (`tsc`), Node.js 및 Node Package Manager (`npm`) 와 같은 친숙한 도구가 사용됩니다. 이 가이드의 예제에서는 NPM을 사용하지만 원하는 경우 Yarn을 사용할 수도 있습니다. AWS 구성 라이브러리를 구성하는 모듈은 NPM 저장소인 npmjs.org를 통해 배포됩니다.

모든 에디터나 IDE를 사용할 수 있습니다. 많은 AWS CDK 개발자가 뛰어난 지원을 제공하는 Visual Studio Code (또는 이에 상응하는 오픈 소스 VScodium) 를 사용합니다. TypeScript

주제

- 다음으로 시작해 보세요. TypeScript
- 프로젝트 생성

- 로컬 tsc 및 cdk
- 컨스트럭트 라이브러리 모듈 관리 AWS
- 의 종속성 관리 TypeScript
- AWS CDK 의 관용구 TypeScript
- 빌드, 합성 및 배포

다음으로 시작해 보세요. TypeScript

를 사용하려면 AWS 계정과 자격 증명이 있어야 하며 Node.js 및 AWS CDK 툴킷이 설치되어 있어야 합니다. AWS CDK시작하기 AWS CDK 섹션을 참조하십시오.

또한 TypeScript 자체 버전 (버전 3.8 이상) 이 필요합니다. 아직 설치하지 않은 경우 를 사용하여 npm 설치할 수 있습니다.

```
npm install -g typescript
```

> **Note**
>
> 권한 오류가 발생하여 시스템에 대한 관리자 액세스 권한이 있는 경우 시도해 보세요`sudo npm install -g typescript`.

정기적으로 최신 소식을 받아보세요`npm update -g typescript`. TypeScript

> **Note**
>
> 타사 언어 지원 중단: 언어 버전은 공급업체 또는 커뮤니티에서 EOL (End Of Life) 을 공유할 때까지만 지원되며 사전 통지를 통해 변경될 수 있습니다.

프로젝트 생성

빈 `cdk init` 디렉터리에서 호출하여 새 AWS CDK 프로젝트를 생성합니다. `--language`옵션을 사용하고 다음을 지정합니다`typescript`.

```
mkdir my-project
cd my-project
```

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

프로젝트를 생성하면 aws-cdk-lib모듈과 해당 종속 항목도 설치됩니다.

cdk init프로젝트 폴더 이름을 사용하여 클래스, 하위 폴더, 파일 등 프로젝트의 다양한 요소에 이름을 지정합니다. 폴더 이름의 하이픈은 밑줄로 변환됩니다. 하지만 이름은 TypeScript 식별자 형식을 따라야 합니다. 예를 들어 숫자로 시작하거나 공백을 포함해서는 안 됩니다.

로컬 tsc 및 cdk

대부분의 경우 이 가이드에서는 CDK 툴킷을 전역적으로 설치 () TypeScript 하고, 제공된 명령 예제 (예:npm install -g typescript aws-cdk) 는 이 가정을 따르는 것으로 cdk synth 가정합니다. 이 방법을 사용하면 두 구성 요소를 모두 최신 상태로 쉽게 유지할 수 있으며, 두 구성 요소 모두 이전 버전과의 호환성에 대한 엄격한 접근 방식을 취하므로 일반적으로 항상 최신 버전을 사용해도 위험은 거의 없습니다.

일부 팀은 TypeScript 컴파일러와 CDK Toolkit과 같은 도구를 포함하여 각 프로젝트 내의 모든 종속성을 지정하는 것을 선호합니다. 이렇게 하면 이러한 구성 요소를 특정 버전에 고정하고 팀의 모든 개발자 (및 CI/CD 환경) 가 정확히 해당 버전을 사용하도록 할 수 있습니다. 이렇게 하면 변경 가능성이 제거되므로 빌드와 배포의 일관성과 반복성을 높일 수 있습니다.

CDK에는 TypeScript 프로젝트 템플릿의 package.json CDK TypeScript 툴킷과 둘 다에 대한 종속성이 포함되어 있으므로 이 접근 방식을 사용하려는 경우 프로젝트를 변경할 필요가 없습니다. 앱을 빌드할 때와 명령을 실행할 때 약간 다른 명령을 사용하기만 하면 됩니다. cdk

Operation	글로벌 도구를 사용하세요.	로컬 도구 사용
프로젝트 초기화	`cdk init --language typescript`	`npx aws-cdk init --language typescript`
빌드	`tsc`	`npm run build`
CDK 툴킷 명령 실행	`cdk ...`	`npm run cdk ...` or `npx aws-cdk ...`

npx aws-cdk현재 프로젝트에 로컬로 설치된 CDK Toolkit 버전이 있는 경우 이를 실행하고, 글로벌 설치 (있는 경우) 로 폴백합니다. 글로벌 설치가 없는 경우 CDK Toolkit의 임시 사본을 npx 다운로드하여 실행합니다. : prints 구문을 사용하여 CDK 툴킷의 임의 버전을 지정할 수 있습니다. @ npx aws-cdk@2.0 --version 2.0.0

> **Tip**
>
> 로컬 CDK 툴킷 설치에서 cdk 명령을 사용할 수 있도록 별칭을 설정합니다.
>
> macOS/Linux
>
> ```
> alias cdk="npx aws-cdk"
> ```
>
> Windows
>
> ```
> doskey cdk=npx aws-cdk $*
> ```

컨스트럭트 라이브러리 모듈 관리 AWS

Node Package Manager (npm) 를 사용하면 필요한 다른 패키지뿐 아니라 앱에서 사용할 수 있는 AWS Construct Library 모듈을 설치하고 업데이트할 수 있습니다. (원하는 npm 경우 yarn 대신 사용할 수도 있습니다.) npm또한 해당 모듈의 종속성을 자동으로 설치합니다.

대부분의 AWS CDK 구문은 라는 이름의 aws-cdk-lib 기본 CDK 패키지에 들어 있습니다. 이 패키지는 에서 만든 새 프로젝트의 기본 종속 항목입니다. cdk init 상위 레벨 AWS 구문이 아직 개발 중인 "실험적" 구조 라이브러리 모듈의 이름은 다음과 같습니다. @aws-cdk/*SERVICE-NAME*-alpha 서비스 이름에는 aws - 접두사가 붙습니다. 모듈 이름이 확실하지 않은 경우 NPM에서 검색하세요.

> **Note**
>
> CDK API 레퍼런스에는 패키지 이름도 나와 있습니다.

예를 들어, 아래 명령은 에 대한 실험용 모듈을 설치합니다. AWS CodeStar

```
npm install @aws-cdk/aws-codestar-alpha
```

일부 서비스의 구성 라이브러리 지원은 둘 이상의 네임스페이스에서 제공됩니다. 예를 들어aws-route53, 그 외에도 Amazon Route 53 네임스페이스,, aws-route53-targetsaws-route53-patterns, 가 세 개 더 있습니다. aws-route53resolver

프로젝트의 종속성은 에서 유지 관리됩니다. package.json 이 파일을 편집하여 일부 또는 모든 종속성을 특정 버전에 고정하거나 특정 기준에 따라 새 버전으로 업데이트하도록 허용할 수 있습

니다. 에서 지정한 규칙에 따라 프로젝트의 NPM 종속성을 허용된 최신 버전으로 업데이트하려면: `package.json`

```
npm update
```

TypeScript에서는 NPM을 사용하여 모듈을 설치할 때 사용하는 것과 동일한 이름으로 모듈을 코드에 가져옵니다. 애플리케이션에서 AWS CDK 클래스와 AWS Construct Library 모듈을 가져올 때는 다음 방법을 따르는 것이 좋습니다. 이 가이드라인을 따르면 코드를 다른 AWS CDK 응용 프로그램과 일관되게 작성할 수 있을 뿐만 아니라 코드를 더 쉽게 이해할 수 있습니다.

- ES6 스타일 `import` 디렉티브를 사용하고, 사용하지 마십시오. `require()`
- 일반적으로 에서 개별 클래스를 가져옵니다. `aws-cdk-lib`

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

- 에서 많은 클래스가 필요한 경우 `aws-cdk-lib` 개별 클래스를 가져오는 `cdk` 대신 의 네임스페이스 별칭을 사용할 수 있습니다. 두 가지를 모두 수행하지 마세요.

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

- 일반적으로 짧은 네임스페이스 별칭을 사용하여 AWS 서비스 구조를 가져옵니다.

  ```
  import { aws_s3 as s3 } from 'aws-cdk-lib';
  ```

의 종속성 관리 TypeScript

TypeScriptCDK 프로젝트에서 종속성은 프로젝트의 기본 디렉터리에 있는 `package.json` 파일에 지정됩니다. 핵심 AWS CDK 모듈은 라는 단일 NPM 패키지에 들어 있습니다. `aws-cdk-lib`

를 사용하여 npm install 패키지를 설치하면 NPM이 패키지를 자동으로 기록합니다. `package.json`

원하는 경우 NPM 대신 Yarn을 사용할 수 있습니다. 하지만 CDK는 Yarn 2의 기본 plug-and-play 모드인 Yarn 모드를 지원하지 않습니다. 이 기능을 끄려면 프로젝트 `.yarnrc.yml` 파일에 다음을 추가하세요.

```
nodeLinker: node-modules
```

CDK 애플리케이션

다음은 `cdk init --language typescript` 명령으로 생성된 예제 `package.json` 파일입니다.

```
{
  "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"
  }
}
```

배포 가능한 CDK 앱의 경우 의 `dependencies` 섹션에서 `aws-cdk-lib` 지정해야 합니다. `package.json` 캐럿 (^) 버전 번호 지정자를 사용하여 지정된 버전보다 이후 버전이 동일한 메이저 버전 내에 있는 한 수락할 것임을 표시할 수 있습니다.

실험적 구문의 경우 변경될 수 있는 API가 있는 알파 구문 라이브러리 모듈의 정확한 버전을 지정하십시오. ^나 ~는 사용하지 마세요. 이러한 모듈의 이후 버전에서는 API가 변경되어 앱이 중단될 수 있기 때문입니다.

의 devDependencies 섹션에서 앱을 테스트하는 데 필요한 라이브러리 및 도구 버전 (예: jest 테스트 프레임워크) 을 지정하십시오package.json. 선택적으로 ^를 사용하여 나중에 호환되는 버전도 허용되도록 지정할 수 있습니다.

서드파티 구성 라이브러리

구성 라이브러리를 개발하는 경우 다음 예제 package.json 파일에 표시된 대로 peerDependencies 및 devDependencies 섹션의 조합을 사용하여 종속성을 지정하십시오.

```
{
  "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"
  }
}
```

peerDependencies에서는 캐럿 (^) 을 사용하여 라이브러리가 작동하는 가장 낮은 버전을 지정합니다. aws-cdk-lib 이렇게 하면 라이브러리와 다양한 CDK 버전의 호환성이 극대화됩니다. API가 변경될 수 있는 alpha 구조 라이브러리 모듈의 정확한 버전을 지정하십시오. 를 peerDependencies 사용하면 트리에 모든 CDK 라이브러리의 복사본이 하나만 남게 됩니다node_modules.

에서 테스트에 필요한 도구와 라이브러리를 지정하고devDependencies, 나중에 호환되는 버전도 사용할 수 있음을 나타내려면 ^를 붙일 수도 있습니다. 라이브러리가 호환된다고 광고하는 다른 CDK 패키지 중 가장 낮은 버전 aws-cdk-lib 및 기타 CDK 패키지를 정확히 (^ 또는 ~를 제외하고) 지정하십시오. 이렇게 하면 해당 버전에서 테스트를 실행할 수 있습니다. 이렇게 하면 새 버전에만 있는 기능을 실수로 사용하는 경우 테스트에서 해당 기능을 잡을 수 있습니다.

> **Warning**
>
> peerDependenciesNPM 7 이상에서만 자동으로 설치됩니다. NPM 6 이전 버전을 사용하거나 Yarn을 사용하는 경우 종속성의 종속성을 에 포함해야 합니다. devDependencies 그렇지 않으면 설치되지 않고 해결되지 않은 피어 종속성에 대한 경고를 받게 됩니다.

종속성 설치 및 업데이트

다음 명령어를 실행하여 프로젝트의 종속 항목을 설치합니다.

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
```

설치된 모듈을 업데이트하려면 위 npm install 및 yarn upgrade 명령을 사용할 수 있습니다. 두 명령 모두 패키지를 의 node_modules package.json 규칙을 충족하는 최신 버전으로 업데이트합니다. 하지만 package.json 자체적으로 업데이트되지는 않으므로 새 최소 버전을 설정하는 것이 좋습니다. 에서 GitHub 패키지를 호스팅하는 경우 Dependabot 버전 업데이트가 자동으로 업데이트되도록 구성할 수 있습니다. package.json 또는 npm-check-updates를 사용합니다.

> **Important**
>
> 기본적으로 NPM과 Yarn은 종속성을 설치하거나 업데이트할 때 지정된 요구 사항을 충족하는 모든 패키지의 최신 버전을 선택합니다. package.json 이러한 버전이 (실수로 또는 의도적

으로) 손상될 위험은 항상 존재합니다. 프로젝트 종속성을 업데이트한 후 철저하게 테스트하세요.

AWS CDK 의 관용구 TypeScript

소품

모든 AWS 구성 라이브러리 클래스는 구문이 정의되는 범위 (구성 트리의 상위), id, props라는 세 가지 인수를 사용하여 인스턴스화됩니다. 인수 props는 구문이 생성하는 리소스를 구성하는 데 사용하는 키/값 쌍의 번들입니다. AWS 다른 클래스와 메서드에서도 인수에 "속성 번들" 패턴을 사용합니다.

TypeScript에서는 필수 및 선택적 인수와 해당 유형을 알려주는 인터페이스를 사용하여 의 `props` 모양을 정의합니다. 이러한 인터페이스는 각 유형의 `props` 인수에 대해 정의되며, 일반적으로 단일 구문 또는 메서드에만 적용됩니다. 예를 들어, Bucket 구문 (내`aws-cdk-lib/aws-s3 module`) 은 BucketProps인터페이스를 준수하는 `props` 인수를 지정합니다.

속성 자체가 객체인 경우 (예: 의 WebsiteRedirect 속성)`BucketProps`, 해당 객체는 모양이 준수해야 하는 자체 인터페이스를 갖게 됩니다 (이 경우). RedirectTarget

AWS Construct Library 클래스를 서브클래스화하거나 props와 유사한 인수를 사용하는 메서드를 재정의하는 경우 기존 인터페이스를 상속하여 코드에 필요한 새 props를 지정하는 새 인터페이스를 만들 수 있습니다. 부모 클래스나 기본 메서드를 호출할 때는 일반적으로 받은 props 인수 전체를 전달할 수 있습니다. 객체에 제공되었지만 인터페이스에서 지정되지 않은 모든 속성은 무시되기 때문입니다.

향후 릴리스에서 우연히 귀하의 재산에 사용한 이름을 가진 새 부동산이 AWS CDK 추가될 수 있습니다. 받은 가치를 상속 체인에 전달하면 예상치 못한 동작이 발생할 수 있습니다. 속성을 제거하거나 설정한 상태로 받은 소품의 얕은 사본을 전달하는 것이 더 안전합니다. `undefined` 예:

```
super(scope, name, {...props, encryptionKeys: undefined});
```

또는 건축물에 속한다는 것을 명확히 알 수 있도록 속성에 이름을 붙이는 것도 좋습니다. 이렇게 하면 향후 AWS CDK 릴리스에서 프로퍼티와 충돌할 가능성이 거의 없습니다. 개수가 많으면 적절하게 이름이 지정된 단일 개체를 사용하여 해당 개체를 보관하세요.

누락된 값

객체 (예: props) 에 누락된 값은 값이 0입니다. `undefined` TypeScript 버전 3.7에는 이러한 값 작업을 간소화하는 연산자가 도입되어 정의되지 않은 값에 도달했을 때 디폴트 설정과 "단락" 체인을 쉽게 지

정할 수 있습니다. 이러한 기능에 대한 자세한 내용은 TypeScript 3.7 릴리스 노트, 특히 처음 두 기능인 선택적 체이닝과 Nullish Coalescing을 참조하십시오.

빌드, 합성 및 배포

일반적으로 애플리케이션을 빌드하고 실행할 때는 프로젝트의 루트 디렉터리에 있어야 합니다.

Node.js TypeScript 직접 실행할 수 없습니다. 대신 TypeScript 컴파일러를 JavaScript 사용하여 응용 프로그램을 변환합니다. `tsc` 그러면 결과 JavaScript 코드가 실행됩니다.

는 앱을 실행해야 할 때마다 이 작업을 AWS CDK 자동으로 수행합니다. 하지만 수동으로 컴파일하여 오류를 확인하고 테스트를 실행하는 것이 유용할 수 있습니다. TypeScript 앱을 수동으로 컴파일하려면 다음을 실행하십시오. `npm run build` 또한 감시 모드로 `npm run watch` 전환해야 할 수도 있습니다. 감시 모드에서는 소스 파일에 변경 내용을 저장할 때마다 TypeScript 컴파일러가 앱을 자동으로 다시 빌드합니다.

AWS CDK 앱에 정의된 스택은 아래 명령을 사용하여 개별적으로 또는 함께 합성하고 배포할 수 있습니다. 일반적으로 프로젝트를 실행할 때는 프로젝트의 기본 디렉터리에 있어야 합니다.

- `cdk synth`: 앱에 있는 하나 이상의 스택에서 AWS CloudFormation 템플릿을 합성합니다 AWS CDK .
- `cdk deploy`: 앱에 있는 하나 이상의 스택에서 정의한 리소스를 에 배포합니다. AWS CDK AWS

단일 명령으로 합성하거나 배포할 여러 스택의 이름을 지정할 수 있습니다. 앱이 스택을 하나만 정의하는 경우 스택을 지정할 필요가 없습니다.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

와일드카드* (원하는 수의 문자) 및? 를 사용할 수도 있습니다. (임의의 단일 문자) 를 사용하여 패턴별로 스택을 식별할 수 있습니다. 와일드카드를 사용할 때는 패턴을 따옴표로 묶으십시오. 그렇지 않으면 셸이 툴킷으로 전달되기 전에 현재 디렉토리의 파일 이름으로 확장하려고 할 수 있습니다. AWS CDK

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> 스택을 배포하기 전에 명시적으로 합성할 필요는 없습니다. 이 단계를 `cdk deploy` 수행하면 최신 코드가 배포되도록 할 수 있습니다.

명령의 전체 설명서를 보려면 을 참조하십시오. `cdk` the section called “AWS CDK 툴킷”

In과 함께 AWS CDK 일하기 JavaScript

JavaScript에서 완전히 지원되는 클라이언트 AWS CDK 언어이며 안정적인 것으로 간주됩니다. in을 AWS Cloud Development Kit (AWS CDK) JavaScript 사용하여 작업하면 Node.js 및 Node Package Manager (npm) 와 같은 친숙한 도구가 사용됩니다. 원하는 경우 Yarn을 사용할 수도 있지만, 이 가이드의 예제에서는 NPM을 사용합니다. AWS 구성 라이브러리를 구성하는 모듈은 NPM 저장소인 npmjs.org를 통해 배포됩니다.

모든 에디터나 IDE를 사용할 수 있습니다. 많은 AWS CDK 개발자가 Visual Studio Code (또는 이에 상응하는 오픈 소스 VScodium) 를 사용하는데, 이 코드는 잘 지원됩니다. JavaScript

주제

- JavaScript 시작하기
- 프로젝트 생성
- 로컬 사용 cdk
- 컨스트럭트 라이브러리 모듈 관리 AWS
- 의 종속성 관리 JavaScript
- AWS CDK 의 관용구 JavaScript
- 합성 및 배포
- 다음과 같은 TypeScript 예제 사용하기 JavaScript
- 로 마이그레이션 TypeScript

JavaScript 시작하기

를 사용하려면 AWS 계정과 자격 증명이 있어야 AWS CDK하며 Node.js 및 툴킷이 설치되어 있어야 합니다. AWS CDK 시작하기 AWS CDK 섹션을 참조하십시오.

JavaScript AWS CDK 응용 프로그램에는 이 외에도 추가 전제 조건이 필요하지 않습니다.

> **Note**
>
> 타사 언어 지원 중단: 언어 버전은 공급업체 또는 커뮤니티에서 EOL (End Of Life) 을 공유할 때까지만 지원되며 사전 통지를 통해 변경될 수 있습니다.

프로젝트 생성

빈 `cdk init` 디렉터리에서 호출하여 새 AWS CDK 프로젝트를 생성합니다. `--language`옵션을 사용하고 다음을 지정합니다`javascript`.

```
mkdir my-project
cd my-project
cdk init app --language javascript
```

프로젝트를 생성하면 [aws-cdk-lib](#)모듈과 해당 종속 항목도 설치됩니다.

`cdk init`프로젝트 폴더 이름을 사용하여 클래스, 하위 폴더, 파일 등 프로젝트의 다양한 요소에 이름을 지정합니다. 폴더 이름의 하이픈은 밑줄로 변환됩니다. 하지만 이름은 JavaScript 식별자 형식을 따라야 합니다. 예를 들어 숫자로 시작하거나 공백을 포함해서는 안 됩니다.

로컬 사용 **cdk**

대부분의 경우 이 안내서에서는 CDK 툴킷을 글로벌 설치 () 하고 제공된 명령 예제 (예:`npm install -g aws-cdk`) 가 이 가정을 따르는 것으로 `cdk synth` 가정합니다. 이 방법을 사용하면 CDK Toolkit을 최신 상태로 쉽게 유지할 수 있으며, CDK는 이전 버전과의 호환성에 대해 엄격한 접근 방식을 취하므로 일반적으로 항상 최신 버전을 사용해도 위험은 거의 없습니다.

일부 팀은 CDK Toolkit과 같은 도구를 포함하여 각 프로젝트 내의 모든 종속성을 지정하는 것을 선호합니다. 이렇게 하면 이러한 구성 요소를 특정 버전에 고정하고 팀의 모든 개발자 (및 CI/CD 환경) 가 정확히 해당 버전을 사용하도록 할 수 있습니다. 이렇게 하면 변경 가능성이 제거되므로 빌드와 배포의 일관성과 반복성을 높일 수 있습니다.

CDK에는 JavaScript 프로젝트 템플릿에 CDK 툴킷에 대한 종속 항목이 포함되어 있으므로 이 접근 방식을 사용하려는 경우 프로젝트를 변경할 필요가 없습니다. `package.json` 앱을 빌드할 때와 명령을 실행할 때 약간 다른 명령을 사용하기만 하면 됩니다. `cdk`

Operation	글로벌 CDK 툴킷 사용	로컬 CDK 툴킷 사용
프로젝트 초기화	`cdk init --language #####`	`npx aws-cdk ### --## #####`
CDK 툴킷 명령 실행	`cdk...`	`npm run cdk...` or `npx aws-cdk...`

`npx aws-cdk`현재 프로젝트에 로컬로 설치된 CDK 툴킷 버전이 있는 경우 이를 실행하고, 글로벌 설치 (있는 경우) 로 폴백합니다. 글로벌 설치가 없는 경우 CDK Toolkit의 임시 사본을 npx 다운로드하여 실행합니다. : prints 구문을 사용하여 CDK 툴킷의 임의 버전을 지정할 수 있습니다. @ `npx aws-cdk@1.120 --version 1.120.0`

> **Tip**
>
> 로컬 CDK 툴킷 설치에서 cdk 명령을 사용할 수 있도록 별칭을 설정합니다.
>
> macOS/Linux
>
> ```
> alias cdk="npx aws-cdk"
> ```
>
> Windows
>
> ```
> doskey cdk=npx aws-cdk $*
> ```

컨스트럭트 라이브러리 모듈 관리 AWS

Node Package Manager (npm) 를 사용하면 필요한 다른 패키지뿐 아니라 앱에서 사용할 수 있는 AWS Construct Library 모듈을 설치하고 업데이트할 수 있습니다. (원하는 npm 경우 `yarn` 대신 사용할 수도 있습니다.) npm또한 해당 모듈의 종속성을 자동으로 설치합니다.

대부분의 AWS CDK 구문은 라는 이름의 `aws-cdk-lib` 기본 CDK 패키지에 들어 있습니다. 이 패키지는 에서 만든 새 프로젝트의 기본 종속 항목입니다. cdk init 상위 레벨 AWS 구문이 아직 개발 중인 “실험적” 구조 라이브러리 모듈의 이름은 다음과 같습니다. `aws-cdk-lib/`*`SERVICE-NAME`*`-alpha` 서비스 이름에는 aws - 접두사가 붙습니다. 모듈 이름이 확실하지 않은 경우 NPM에서 검색하세요.

> **Note**
>
> CDK API 레퍼런스에는 패키지 이름도 나와 있습니다.

예를 들어, 아래 명령은 에 대한 실험용 모듈을 설치합니다. AWS CodeStar

```
npm install @aws-cdk/aws-codestar-alpha
```

일부 서비스의 구성 라이브러리 지원은 둘 이상의 네임스페이스에서 제공됩니다. 예를 들어`aws-route53`, 그 외에도 Amazon Route 53 네임스페이스,, `aws-route53-targets``aws-route53-patterns`, 가 세 개 더 있습니다. `aws-route53resolver`

프로젝트의 종속성은 에서 유지 관리됩니다. `package.json` 이 파일을 편집하여 일부 또는 모든 종속성을 특정 버전에 고정하거나 특정 기준에 따라 새 버전으로 업데이트하도록 허용할 수 있습니다. 에서 지정한 규칙에 따라 프로젝트의 NPM 종속성을 허용된 최신 버전으로 업데이트하려면: `package.json`

```
npm update
```

JavaScript에서는 NPM을 사용하여 모듈을 설치할 때 사용하는 것과 동일한 이름으로 모듈을 코드에 가져옵니다. 애플리케이션에서 AWS CDK 클래스와 AWS Construct Library 모듈을 가져올 때는 다음 방법을 따르는 것이 좋습니다. 이 가이드라인을 따르면 코드를 다른 AWS CDK 응용 프로그램과 일관되게 작성할 수 있을 뿐만 아니라 코드를 더 쉽게 이해할 수 있습니다.

- ES6 스타일 `import` 지침이 아니라 사용하십시오`require()`. 이전 버전의 Node.js 에서는 ES6 가져오기를 지원하지 않으므로 이전 구문을 사용하는 것이 더 광범위하게 호환됩니다. (ES6 가져오기를 정말로 사용하고 싶다면 esm을 사용하여 프로젝트가 지원되는 모든 Node.js 버전과 호환되는지 확인하세요.)
- 일반적으로 에서 개별 클래스를 가져옵니다. `aws-cdk-lib`

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

- 에서 많은 클래스가 필요한 경우 `aws-cdk-lib` 개별 클래스를 가져오는 `cdk` 대신 의 네임스페이스 별칭을 사용할 수 있습니다. 둘 다 하지 마세요.

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

- 일반적으로 짧은 네임스페이스 별칭을 사용하여 AWS Construct Libraries를 가져오세요.

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

의 종속성 관리 JavaScript

JavaScriptCDK 프로젝트에서 종속성은 프로젝트의 기본 디렉토리에 있는 `package.json` 파일에 지정됩니다. 핵심 AWS CDK 모듈은 라는 단일 NPM 패키지에 들어 있습니다. `aws-cdk-lib`

를 사용하여 npm install 패키지를 설치하면 NPM이 패키지를 자동으로 기록합니다. `package.json`

원하는 경우 NPM 대신 Yarn을 사용할 수 있습니다. 하지만 CDK는 Yarn 2의 기본 plug-and-play 모드인 Yarn 모드를 지원하지 않습니다. 이 기능을 끄려면 프로젝트 `.yarnrc.yml` 파일에 다음을 추가하세요.

```
nodeLinker: node-modules
```

CDK 애플리케이션

다음은 `cdk init --language typescript` 명령으로 생성된 예제 `package.json` 파일입니다. 에 대해 JavaScript 생성된 파일은 TypeScript 관련 항목이 없다는 점만 제외하면 비슷합니다.

```
{
  "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"
  }
}
```

배포 가능한 CDK 앱의 경우 의 dependencies 섹션에서 aws-cdk-lib 지정해야 합니다. package.json 캐럿 (^) 버전 번호 지정자를 사용하여 지정된 버전보다 이후 버전이 동일한 메이저 버전 내에 있는 한 수락할 것임을 표시할 수 있습니다.

실험적 구문의 경우 변경될 수 있는 API가 있는 알파 구문 라이브러리 모듈의 정확한 버전을 지정하십시오. ^나 ~는 사용하지 마세요. 이러한 모듈의 이후 버전에서는 API가 변경되어 앱이 중단될 수 있기 때문입니다.

의 devDependencies 섹션에서 앱을 테스트하는 데 필요한 라이브러리 및 도구 버전 (예: jest 테스트 프레임워크) 을 지정하십시오package.json. 선택적으로 ^를 사용하여 나중에 호환되는 버전도 허용되도록 지정할 수 있습니다.

타사 구성 라이브러리

구성 라이브러리를 개발하는 경우 다음 예제 package.json 파일에 표시된 대로 peerDependencies 및 devDependencies 섹션의 조합을 사용하여 종속성을 지정하십시오.

```
{
  "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"
  }
```

```
}
```

peerDependencies에서는 캐럿 (^) 을 사용하여 라이브러리가 작동하는 가장 낮은 버전을 지정합니다. aws-cdk-lib 이렇게 하면 라이브러리와 다양한 CDK 버전의 호환성이 극대화됩니다. API가 변경될 수 있는 alpha 구조 라이브러리 모듈의 정확한 버전을 지정하십시오. 를 peerDependencies 사용하면 트리에 모든 CDK 라이브러리의 복사본이 하나만 남게 됩니다node_modules.

에서 테스트에 필요한 도구와 라이브러리를 지정하고devDependencies, 나중에 호환되는 버전도 사용할 수 있음을 나타내려면 ^를 붙일 수도 있습니다. 라이브러리가 호환된다고 광고하는 다른 CDK 패키지 중 가장 낮은 버전 aws-cdk-lib 및 기타 CDK 패키지를 정확히 (^ 또는 ~를 제외하고) 지정하십시오. 이렇게 하면 해당 버전에서 테스트를 실행할 수 있습니다. 이렇게 하면 새 버전에만 있는 기능을 실수로 사용하는 경우 테스트에서 해당 기능을 잡을 수 있습니다.

> **Warning**
>
> peerDependenciesNPM 7 이상에서만 자동으로 설치됩니다. NPM 6 이전 버전을 사용하거나 Yarn을 사용하는 경우 종속성의 종속성을 에 포함해야 합니다. devDependencies 그렇지 않으면 설치되지 않고 해결되지 않은 피어 종속성에 대한 경고를 받게 됩니다.

종속성 설치 및 업데이트

다음 명령어를 실행하여 프로젝트의 종속 항목을 설치합니다.

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
```

설치된 모듈을 업데이트하려면 위 npm install 및 yarn upgrade 명령을 사용할 수 있습니다. 두 명령 모두 패키지를 의 node_modules package.json 규칙을 충족하는 최신 버전으로 업데이트합니다. 하지만 package.json 자체적으로 업데이트되지는 않으므로 새 최소 버전을 설정하는 것이 좋습니다. 에서 GitHub 패키지를 호스팅하는 경우 Dependabot 버전 업데이트가 자동으로 업데이트되도록 구성할 수 있습니다. package.json 또는 npm-check-updates를 사용합니다.

> **Important**
>
> 기본적으로 NPM과 Yarn은 종속성을 설치하거나 업데이트할 때 지정된 요구 사항을 충족하는 모든 패키지의 최신 버전을 선택합니다. package.json 이러한 버전이 (실수로 또는 의도적으로) 손상될 위험은 항상 존재합니다. 프로젝트 종속성을 업데이트한 후 철저하게 테스트하세요.

AWS CDK 의 관용구 JavaScript

소품

모든 AWS Construct Library 클래스는 구문이 정의되는 범위 (구성 트리의 부모), id, props라는 세 가지 인수를 사용하여 인스턴스화됩니다. props는 구문이 생성하는 리소스를 구성하는 데 사용하는 키/값 쌍의 번들입니다. AWS 다른 클래스와 메서드에서도 인수에 “속성 번들” 패턴을 사용합니다.

JavaScript 자동 완성 기능이 있는 IDE 또는 편집기를 사용하면 속성 이름의 철자 오류를 방지하는 데 도움이 됩니다. 구문에 필요한 encryptionKeys 속성에 철자를 입력하면 구문을 인스턴스화할 때 의도한 값이 전달되지 않은 것입니다. encryptionkeys 이 경우 속성이 필요한 경우 합성 시 오류가 발생하고, 선택 사항인 경우 속성이 자동으로 무시될 수 있습니다. 후자의 경우 오버라이드하려는 기본 동작이 발생할 수 있습니다. 여기서는 특히 주의해야 합니다.

AWS Construct Library 클래스를 서브클래스화할 때 (또는 props와 유사한 인수를 사용하는 메서드를 오버라이드할 때), 직접 사용할 수 있도록 추가 속성을 허용해야 할 수도 있습니다. 이러한 값은 부모 클래스나 오버라이드된 메서드에서 무시됩니다. 해당 코드에서는 이러한 값에 액세스할 수 없으므로 일반적으로 받은 모든 소품을 전달할 수 있기 때문입니다.

향후 릴리스에서 우연히 귀하의 재산에 사용한 이름을 가진 새 부동산이 AWS CDK 추가될 수 있습니다. 받은 가치를 상속 체인에 전달하면 예상치 못한 동작이 발생할 수 있습니다. 속성을 제거하거나 설정한 상태로 받은 소품의 얕은 사본을 전달하는 것이 더 안전합니다. undefined 예:

```
super(scope, name, {...props, encryptionKeys: undefined});
```

또는 건축물에 속한다는 것을 명확히 알 수 있도록 속성에 이름을 붙이는 것도 좋습니다. 이렇게 하면 향후 AWS CDK 릴리스에서 프로퍼티와 충돌할 가능성이 거의 없습니다. 개수가 많으면 적절하게 이름이 지정된 단일 개체를 사용하여 해당 개체를 보관하세요.

누락된 값

개체에 누락된 값 (예:`props`) 에는 값이 들어 있습니다. `undefined` JavaScript 이러한 문제를 처리하는 데에는 일반적인 방법이 적용됩니다. 예를 들어, 정의되지 않은 값의 속성에 액세스할 때 흔히 사용되는 관용구는 다음과 같습니다.

```
// 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;
```

그러나 다른 'false' 값이 a 있을 수 있다면 테스트를 좀 더 명확하게 하는 것이 좋습니다. `undefined` 여기서는 `null` 과 `undefined` 가 같다는 점을 이용하여 두 값을 동시에 테스트해 보겠습니다.

```
let c = a == null ? a : a.b;
```

> **Tip**
>
> Node.js 14.0 이상에서는 정의되지 않은 값의 처리를 단순화할 수 있는 새로운 연산자를 지원합니다. 자세한 내용은 선택적 체이닝 및 nullish 병합 제안을 참조하십시오.

합성 및 배포

AWS CDK 앱에 정의된 스택은 아래 명령을 사용하여 개별적으로 또는 함께 합성하고 배포할 수 있습니다. 일반적으로 프로젝트를 실행할 때는 프로젝트의 기본 디렉터리에 있어야 합니다.

- `cdk synth`: 앱에 있는 하나 이상의 스택에서 AWS CloudFormation 템플릿을 합성합니다 AWS CDK .
- `cdk deploy`: 앱에 있는 하나 이상의 스택에서 정의한 리소스를 에 배포합니다. AWS CDK AWS

단일 명령으로 합성하거나 배포할 여러 스택의 이름을 지정할 수 있습니다. 앱이 스택을 하나만 정의하는 경우 스택을 지정할 필요가 없습니다.

```
cdk synth                # app defines single stack
```

```
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

와일드카드* (원하는 수의 문자) 및? 를 사용할 수도 있습니다. (임의의 단일 문자) 를 사용하여 패턴별로 스택을 식별할 수 있습니다. 와일드카드를 사용할 때는 패턴을 따옴표로 묶으십시오. 그렇지 않으면 셸이 툴킷으로 전달되기 전에 현재 디렉토리의 파일 이름으로 확장하려고 할 수 있습니다. AWS CDK

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> 스택을 배포하기 전에 명시적으로 합성할 필요는 없습니다. 이 단계를 `cdk deploy` 수행하면 최신 코드가 배포되도록 할 수 있습니다.

명령의 전체 설명서를 보려면 을 참조하십시오. `cdk` the section called “AWS CDK 툴킷”

다음과 같은 TypeScript 예제 사용하기 JavaScript

TypeScript는 우리가 개발하는 데 사용하는 언어이며 응용 프로그램 개발에 지원되는 첫 번째 언어이기 때문에 사용 가능한 AWS CDK 코드 예제가 많이 작성되어 TypeScript 있습니다. AWS CDK이러한 코드 예제는 JavaScript 개발자에게 유용한 리소스가 될 수 있습니다. 코드의 TypeScript 특정 부분만 제거하면 됩니다.

TypeScript 스니펫은 종종 최신 `import` ECMAScript와 `export` 키워드를 사용하여 다른 모듈에서 객체를 가져오고 해당 객체를 현재 모듈 외부에서 사용할 수 있도록 선언합니다. Node.js 최신 릴리스에서 이러한 키워드를 지원하기 시작한 지 얼마 되지 않았습니다. 사용 중인 (또는 지원하려는) Node.js 버전에 따라 이전 구문을 사용하도록 가져오기 및 내보내기를 다시 작성할 수 있습니다.

가져오기를 `require()` 함수 호출로 대체할 수 있습니다.

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');
```

내보내기를 `module.exports` 개체에 할당할 수 있습니다.

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**
>
> 이전 스타일의 가져오기와 내보내기를 사용하는 대신 모듈을 사용할 수 있습니다. esm

가져오기와 내보내기를 정렬했으면 실제 코드를 자세히 살펴볼 수 있습니다. 일반적으로 사용되는 TypeScript 다음과 같은 기능이 발생할 수 있습니다.

- 유형 주석
- 인터페이스 정의
- 타입 변환/캐스트
- 액세스 한정자

변수, 클래스 멤버, 함수 매개 변수 및 함수 반환 유형에 대해 형식 주석을 제공할 수 있습니다. 변수, 매개 변수 및 멤버의 경우 식별자 뒤에 콜론과 형식을 붙여 형식을 지정합니다. 함수 반환 값은 함수 서명을 따르며 콜론과 유형으로 구성됩니다.

유형 주석이 달린 코드를 로 변환하려면 콜론과 JavaScript 유형을 제거합니다. 클래스 멤버에는 특정 값이 있어야 합니다. 클래스 멤버에 형식 주석만 있는 undefined 경우에는 클래스 멤버를 로 설정하십시오 JavaScript. 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) {
    // ...
}
```

에서 TypeScript 인터페이스는 필수 및 선택적 속성의 묶음과 해당 유형, 이름을 지정하는 데 사용됩니다. 그런 다음 인터페이스 이름을 유형 주석으로 사용할 수 있습니다. TypeScript 예를 들어 함수의 인수로 사용하는 객체가 올바른 유형의 필수 속성을 가지고 있는지 확인합니다.

```
interface myFuncProps {
    code: lambda.Code,
    handler?: string
```

```
}
```

JavaScript 에는 인터페이스 기능이 없으므로 형식 주석을 제거한 후에는 인터페이스 선언을 완전히 삭제하십시오.

함수나 메서드가 범용 형식 (예:`object`) 을 반환하지만 이 값을 좀 더 구체적인 하위 유형으로 취급하여 보다 일반적인 형식의 인터페이스에 속하지 않는 속성이나 메서드에 액세스하려는 경우 형식 또는 인터페이스 이름을 사용하여 `as` 값을 캐스팅할 수 있습니다. TypeScript JavaScript 이 식별자를 지원 (또는 필요) 하지 않으므로 다음 식별자를 `as` 제거하기만 하면 됩니다. 덜 일반적인 캐스트 구문은 대괄호 안에 타입 이름을 사용하는 것인데, 이러한 캐스트도 삭제해야 합니다`<LikeThis>`.

마지막으로, 액세스 한정자 및 클래스 `public` `protected` `private` 멤버를 TypeScript 지원합니다. 의 모든 클래스 JavaScript 멤버는 공개됩니다. 어디에서든 이 수정자를 제거하기만 하면 됩니다.

이러한 TypeScript 기능을 식별하고 제거하는 방법을 알면 짧은 TypeScript 스니펫을 적용하는 데 큰 도움이 됩니다. JavaScript 하지만 긴 TypeScript 예제는 다른 기능을 사용할 가능성이 더 높기 때문에 이런 방식으로 변환하는 것은 비실용적일 수 있습니다. TypeScript 이러한 상황에서는 Sucrase를 사용하는 것이 좋습니다. 예를 들어 코드에서 정의되지 않은 변수를 사용하는 경우 Sucrase는 불평하지 않습니다. `tsc` 구문상 유효하다면 몇 가지 예외를 제외하고 Sucrase는 이를 다음과 같이 변환할 수 있습니다. JavaScript 따라서 자체적으로 실행할 수 없는 스니펫을 변환하는 데 특히 유용합니다.

로 마이그레이션 TypeScript

프로젝트 규모가 커지고 TypeScript복잡해짐에 따라 많은 JavaScript 개발자들이 이전합니다. TypeScript JavaScript모든 JavaScript 코드가 유효한 코드이므로 코드를 변경할 필요가 없는 TypeScript 코드의 상위 집합이며 지원되는 언어이기도 합니다. AWS CDK 유형 주석 및 기타 TypeScript 기능은 선택 사항이며 가치가 있는 경우 AWS CDK 앱에 추가할 수 있습니다. TypeScript 또한 Node.js 업그레이드 없이도 선택적 체이닝 및 nullish 병합과 같은 새로운 JavaScript 기능을 완성하기 전에 미리 사용할 수 있습니다.

TypeScript개체 내에서 필수 및 선택적 속성 (및 해당 유형) 의 번들을 정의하는 의 "모양 기반" 인터페이스를 사용하면 코드를 작성하는 동안 흔히 발생하는 실수를 찾아낼 수 있으며 IDE에서 강력한 자동 완성 및 기타 실시간 코딩 조언을 더 쉽게 제공할 수 있습니다.

TypeScript 코딩에는 컴파일러로 앱을 컴파일하는 추가 단계가 필요합니다. TypeScript `tsc` 일반적인 AWS CDK 앱의 경우 컴파일하는 데 기껏해야 몇 초가 걸립니다.

기존 JavaScript AWS CDK 앱을 마이그레이션하는 가장 쉬운 방법은 를 사용하여 `cdk init app --language typescript` 새 TypeScript 프로젝트를 만든 다음 소스 파일 (및 AWS Lambda 함수 소

스 코드와 같은 기타 필수 파일) 을 새 프로젝트에 복사하는 것입니다. TypeScript JavaScript 파일 이름을 변경하여 끝에서 `.ts` TypeScript 개발을 시작하세요.

AWS CDK Python에서 작업하기

Python은 에서 완전히 지원되는 클라이언트 AWS Cloud Development Kit (AWS CDK) 언어이며 안정적인 것으로 간주됩니다. Python으로 작업하면 AWS CDK 표준 Python 구현 (CPython), 가상 환경, Python 패키지 설치 프로그램 `pip` 등 친숙한 도구가 사용됩니다. `virtualenv` AWS 구성 라이브러리를 구성하는 모듈은 pypi.org를 통해 배포됩니다. Python 버전의 이벤트는 Python 스타일 식별자 (예: 메서드 이름) 를 사용합니다. AWS CDK `snake_case`

모든 에디터나 IDE를 사용할 수 있습니다. 많은 AWS CDK 개발자들이 공식 확장 프로그램을 통해 Python을 잘 지원하는 Visual Studio Code (또는 이에 상응하는 오픈 소스 VScodium) 를 사용합니다. Python에 포함된 IDLE 편집기는 시작하기에 충분합니다. 의 Python 모듈에는 유형 힌트가 있으며, 이는 유형 유효성 AWS CDK 검사를 지원하는 린팅 도구 또는 IDE에 유용합니다.

주제

- Python 시작하기
- 프로젝트 생성
- AWS 구성 라이브러리 모듈 관리
- 에서 종속성 관리 Python
- AWS CDK 파이썬의 관용구
- 합성 및 배포

Python 시작하기

를 사용하려면 AWS 계정과 자격 증명이 있어야 AWS CDK하며 Node.js 및 툴킷이 설치되어 있어야 합니다. AWS CDK 시작하기 AWS CDK 섹션을 참조하십시오.

파이썬 AWS CDK 응용 프로그램에는 Python 3.6 이상이 필요합니다. 아직 설치하지 않았다면 python.org에서 운영 체제와 호환되는 버전을 다운로드하십시오. Linux를 실행하는 경우 시스템에 호환되는 버전이 설치되어 있거나 배포판의 패키지 관리자 (`yum`, `apt` 등) 를 사용하여 설치할 수 있습니다. Mac 사용자는 macOS용 리눅스 스타일 패키지 관리자인 Homebrew에 관심이 있을 수 있습니다.

> **Note**
>
> 타사 언어 지원 중단: 언어 버전은 공급업체 또는 커뮤니티에서 EOL (End Of Life) 을 공유할 때까지만 지원되며 사전 통지로 변경될 수 있습니다.

Python 패키지 설치 프로그램과 가상 환경 관리자도 필요합니다. `pip` `virtualenv` 호환되는 Python 버전의 Windows 설치에는 다음과 같은 도구가 포함됩니다. Linux에서는 `pip` 패키지 관리자에 별도의 패키지로 `virtualenv` 제공될 수 있습니다. 또는 다음 명령을 사용하여 설치할 수도 있습니다.

```
python -m ensurepip --upgrade
python -m pip install --upgrade pip
python -m pip install --upgrade virtualenv
```

권한 오류가 발생하는 경우 `--user` 플래그를 사용하여 위 명령을 실행하여 모듈이 사용자 디렉토리에 설치되도록 하거나 를 사용하여 `sudo` 모듈을 시스템 전체에 설치할 수 있는 권한을 얻으십시오.

> **Note**
>
> 리눅스 배포판은 Python 3.x의 실행 파일 이름을 사용하고 `python3` Python 2.x 설치를 `python` 참조하는 것이 일반적입니다. 일부 배포판에는 `python` 명령이 Python 3을 참조하도록 하는 설치 가능한 선택적 패키지가 있습니다. 그렇지 않으면 프로젝트의 메인 `cdk.json` 디렉터리에서 편집하여 애플리케이션을 실행하는 데 사용되는 명령을 조정할 수 있습니다.

> **Note**
>
> Windows에서는 py 실행 파일인 Windows용 >Python 런처를 사용하여 Python (및pip) 을 호출할 수 있습니다. 무엇보다도 런처를 사용하면 사용하려는 설치된 Python 버전을 쉽게 지정할 수 있습니다.
>
> Windows 버전의 Python을 설치한 후에도 명령줄에 입력하여 python Windows Store에서 Python을 설치하라는 메시지가 표시되면 Windows의 앱 실행 별칭 관리 설정 패널을 열고 Python용 앱 설치 프로그램 항목 두 개를 끄십시오.

프로젝트 생성

빈 디렉터리에서 `cdk init` 호출하여 새 AWS CDK 프로젝트를 생성합니다. `--language`옵션을 사용하고 다음을 지정합니다`python`.

```
mkdir my-project
cd my-project
cdk init app --language python
```

`cdk init`프로젝트 폴더 이름을 사용하여 클래스, 하위 폴더, 파일 등 프로젝트의 다양한 요소에 이름을 지정합니다. 폴더 이름의 하이픈은 밑줄로 변환됩니다. 하지만 이름은 Python 식별자의 형식을 따라야 합니다. 예를 들어, 숫자로 시작하거나 공백을 포함해서는 안 됩니다.

새 프로젝트를 사용하려면 해당 가상 환경을 활성화하세요. 이렇게 하면 프로젝트의 종속성을 전역적으로 설치하는 대신 프로젝트 폴더에 로컬로 설치할 수 있습니다.

```
source .venv/bin/activate
```

> **Note**
>
> 이를 Mac/Linux 명령으로 인식하여 가상 환경을 활성화할 수 있습니다. Python 템플릿에는 Windows에서 동일한 명령을 사용할 수 있도록 하는 배치 파일인 `source.bat`가 포함되어 있습니다. 기존 Windows `.venv\Scripts\activate.bat` 명령어인, 도 작동합니다.
> CDK Toolkit v1.70.0 또는 이전 버전을 사용하여 AWS CDK 프로젝트를 초기화한 경우 가상 환경이 대신 디렉터리에 있습니다. `.env` `.venv`

> **Important**
>
> 작업을 시작할 때마다 프로젝트의 가상 환경을 활성화하십시오. 그렇지 않으면 거기에 설치된 모듈에 액세스할 수 없으며 설치한 모듈은 Python 글로벌 모듈 디렉터리로 이동합니다 (또는 권한 오류가 발생합니다).

가상 환경을 처음 활성화한 후 앱의 표준 종속 항목을 설치하세요.

```
python -m pip install -r requirements.txt
```

AWS 구성 라이브러리 모듈 관리

Python 패키지 설치 프로그램을 사용하여 필요한 다른 패키지뿐 아니라 앱에서 사용할 수 있도록 AWS Construct Library 모듈을 설치하고 업데이트하십시오. pip pip또한 해당 모듈의 종속 항목을 자동으로 설치합니다. 시스템이 독립형 pip 명령으로 인식되지 않는 경우 다음과 pip 같이 Python 모듈로 호출하십시오.

```
python -m pip PIP-COMMAND
```

대부분의 AWS CDK 구문은 안에 있습니다. `aws-cdk-lib` 실험 모듈은 와 같은 `aws-cdk.SERVICE-NAME.alpha` 이름을 가진 별도의 모듈에 있습니다. 서비스 이름에는 aws 접두사가 포함됩니다. 모듈 이름이 확실하지 않은 경우 PyPI에서 검색하세요. 예를 들어, 아래 명령은 라이브러리를 설치합니다. AWS CodeStar

```
python -m pip install aws-cdk.aws-codestar-alpha
```

일부 서비스의 구조는 둘 이상의 네임스페이스에 있습니다. 예를 들어, 이외에도`aws-cdk.aws-route53`, 및 라는 `aws-route53-targets` 이름의 Amazon Route 53 네임스페이스가 세 개 더 있습니다. `aws-route53-patterns` `aws-route53resolver`

> **Note**
>
> CDK API 레퍼런스의 Python 에디션에도 패키지 이름이 나와 있습니다.

AWS Construct Library 모듈을 Python 코드로 가져오는 데 사용되는 이름은 다음과 같습니다.

```
import aws_cdk.aws_s3 as s3
import aws_cdk.aws_lambda as lambda_
```

응용 프로그램에서 AWS CDK 클래스와 AWS Construct Library 모듈을 가져올 때는 다음 방법을 따르는 것이 좋습니다. 이 가이드라인을 따르면 코드를 다른 AWS CDK 응용 프로그램과 일관되게 작성할 수 있을 뿐만 아니라 코드를 더 쉽게 이해할 수 있습니다.

- 일반적으로 최상위 수준에서 `aws_cdk` 개별 클래스를 가져옵니다.

  ```
  from aws_cdk import App, Construct
  ```

- 에서 많은 클래스가 필요한 경우 개별 aws_cdk 클래스를 가져오는 cdk 대신 의 네임스페이스 별칭을 사용할 수 있습니다. 둘 다 하지 마세요.

```
import aws_cdk as cdk
```

- 일반적으로 짧은 네임스페이스 별칭을 사용하여 AWS Construct Libraries를 가져오세요.

```
import aws_cdk.aws_s3 as s3
```

모듈을 설치한 후 프로젝트 파일을 업데이트하세요. 이 requirements.txt 파일에는 프로젝트의 종속성이 나열되어 있습니다. 를 사용하는 pip freeze 것보다 수동으로 수행하는 것이 가장 좋습니다. pip freezePython 가상 환경에 설치된 모든 모듈의 현재 버전을 캡처합니다. 이는 다른 곳에서 실행할 프로젝트를 번들링할 때 유용할 수 있습니다.

하지만 일반적으로 최상위 종속성 (앱이 직접 의존하는 모듈) 만 나열하고 해당 라이브러리의 종속성은 나열하지 않아야 합니다. requirements.txt 이 전략을 사용하면 종속성을 더 간단하게 업데이트할 수 있습니다.

업그레이드를 requirements.txt 허용하도록 편집할 수 있습니다. 호환성이 더 높은 버전으로 ~= 업그레이드하려면 == 앞의 버전 번호를 로 바꾸거나 버전 요구 사항을 완전히 제거하여 사용 가능한 최신 모듈 버전을 지정할 수 있습니다.

업그레이드가 가능하도록 적절하게 requirements.txt 편집한 후에는 다음 명령을 실행하여 프로젝트에 설치된 모듈을 언제든지 업그레이드할 수 있습니다.

```
pip install --upgrade -r requirements.txt
```

에서 종속성 관리 Python

Python에서는 응용 프로그램이나 setup.py 구성 라이브러리에 종속성을 삽입하여 종속성을 지정합니다. requirements.txt 그런 다음 PIP 도구를 사용하여 종속성을 관리합니다. PIP는 다음 방법 중 하나로 호출됩니다.

```
pip command options
python -m pip command options
```

python -m pip호출은 대부분의 시스템에서 pip 작동하므로 PIP 실행 파일이 시스템 경로에 있어야 합니다. pip작동하지 않으면 로 교체해 보세요. python -m pip

이 cdk init --language python 명령은 새 프로젝트를 위한 가상 환경을 만듭니다. 이렇게 하면 각 프로젝트에 고유한 버전의 종속성과 기본 `requirements.txt` 파일을 지정할 수 있습니다. 프로젝트 작업을 시작할 source .venv/bin/activate 때마다 이 가상 환경을 실행하여 활성화해야 합니다.

CDK 애플리케이션

다음은 예 `requirements.txt` 파일입니다. PIP에는 종속성 잠금 기능이 없으므로 다음과 같이 == 연산자를 사용하여 모든 종속성에 대해 정확한 버전을 지정하는 것이 좋습니다.

```
aws-cdk-lib==2.14.0
aws-cdk.aws-appsync-alpha==2.10.0a0
```

로 pip install 모듈을 설치해도 모듈이 자동으로 추가되지는 않습니다. `requirements.txt` 직접 해야 합니다. 종속 항목의 최신 버전으로 업그레이드하려면 에서 `requirements.txt` 버전 번호를 편집하십시오.

프로젝트 생성 또는 편집 후 프로젝트 종속성을 설치하거나 `requirements.txt` 업데이트하려면 다음을 실행하십시오.

```
python -m pip install -r requirements.txt
```

> **Tip**
>
> 이 pip freeze 명령은 설치된 모든 종속성 버전을 텍스트 파일에 쓸 수 있는 형식으로 출력합니다. 와 함께 `pip install -r` 요구 사항 파일로 사용할 수 있습니다. 이 파일은 모든 종속성(전이 종속성 포함) 을 테스트할 때 사용한 정확한 버전에 고정하는 데 편리합니다. 나중에 패키지를 업그레이드할 때 문제가 발생하지 않도록 하려면 `freeze.txt` (not) 와 같은 별도의 파일을 사용하십시오. `requirements.txt` 그런 다음 프로젝트의 종속성을 업그레이드할 때 다시 생성하세요.

서드파티 구성 라이브러리

라이브러리에서는 종속성이 지정되므로 `setup.py` 애플리케이션에서 패키지를 사용할 때 전이적 종속성이 자동으로 다운로드됩니다. 그렇지 않으면 패키지를 사용하려는 모든 애플리케이션이 종속성을 해당 패키지로 복사해야 합니다. `requirements.txt` 여기에 `setup.py` 예가 나와 있습니다.

```
from setuptools import setup
```

```
setup(
  name='my-package',
  version='0.0.1',
  install_requires=[
    'aws-cdk-lib==2.14.0',
  ],
  ...
)
```

개발용 패키지를 작업하려면 가상 환경을 만들거나 활성화한 후 다음 명령을 실행합니다.

```
python -m pip install -e .
```

PIP는 전이적 종속성을 자동으로 설치하지만 한 패키지의 복사본은 하나만 설치할 수 있습니다. 종속성 트리에서 가장 많이 지정된 버전이 선택되며, 설치되는 패키지 버전의 최종 결정은 항상 애플리케이션에 표시됩니다.

AWS CDK 파이썬의 관용구

언어 충돌

Python에서 `lambda` 는 언어 키워드이므로 AWS Lambda 구성 라이브러리 모듈 또는 Lambda 함수의 이름으로 사용할 수 없습니다. 이러한 충돌에 대한 Python 규칙은 변수 이름에서와 `lambda_` 같이 후행 밑줄을 사용하는 것입니다.

규칙에 따라 구문의 두 번째 인수에는 이름이 AWS CDK 지정됩니다. `id` 스택과 구문을 직접 작성할 때 매개 변수를 호출하면 객체의 고유 식별자를 반환하는 Python 내장 함수를 `id` `id()` “그림자”로 만듭니다. 이 함수는 자주 사용되지는 않지만, 예를 들어 구문에 이 함수가 필요할 경우 인수 이름을 바꾸세요. `construct_id`

인수 및 속성

모든 AWS Construct Library 클래스는 구문이 정의되는 범위 (구성 트리의 부모), id, props, 구문이 생성하는 리소스를 구성하는 데 사용하는 키/값 쌍의 번들이라는 세 가지 인수를 사용하여 인스턴스화됩니다. 다른 클래스와 메서드에서도 인수에 “속성 번들” 패턴을 사용합니다.

scope 및 id는 항상 키워드 인수가 아닌 위치 인수로 전달되어야 합니다. 구문이 scope 또는 id라는 속성을 허용하는 경우 이름이 변경되기 때문입니다.

Python에서 props는 키워드 인자로 표현됩니다. 인수에 중첩된 데이터 구조가 포함된 경우 인스턴스화 시 자체 키워드 인수를 취하는 클래스를 사용하여 이러한 데이터 구조를 표현합니다. 구조화된 인수를 사용하는 다른 메서드 호출에도 동일한 패턴이 적용됩니다.

예를 들어 Amazon S3 버킷의 `add_lifecycle_rule` 메서드에서 `transitions` 속성은 `Transition` 인스턴스 목록입니다.

```
bucket.add_lifecycle_rule(
  transitions=[
    Transition(
      storage_class=StorageClass.GLACIER,
      transition_after=Duration.days(10)
    )
  ]
)
```

클래스를 확장하거나 메서드를 재정의할 때 부모 클래스에서는 이해할 수 없는 추가 인수를 원하는 용도로 사용할 수 있습니다. 이 경우 관용구를 사용해도 상관없는 인수는 받아들이고, 키워드 전용 인수를 사용하여 원하는 인수를 받아들여야 합니다. `**kwargs` 부모 생성자나 오버라이드된 메서드를 호출할 때는 예상한 인수만 전달하세요 (대개 그냥). `**kwargs` 부모 클래스나 메서드가 예상하지 않은 인수를 전달하면 오류가 발생합니다.

```
class MyConstruct(Construct):
    def __init__(self, id, *, MyProperty=42, **kwargs):
        super().__init__(self, id, **kwargs)
        # ...
```

향후 릴리스에서 우연히 귀하의 재산에 사용한 이름을 가진 새 부동산이 AWS CDK 추가될 수 있습니다. 이렇게 해도 구문이나 메서드 사용자에게 기술적인 문제는 발생하지 않지만 (속성이 '체인' 위로 '전달되지 않기 때문에 부모 클래스나 오버라이드된 메서드는 단순히 디폴트 값을 사용함) 혼란을 야기할 수 있습니다. 속성의 이름을 지정하여 구문에 명확하게 속하도록 하면 이러한 잠재적 문제를 피할 수 있습니다. 새 속성이 많으면 해당 속성을 적절한 이름의 클래스로 묶어 단일 키워드 인수로 전달하십시오.

누락된 값

누락되거나 AWS CDK 정의되지 None 않은 값을 나타내는 데 사용됩니다. 를 사용하여 작업할 `**kwargs` 때는 속성이 제공되지 않은 경우 사전의 `get()` 메서드를 사용하여 기본값을 제공하십시오. 이 `kwargs[...]` 경우 값이 누락될 수 `KeyError` 있으므로 사용하지 마십시오.

```
encrypted = kwargs.get("encrypted")         # None if no property "encrypted" exists
encrypted = kwargs.get("encrypted", False)  # specify default of False if property is
 missing
```

일부 AWS CDK 메서드 (예`tryGetContext()`: 런타임 컨텍스트 값 가져오기) 가 반환될 수 있으며None, 이 경우 명시적으로 확인해야 합니다.

인터페이스 사용

Python에는 다른 언어처럼 인터페이스 기능이 없지만 비슷한 추상 기본 클래스가 있습니다. (인터페이스에 익숙하지 않다면 Wikipedia에 대한 소개를 참고하세요.) TypeScript구현된 언어는 인터페이스를 제공하며, 구문 및 기타 AWS CDK 객체에는 특정 클래스로부터 상속되는 대신 특정 인터페이스를 준수하는 객체가 필요한 경우가 많습니다. AWS CDK 따라서 는 AWS CDK JSII 계층의 일부로 자체 인터페이스 기능을 제공합니다.

클래스가 특정 인터페이스를 구현한다는 것을 나타내려면 데코레이터를 사용하면 됩니다.`@jsii.implements`

```
from aws_cdk import IAspect, IConstruct
import jsii

@jsii.implements(IAspect)
class MyAspect():
    def visit(self, node: IConstruct) -> None:
        print("Visited", node.node.path)
```

유형 관련 문제

Python은 모든 변수가 모든 유형의 값을 참조할 수 있는 동적 타이핑을 사용합니다. 매개 변수와 반환 값에는 유형으로 주석을 달 수 있지만 이는 “힌트”이므로 적용되지 않습니다. 즉, Python에서는 잘못된 유형의 값을 구문에 쉽게 전달할 수 AWS CDK 있습니다. 정적 형식 언어에서처럼 빌드 중에 형식 오류가 발생하는 대신 JSII 계층 (Python과 TypeScript 코어 사이를 변환) 이 예상치 못한 유형을 처리할 수 없을 때 런타임 오류가 발생할 수 있습니다. AWS CDK

경험상 Python 프로그래머가 범하는 유형 오류는 이러한 범주에 속하는 경향이 있습니다.

- 구문에 컨테이너 (Python 목록 또는 사전) 가 필요한 경우 단일 값을 전달하거나 그 반대의 경우도 마찬가지입니다.
- layer 1 (`CfnXxxxxx`) 구문과 관련된 유형의 값을 L2 또는 L3 구문에 전달하거나 그 반대로 전달합니다.

AWS CDK Python 모듈에는 유형 주석이 포함되어 있으므로 이를 지원하는 도구를 사용하여 유형을 쉽게 처리할 수 있습니다. 예를 들어 이러한 기능을 지원하는 IDE를 사용하지 않는 경우 빌드 프로세스의 한 PyCharm단계로 MyPy형식 유효성 검사기를 호출하는 것이 좋습니다. 유형 관련 오류에 대한 오류 메시지를 개선할 수 있는 런타임 유형 검사기도 있습니다.

합성 및 배포

AWS CDK 앱에 정의된 스택은 아래 명령을 사용하여 개별적으로 또는 함께 합성하고 배포할 수 있습니다. 일반적으로 프로젝트를 실행할 때는 프로젝트의 기본 디렉터리에 있어야 합니다.

- `cdk synth`: 앱에 있는 하나 이상의 스택에서 AWS CloudFormation 템플릿을 합성합니다 AWS CDK .
- `cdk deploy`: 앱에 있는 하나 이상의 스택에서 정의한 리소스를 에 배포합니다. AWS CDK AWS

단일 명령으로 합성하거나 배포할 여러 스택의 이름을 지정할 수 있습니다. 앱이 스택을 하나만 정의하는 경우 스택을 지정할 필요가 없습니다.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

와일드카드* (원하는 수의 문자) 및? 를 사용할 수도 있습니다. (임의의 단일 문자) 를 사용하여 패턴별로 스택을 식별할 수 있습니다. 와일드카드를 사용할 때는 패턴을 따옴표로 묶으십시오. 그렇지 않으면 셸이 툴킷으로 전달되기 전에 현재 디렉토리의 파일 이름으로 확장하려고 할 수 있습니다. AWS CDK

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> 스택을 배포하기 전에 명시적으로 합성할 필요는 없습니다. 이 단계를 `cdk deploy` 수행하면 최신 코드가 배포되도록 할 수 있습니다.

명령의 전체 설명서를 보려면 을 참조하십시오. `cdk` the section called “AWS CDK 툴킷”

AWS CDK Java에서 작업하기

Java는 에서 완전히 지원되는 클라이언트 AWS CDK 언어이며 안정적인 것으로 간주됩니다. JDK (오라클 또는 Amazon Corretto와 같은 OpenJDK 배포판) 및 Apache Maven을 비롯한 친숙한 도구를 사용하여 Java로 AWS CDK 애플리케이션을 개발할 수 있습니다.

는 Java 8 이상을 지원합니다. AWS CDK 그러나 최신 버전의 언어에는 AWS CDK 응용 프로그램 개발에 특히 편리한 개선 사항이 포함되어 있으므로 가능한 최신 버전을 사용하는 것이 좋습니다. 예를 들어 Java 9에는 객체 리터럴로 작성되는 해시맵을 선언하는 편리한 방법인 `Map.of()` 메서드가 도입되었습니다. TypeScript Java 10에서는 키워드를 사용한 로컬 형식 추론을 도입합니다. `var`

> **Note**
>
> 이 개발자 안내서에 있는 대부분의 코드 예제는 Java 8에서 작동합니다. 몇 가지 예제에서는 를 사용합니다`Map.of()`. 이 예제에는 Java 9가 필요하다는 설명이 포함되어 있습니다.

Maven 프로젝트를 읽을 수 있는 모든 텍스트 편집기 또는 Java IDE를 사용하여 AWS CDK 앱에서 작업할 수 있습니다. 이 가이드에서는 Eclipse 힌트를 제공하지만 IntelliJ IDEA 및 기타 IDE는 Maven 프로젝트를 가져올 수 있으며 Java로 응용 프로그램을 개발하는 데 사용할 수 있습니다. NetBeans AWS CDK

Java가 아닌 JVM 호스팅 언어 (예: Kotlin, Groovy, Clojure 또는 Scala) 로 AWS CDK 애플리케이션을 작성할 수 있지만, 사용 경험이 특별히 까다롭지 않을 수 있으므로 이러한 언어에 대한 지원을 제공할 수 없습니다.

주제

- Java 시작하기
- 프로젝트 생성
- AWS 구성 라이브러리 모듈 관리
- 의 종속성 관리 Java
- AWS CDK 자바 관용구
- 빌드, 합성, 배포

Java 시작하기

를 사용하려면 계정과 자격 증명이 있어야 AWS CDK하며 Node.js 및 툴킷이 설치되어 있어야 합니다. AWS AWS CDK 시작하기 AWS CDK 섹션을 참조하십시오.

자바 AWS CDK 애플리케이션에는 Java 8 (v1.8) 이상이 필요합니다. Amazon Corretto를 권장하지만 모든 OpenJDK 배포판이나 오라클의 JDK를 사용할 수 있습니다. 아파치 메이븐 3.5 이상도 필요합니다. Gradle과 같은 도구를 사용할 수도 있지만 툴킷에서 생성되는 애플리케이션 스켈레톤은 Maven 프로젝트입니다. AWS CDK

> **Note**
>
> 타사 언어 지원 중단: 언어 버전은 공급업체 또는 커뮤니티에서 EOL (End Of Life) 을 공유할 때까지만 지원되며 사전 통지로 변경될 수 있습니다.

프로젝트 생성

빈 `cdk init` 디렉터리에서 호출하여 새 AWS CDK 프로젝트를 생성합니다. `--language`옵션을 사용하고 다음을 지정합니다`java`.

```
mkdir my-project
cd my-project
cdk init app --language java
```

`cdk init`프로젝트 폴더 이름을 사용하여 클래스, 하위 폴더, 파일 등 프로젝트의 다양한 요소에 이름을 지정합니다. 폴더 이름의 하이픈은 밑줄로 변환됩니다. 하지만 이름은 Java 식별자 형식을 따라야 합니다. 예를 들어, 숫자로 시작하거나 공백을 포함해서는 안 됩니다.

결과 프로젝트에는 `software.amazon.awscdk` Maven 패키지에 대한 참조가 포함됩니다. Maven은 이 프로젝트와 해당 종속성을 자동으로 설치합니다.

IDE를 사용하는 경우 이제 프로젝트를 열거나 가져올 수 있습니다. 예를 들어 Eclipse에서는 [파일] > [가져오기] > [Maven] > [기존 Maven 프로젝트] 를 선택합니다. 프로젝트 설정이 Java 8 (1.8) 을 사용하도록 설정되어 있는지 확인하십시오.

AWS 구성 라이브러리 모듈 관리

Maven을 사용하여 그룹에 `software.amazon.awscdk` 있는 AWS 구성 라이브러리 패키지를 설치합니다. 대부분의 구문은 아티팩트에 있으며`aws-cdk-lib`, 이 아티팩트는 기본적으로 새 Java 프로젝

트에 추가됩니다. 더 높은 수준의 CDK 지원이 아직 개발 중인 서비스의 모듈은 별도의 "실험용" 패키지에 들어 있으며, 해당 서비스 이름의 짧은 버전 (no 또는 AWS Amazon 접두사) 으로 이름이 지정됩니다. Maven 중앙 리포지토리를 검색하여 모든 AWS CDK 이름을 찾고 모듈 라이브러리를 구성하십시오. AWS

> **Note**
>
> CDK API 레퍼런스의 자바 에디션에도 패키지 이름이 나와 있습니다.

일부 서비스의 AWS 구성 라이브러리 지원은 둘 이상의 네임스페이스에서 제공됩니다. 예를 들어, Amazon Route 53의 기능은`software.amazon.awscdk.route53`,, `route53-patterns`로 구분되어 `route53resolver`, `route53-targets` 있습니다.

기본 AWS CDK 패키지는 Java 코드로 다음과 같이 가져옵니다`software.amazon.awscdk`. AWS 구성 라이브러리의 다양한 서비스에 대한 모듈은 Maven 패키지 이름에 `software.amazon.awscdk.services` 포함되며 이름은 Maven 패키지 이름과 비슷합니다. 예를 들어, Amazon S3 모듈의 네임스페이스는 입니다. `software.amazon.awscdk.services.s3`

각 Java 소스 파일에서 사용하는 AWS Construct Library 클래스마다 별도의 Java `import` 문을 작성하고 와일드카드를 가져오지 않는 것이 좋습니다. 명령문 없이도 언제든지 형식의 정규화된 이름 (네임스페이스 포함) 을 사용할 수 있습니다. `import`

애플리케이션이 실험용 패키지를 사용하는 경우 프로젝트를 `pom.xml` 편집하고 컨테이너에 새 <dependency> 요소를 추가하세요. <dependencies> 예를 들어, 다음 <dependency> 요소는 CodeStar 실험적 구성 라이브러리 모듈을 지정합니다.

```
<dependency>
    <groupId>software.amazon.awscdk</groupId>
    <artifactId>codestar-alpha</artifactId>
    <version>2.0.0-alpha.10</version>
</dependency>
```

> **Tip**
>
> Java IDE를 사용하는 경우 아마도 Maven 종속성을 관리하는 기능이 있을 것입니다. 하지만 IDE의 기능이 `pom.xml` 직접 수행하는 작업과 일치한다는 확신이 들지 않는 한 직접 편집하는 것이 좋습니다.

의 종속성 관리 Java

Java에서는 Maven을 사용하여 종속성을 pom.xml 지정하고 설치합니다. <dependencies>컨테이너에는 각 패키지의 <dependency> 요소가 포함되어 있습니다. 다음은 일반적인 CDK Java 앱에 pom.xml 대한 섹션입니다.

```
<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**
>
> 많은 Java IDE에는 Maven 지원 및 시각적 pom.xml 편집기가 통합되어 있어 종속성 관리에 편리할 수 있습니다.

Maven은 종속성 잠금을 지원하지 않습니다. 에서 버전 범위를 지정할 수 있지만 빌드를 pom.xml 반복 가능하게 유지하려면 항상 정확한 버전을 사용하는 것이 좋습니다.

Maven은 전이적 종속성을 자동으로 설치하지만 각 패키지에는 하나의 사본만 설치할 수 있습니다. POM 트리에서 가장 높게 지정된 버전이 선택되며, 설치되는 패키지 버전의 최종 결정은 항상 응용 프로그램에 표시됩니다.

Maven은 프로젝트를 빌드 (mvn compile) 또는 패키지 (mvn package) 할 때마다 종속성을 자동으로 설치하거나 업데이트합니다. CDK 툴킷은 실행할 때마다 이 작업을 자동으로 수행하므로 일반적으로 Maven을 수동으로 호출할 필요가 없습니다.

AWS CDK 자바 관용구

소품

모든 AWS Construct Library 클래스는 구문이 정의되는 범위 (구성 트리의 부모), id, props라는 세 가지 인수를 사용하여 인스턴스화됩니다. props는 구문이 생성하는 리소스를 구성하는 데 사용하는 키/값 쌍의 번들입니다. 다른 클래스와 메서드에서도 인수에 “속성 번들” 패턴을 사용합니다.

Java에서 props는 Builder 패턴을 사용하여 표현됩니다. 각 구문 유형에는 해당하는 props 유형이 있습니다. 예를 들어, `Bucket` 구문 (Amazon S3 버킷을 나타냄) 은 인스턴스를 props로 사용합니다. `BucketProps`

`BucketProps`클래스 (모든 AWS Construct Library props 클래스와 마찬가지로) 에는 라는 내부 클래스가 있습니다. `Builder` `BucketProps.Builder`형식은 `BucketProps` 인스턴스의 다양한 속성을 설정하는 메서드를 제공합니다. 각 메서드는 `Builder` 인스턴스를 반환하므로 메서드 호출을 연결하여 여러 속성을 설정할 수 있습니다. 체인의 끝에서 `BucketProps` 객체를 실제로 `build()` 생성하기 위해 호출합니다.

```
Bucket bucket = new Bucket(this, "MyBucket", new BucketProps.Builder()
                           .versioned(true)
                           .encryption(BucketEncryption.KMS_MANAGED)
                           .build());
```

props와 유사한 객체를 최종 인수로 사용하는 구문 및 기타 클래스는 단축키를 제공합니다. 클래스에는 해당 클래스와 해당 `Builder` props 객체를 한 번에 인스턴스화하는 자체 클래스가 있습니다. 이렇게 하면 `BucketProps` `Bucket` a와 —를 모두 명시적으로 인스턴스화할 필요가 없고 props 유형을 임포트할 필요도 없습니다.

```
Bucket bucket = Bucket.Builder.create(this, "MyBucket")
                           .versioned(true)
                           .encryption(BucketEncryption.KMS_MANAGED)
                           .build();
```

기존 구문에서 자체 구문을 파생하는 경우 추가 속성을 허용하는 것이 좋습니다. 다음과 같은 빌더 패턴을 따르는 것이 좋습니다. 하지만 이는 구체 클래스를 서브클래싱하는 것만큼 간단하지 않습니다. 두 개의 새 `Builder` 클래스에서 움직이는 부분을 직접 제공해야 합니다. 구문에 하나 이상의 추가 인수를 받아들이는 것이 더 나을 수도 있습니다. 인수가 선택사항인 경우 추가 생성자를 제공해야 합니다.

제네릭 구조

일부 API에서는 JavaScript 배열이나 유형이 지정되지 않은 객체를 메서드에 대한 입력으로 AWS CDK 사용합니다. (예를 들어 AWS CodeBuild의 `BuildSpec.fromObject()`메서드를 참조하십시오.) Java에서 이러한 객체는 다음과 같이 표현됩니다`java.util.Map<String, Object>`. 값이 모두 문자열인 경우 사용할 수 있습니다`Map<String, String>`.

Java는 다른 언어처럼 이러한 컨테이너에 대한 리터럴을 작성하는 방법을 제공하지 않습니다. Java 9 이상에서는 이러한 호출 중 하나를 `java.util.Map.of()`사용하여 최대 10개 항목의 맵을 인라인으로 편리하게 정의할 수 있습니다.

```
java.util.Map.of(
    "base-directory", "dist",
    "files", "LambdaStack.template.json"
 )
```

항목이 10개 이상인 맵을 생성하려면 를 사용하십시오 `java.util.Map.ofEntries()`.

Java 8을 사용하는 경우 이와 유사한 자체 메서드를 제공할 수 있습니다.

JavaScript 배열은 Java로 `List<Object>` 또는 `List<String>` 형식으로 표시됩니다. 이 `java.util.Arrays.asList` 메서드는 짧은 `List` s를 정의하는 데 편리합니다.

```
List<String> cmds = Arrays.asList("cd lambda", "npm install", "npm install typescript")
```

누락된 값

Java에서 props와 같은 AWS CDK 객체의 누락된 값은 로 `null` 표시됩니다. 가능한 모든 값을 명시적으로 `null` 테스트하여 값을 다루기 전에 값이 포함되어 있는지 확인해야 합니다. Java에는 다른 언어처럼 null 값을 처리하는 데 도움이 되는 "문법"이 없습니다. 상황에 따라 ObjectUtil Apache가 firstNonNull유용할 수도 defaultIfNull있습니다. 또는 null이 될 수 있는 값을 더 쉽게 처리하고 코드를 더 읽기 쉽게 만들 수 있는 정적 도우미 메서드를 직접 작성해 보세요.

빌드, 합성, 배포

는 앱을 실행하기 전에 AWS CDK 자동으로 컴파일합니다. 하지만 앱을 수동으로 빌드하여 오류를 확인하고 테스트를 실행하는 것이 유용할 수 있습니다. IDE에서 (예: Eclipse에서 Control-B 누르기) 하거나 프로젝트의 루트 디렉터리에 있는 동안 명령 프롬프트에서 명령을 실행하여 이 작업을 수행할 수 있습니다. `mvn compile`

작성한 모든 테스트를 명령 `mvn test` 프롬프트에서 실행하여 실행하십시오.

AWS CDK 앱에 정의된 스택은 아래 명령어를 사용하여 개별적으로 또는 함께 합성하고 배포할 수 있습니다. 일반적으로 프로젝트를 실행할 때는 프로젝트의 기본 디렉터리에 있어야 합니다.

- `cdk synth`: 앱에 있는 하나 이상의 스택에서 AWS CloudFormation 템플릿을 합성합니다 AWS CDK .
- `cdk deploy`: 앱에 있는 하나 이상의 스택으로 정의된 리소스를 에 배포합니다. AWS CDK AWS

단일 명령으로 합성하거나 배포할 여러 스택의 이름을 지정할 수 있습니다. 앱이 스택을 하나만 정의하는 경우 스택을 지정할 필요가 없습니다.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

와일드카드* (원하는 수의 문자) 및? 를 사용할 수도 있습니다. (임의의 단일 문자) 를 사용하여 패턴별로 스택을 식별할 수 있습니다. 와일드카드를 사용할 때는 패턴을 따옴표로 묶으십시오. 그렇지 않으면 셸이 툴킷으로 전달되기 전에 현재 디렉토리의 파일 이름으로 확장하려고 할 수 있습니다. AWS CDK

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> 스택을 배포하기 전에 명시적으로 합성할 필요는 없습니다. 이 단계를 `cdk deploy` 수행하면 최신 코드가 배포되도록 할 수 있습니다.

명령의 전체 설명서를 보려면 을 참조하십시오. `cdk` the section called “AWS CDK 툴킷”

C AWS CDK #에서 작업하기

.NET은 에서 완전히 지원되는 클라이언트 AWS CDK 언어이며 안정적인 것으로 간주됩니다. C #은 예제와 지원을 제공하는 주요 .NET 언어입니다. Visual Basic 또는 F #과 같은 다른.NET 언어로 AWS CDK 응용 프로그램을 작성할 수 있지만 AWS CDK에서 이러한 언어를 사용하는 것은 제한적으로 지원됩니다.

Visual Studio, Visual Studio Code, `dotnet` 명령, 패키지 관리자 등의 친숙한 도구를 사용하여 C #으로 AWS CDK 응용 프로그램을 개발할 수 있습니다. NuGet AWS 구성 라이브러리를 구성하는 모듈은 nuget.org를 통해 배포됩니다.

C #으로 AWS CDK 앱을 개발하려면 Windows용 Visual Studio 2019 (모든 에디션) 를 사용하는 것이 좋습니다.

주제

- C# 시작하기
- 프로젝트 생성
- AWS 구성 라이브러리 모듈 관리
- 에서 종속성 관리 C#
- AWS CDK C #의 관용구
- 빌드, 합성 및 배포

C# 시작하기

를 사용하려면 AWS 계정과 자격 증명이 있어야 AWS CDK하며 Node.js 및 AWS CDK 툴킷이 설치되어 있어야 합니다. 시작하기 AWS CDK 섹션을 참조하십시오.

C# AWS CDK 응용 프로그램에는 여기에서 사용할 수 있는.NET Core v3.1 이상이 필요합니다.

.NET 도구 모음에는.NET `dotnet` 응용 프로그램을 빌드 및 실행하고 패키지를 관리하기 위한 명령줄 도구가 포함되어 있습니다. NuGet 주로 Visual Studio에서 작업하는 경우에도 이 명령은 배치 작업과 구성 라이브러리 패키지 설치에 유용할 수 있습니다. AWS

프로젝트 생성

빈 `cdk init` 디렉터리에서 호출하여 새 AWS CDK 프로젝트를 만듭니다. `--language`옵션을 사용하고 다음을 지정합니다`csharp`.

```
mkdir my-project
cd my-project
cdk init app --language csharp
```

`cdk init`프로젝트 폴더 이름을 사용하여 클래스, 하위 폴더, 파일 등 프로젝트의 다양한 요소에 이름을 지정합니다. 폴더 이름의 하이픈은 밑줄로 변환됩니다. 하지만 이름은 C# 식별자 형식을 따라야 합니다. 예를 들어, 숫자로 시작하거나 공백을 포함해서는 안 됩니다.

결과 프로젝트에는 Amazon.CDK.Lib NuGet 패키지에 대한 참조가 포함됩니다. 에 의해 NuGet 프로젝트 및 해당 종속성이 자동으로 설치됩니다.

AWS 구성 라이브러리 모듈 관리

.NET 에코시스템은 NuGet 패키지 관리자를 사용합니다. 핵심 클래스와 모든 안정적인 서비스 구조를 포함하는 기본 CDK 패키지는 입니다. Amazon.CDK.Lib 새로운 기능이 활발히 개발 중인 실험용 모듈의 이름은 다음과 같이 Amazon.CDK.AWS.*SERVICE-NAME*.Alpha 지정됩니다. 여기서 서비스 이름은 AWS or Amazon 접두사가 없는 짧은 이름입니다. 예를 들어, AWS IoT 모듈의 NuGet 패키지 이름은 입니다Amazon.CDK.AWS.IoT.Alpha. 원하는 패키지를 찾을 수 없는 경우 NuGet.org를 검색하십시오.

> **Note**
>
> CDK API 레퍼런스의 .NET 에디션에는 패키지 이름도 나와 있습니다.

일부 서비스의 AWS 구성 라이브러리 지원은 둘 이상의 모듈에서 제공됩니다. 예를 들어, 라는 이름의 Amazon.CDK.AWS.IoT.Actions.Alpha 두 번째 AWS IoT 모듈이 있습니다.

대부분의 AWS CDK 앱에 필요한 AWS CDK의 기본 모듈은 C# 코드로 다음과 같이 Amazon.CDK 가져옵니다. AWS 구성 라이브러리의 다양한 서비스에 대한 모듈은 아래에 Amazon.CDK.AWS 있습니다. 예를 들어, Amazon S3 모듈의 네임스페이스는 입니다. Amazon.CDK.AWS.S3

CDK 코어 구조 및 각 C# 소스 파일에서 사용하는 각 AWS 서비스에 대해 C# using 지시문을 작성하는 것이 좋습니다. 이름 충돌 해결에 도움이 되도록 네임스페이스나 형식에 별칭을 사용하는 것이 편리할 수도 있습니다. 명령문 없이도 언제든지 형식의 정규화된 이름 (네임스페이스 포함) 을 사용할 수 있습니다. using

에서 종속성 관리 C#

C# AWS CDK 앱에서는 를 사용하여 종속성을 관리합니다. NuGet NuGet 대부분 동등한 표준 인터페이스 4개가 있습니다. 필요와 작업 스타일에 맞는 것을 사용하세요. Paket과 같은 호환 가능한 도구를 MyGet사용하거나 .csproj 파일을 직접 편집할 수도 있습니다.

NuGet 종속성에 대한 버전 범위를 지정할 수 없습니다. 모든 종속성은 특정 버전에 고정됩니다.

종속성을 업데이트하면 Visual Studio는 다음에 NuGet 빌드할 때 각 패키지의 지정된 버전을 검색하는 데 사용합니다. Visual Studio를 사용하지 않는 경우 dotnet restore 명령을 사용하여 종속성을 업데이트하세요.

프로젝트 파일을 직접 편집하기

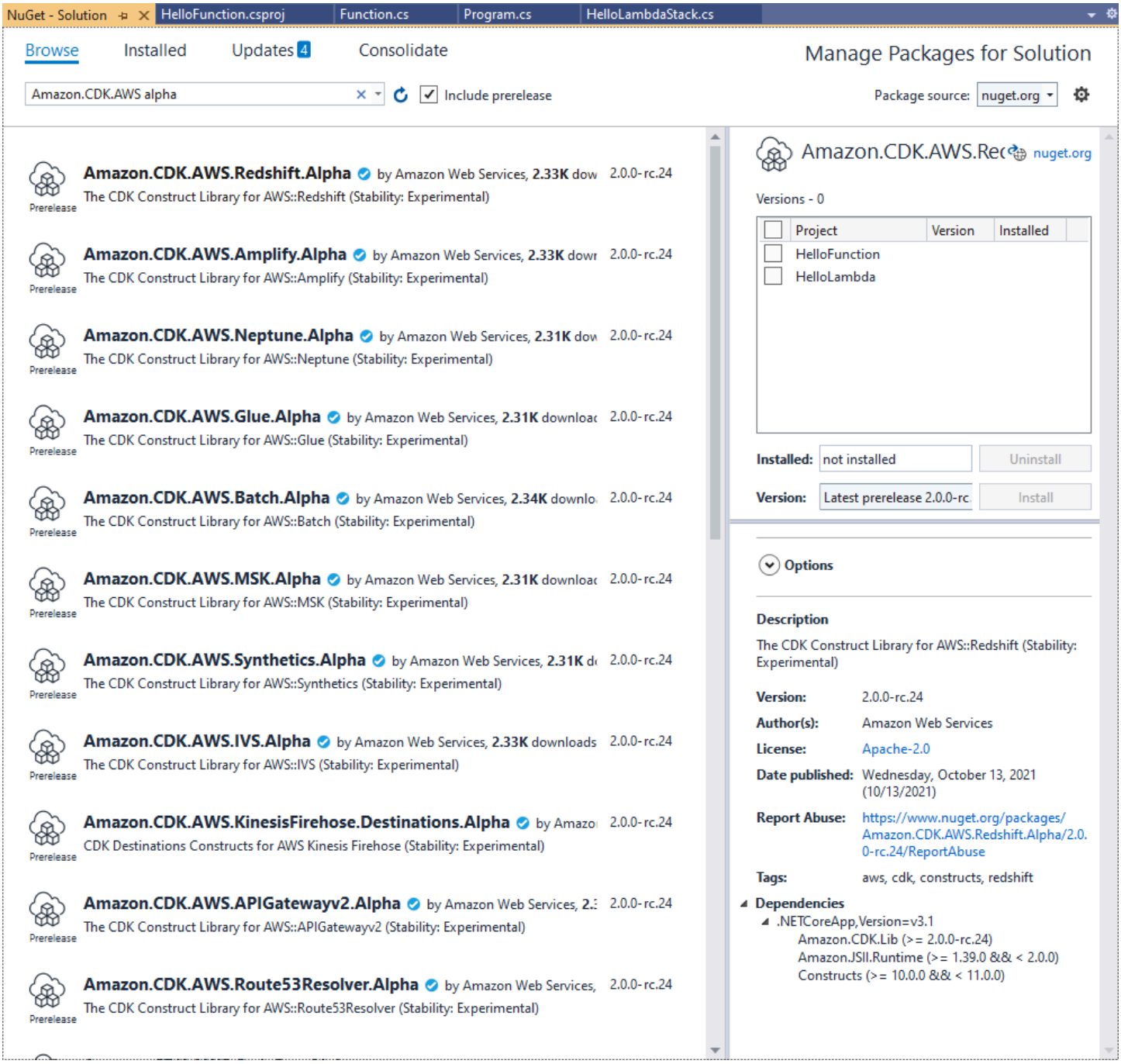
프로젝트 `.csproj` 파일에는 종속성을 `<PackageReference` 요소로 나열하는 `<ItemGroup>` 컨테이너가 포함되어 있습니다.

```
<ItemGroup>
    <PackageReference Include="Amazon.CDK.Lib" Version="2.14.0" />
    <PackageReference Include="Constructs" Version="%constructs-version%" />
</ItemGroup>
```

비주얼 스튜디오 NuGet GUI

Visual Studio의 NuGet 도구는 도구 > NuGet 패키지 관리자 > 솔루션용 NuGet 패키지 관리에서 액세스할 수 있습니다. 찾아보기 탭을 사용하여 설치하려는 AWS 구성 라이브러리 패키지를 찾을 수 있습니다. 모듈의 프리릴리즈 버전을 포함하여 원하는 버전을 선택하고 열려 있는 모든 프로젝트에 추가할 수 있습니다.

> **Note**
>
> “실험용”으로 간주되는 모든 AWS Construct Library 모듈 (참조the section called “버전 관리”)은 에서 NuGet 프리릴리즈로 플래그가 지정되며 이름 접미사가 붙습니다. `alpha`



업데이트 페이지에서 새 버전의 패키지를 설치하세요.

NuGet 콘솔

NuGet 콘솔은 Visual Studio 프로젝트의 컨텍스트에서 NuGet 작동하는 PowerShell 기반 인터페이스입니다. Visual Studio에서 도구 > NuGet 패키지 관리자 > 패키지 관리자 콘솔을 선택하여 열 수 있습니다. 이 도구를 사용하는 방법에 대한 자세한 내용은 Visual Studio의 패키지 관리자 콘솔을 사용하여 패키지 설치 및 관리를 참조하십시오.

`dotnet`명령:

이 `dotnet` 명령은 Visual Studio C# 프로젝트 작업을 위한 기본 명령줄 도구입니다. 모든 Windows 명령 프롬프트에서 이 명령을 호출할 수 있습니다. 다양한 기능 중에서 Visual Studio NuGet 프로젝트에 종속성을 추가할 `dotnet` 수 있습니다.

Visual Studio 프로젝트 (`.csproj`) 파일과 동일한 디렉터리에 있다고 가정하고 다음과 같은 명령을 실행하여 패키지를 설치합니다. 프로젝트를 만들 때 기본 CDK 라이브러리가 포함되므로 실험용 모듈만 명시적으로 설치하면 됩니다. 시험용 모듈을 사용하려면 명시적인 버전 번호를 지정해야 합니다.

```
dotnet add package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

다른 디렉터리에서 명령을 실행할 수 있습니다. 이렇게 하려면 `add` 키워드 뒤에 프로젝트 파일 또는 프로젝트 파일이 들어 있는 디렉터리의 경로를 포함하세요. 다음 예제에서는 사용자가 AWS CDK 프로젝트의 기본 디렉터리에 있다고 가정합니다.

```
dotnet add src/PROJECT-DIR package Amazon.CDK.AWS.IoT.Alpha -v VERSION-NUMBER
```

특정 버전의 패키지를 설치하려면 `-v` 플래그와 원하는 버전을 포함해야 합니다.

패키지를 업데이트하려면 패키지를 설치할 때 사용한 것과 동일한 `dotnet add` 명령을 실행합니다. 실험용 모듈의 경우에도 명시적인 버전 번호를 지정해야 합니다.

`dotnet`명령을 사용하여 패키지를 관리하는 방법에 대한 자세한 내용은 dotnet CLI를 사용한 패키지 설치 및 관리를 참조하십시오.

커맨드 `nuget`

`nuget`명령줄 도구는 NuGet 패키지를 설치하고 업데이트할 수 있습니다. 하지만 Visual Studio 프로젝트를 프로젝트 설정 방식과 `cdk init` 다르게 설정해야 합니다. (기술적 세부 정보: `Packages.config` 프로젝트와 함께 `nuget` 작동하면서 새로운 스타일의 `PackageReference` 프로젝트를 `cdk init` 만듭니다.)

에서 만든 AWS CDK 프로젝트에는 이 `nuget` 도구를 사용하지 않는 것이 좋습니다. `cdk init` 다른 유형의 프로젝트를 사용 `nuget` 중이고 사용하려면 NuGet CLI 참조를 참조하십시오.

AWS CDK C #의 관용구

소품

모든 AWS Construct Library 클래스는 구문이 정의되는 범위 (구성 트리의 부모), id, props라는 세 가지 인수를 사용하여 인스턴스화됩니다. props는 구문이 생성하는 리소스를 구성하는 데 사용하는 키/값 쌍의 번들입니다. 다른 클래스와 메서드에서도 인수에 “속성 번들” 패턴을 사용합니다.

C #에서 props는 props 형식을 사용하여 표현됩니다. 관용적인 C# 방식으로 객체 이니셜라이저를 사용하여 다양한 속성을 설정할 수 있습니다. 여기서는 구문을 사용하여 Amazon S3 버킷을 `Bucket` 생성합니다. 해당 props 유형은 입니다`BucketProps`.

```
var bucket = new Bucket(this, "MyBucket", new BucketProps {
    Versioned = true
});
```

> **Tip**
>
> 패키지를 프로젝트에 추가하면 `Amazon.JSII.Analyzers` Visual Studio 내의 소품 정의에서 필수 값을 확인할 수 있습니다.

클래스를 확장하거나 메서드를 재정의할 때 부모 클래스에서는 이해할 수 없는 추가 소품을 원하는 용도로 사용할 수 있습니다. 이렇게 하려면 적절한 props 유형을 서브클래싱하고 새 속성을 추가해야 합니다.

```
// 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"
});
```

부모 클래스의 이니셜라이저나 오버라이드된 메서드를 호출할 때는 일반적으로 수신한 props를 전달할 수 있습니다. 새 형식은 상위 형식과 호환되며 추가한 추가 소품은 무시됩니다.

향후 릴리스에서 우연히 귀하의 재산에 사용한 이름을 가진 새 부동산이 AWS CDK 추가될 수 있습니다. 이렇게 해도 구문이나 메서드를 사용할 때 기술적인 문제는 발생하지 않지만 (속성이 "체인 위로" 전달되지 않기 때문에 부모 클래스나 오버라이드된 메서드는 단순히 기본값을 사용함) 구문 사용자에게 혼란을 줄 수 있습니다. 속성의 이름을 지정하여 구문에 명확하게 속하도록 하면 이러한 잠재적 문제를 피할 수 있습니다. 새 속성이 많으면 해당 속성을 적절한 이름의 클래스로 묶어 단일 속성으로 전달하십시오.

제네릭 구조체

일부 API에서는 JavaScript 배열이나 유형이 지정되지 않은 객체를 메서드에 대한 입력으로 AWS CDK 사용합니다. (예를 들어 AWS CodeBuild의 `BuildSpec.fromObject()`메서드를 참조하십시오.) C #에서 이러한 객체는 다음과 같이 `System.Collections.Generic.Dictionary<String, Object>` 표현됩니다. 값이 모두 문자열인 경우 사용할 `Dictionary<String, String>` 수 있습니다. JavaScript 배열은 C #에서 `object[]` 또는 `string[]` 배열 유형으로 표시됩니다.

> **Tip**
>
> 이러한 특정 사전 형식을 보다 쉽게 사용할 수 있도록 짧은 별칭을 정의할 수 있습니다.
>
> ```
> using StringDict = System.Collections.Generic.Dictionary<string, string>;
> using ObjectDict = System.Collections.Generic.Dictionary<string, object>;
> ```

누락된 값

C #에서 props와 같은 AWS CDK 객체의 누락된 값은 로 표시됩니다. `null` 이러한 값을 사용할 때는 null 조건부 멤버 액세스 `?.` 연산자와 null 병합 연산자를 사용하면 편리합니다. `??`

```
// 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";
```

빌드, 합성 및 배포

는 앱을 실행하기 전에 AWS CDK 자동으로 컴파일합니다. 하지만 앱을 수동으로 빌드하여 오류를 확인하고 테스트를 실행하는 것이 유용할 수 있습니다. Visual Studio에서 F6을 누르거나 `dotnet build src` 명령줄에서 명령을 실행하여 이 작업을 수행할 수 있습니다. 여기서 `src` 는 Visual Studio 솔루션 (`.sln`) 파일이 포함된 프로젝트 디렉터리의 디렉터리입니다.

AWS CDK 앱에 정의된 스택은 아래 명령을 사용하여 개별적으로 또는 함께 합성하고 배포할 수 있습니다. 일반적으로 프로젝트를 실행할 때는 프로젝트의 기본 디렉터리에 있어야 합니다.

- `cdk synth`: 앱에 있는 하나 이상의 스택에서 AWS CloudFormation 템플릿을 합성합니다 AWS CDK .
- `cdk deploy`: 앱에 있는 하나 이상의 스택에서 정의한 리소스를 에 배포합니다. AWS CDK AWS

단일 명령으로 합성하거나 배포할 여러 스택의 이름을 지정할 수 있습니다. 앱이 스택을 하나만 정의하는 경우 스택을 지정할 필요가 없습니다.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

와일드카드* (원하는 수의 문자) 및? 를 사용할 수도 있습니다. (임의의 단일 문자) 를 사용하여 패턴별로 스택을 식별할 수 있습니다. 와일드카드를 사용할 때는 패턴을 따옴표로 묶으십시오. 그렇지 않으면 셸이 툴킷으로 전달되기 전에 현재 디렉토리의 파일 이름으로 확장하려고 할 수 있습니다. AWS CDK

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> 스택을 배포하기 전에 명시적으로 합성할 필요는 없습니다. 이 단계를 `cdk deploy` 수행하면 최신 코드가 배포되도록 할 수 있습니다.

명령의 전체 설명서를 보려면 을 참조하십시오. `cdk` the section called “AWS CDK 툴킷”

AWS CDK Go에서 작업하기

Go는 에서 완전히 지원되는 클라이언트 AWS Cloud Development Kit (AWS CDK) 언어이며 안정적인 것으로 간주됩니다. AWS CDK Go에서 작업하려면 익숙한 도구를 사용합니다. Go 버전의 이벤트에서는 AWS CDK Go 스타일 식별자를 사용합니다.

CDK가 지원하는 다른 언어와 달리 Go는 전통적인 객체 지향 프로그래밍 언어가 아닙니다. Go는 컴포지션을 사용하지만 다른 언어는 종종 상속을 활용합니다. 가능한 한 관용적 Go를 사용하려고 노력했지만 CDK가 다를 수 있는 부분이 있습니다.

이 항목에서는 Go를 사용하여 작업할 때의 지침을 제공합니다. AWS CDK 를 위한 간단한 Go 프로젝트에 대한 설명은 발표 블로그 게시물을 참조하십시오. AWS CDK

주제

Go 시작하기

를 사용하려면 AWS 계정과 자격 증명이 있어야 AWS CDK하며 Node.js 및 툴킷이 설치되어 있어야 합니다. AWS CDK 시작하기 AWS CDK 섹션을 참조하십시오.

Go 바인딩은 표준 Go 툴체인 v1.18 이상을 AWS CDK 사용합니다. 원하는 편집기를 사용할 수 있습니다.

> **Note**
>
> 타사 언어 지원 중단: 언어 버전은 공급업체 또는 커뮤니티에서 EOL (End Of Life) 을 공유할 때까지만 지원되며 사전 통지를 통해 변경될 수 있습니다.

프로젝트 생성

빈 `cdk init` 디렉터리에서 호출하여 새 AWS CDK 프로젝트를 생성합니다. `--language`옵션을 사용하고 다음을 지정합니다go.

```
mkdir my-project
cd my-project
cdk init app --language go
```

`cdk init`프로젝트 폴더 이름을 사용하여 클래스, 하위 폴더, 파일 등 프로젝트의 다양한 요소에 이름을 지정합니다. 폴더 이름의 하이픈은 밑줄로 변환됩니다. 하지만 이름은 Go 식별자 형식을 따라야 합니다. 예를 들어, 숫자로 시작하거나 공백을 포함해서는 안 됩니다.

결과 프로젝트에는 핵심 AWS CDK Go `github.com/aws/aws-cdk-go/awscdk/v2` 모듈인 in에 대한 참조가 포함됩니다go.mod. 이 모듈과 기타 필수 모듈을 설치하는 go get 데 문제가 있습니다.

AWS 구성 라이브러리 모듈 관리

대부분의 AWS CDK 문서 및 예제에서 “모듈”이라는 단어는 종종 AWS 서비스당 하나 이상의 AWS Construct Library 모듈을 가리키는 데 사용되는데, 이는 Go에서 사용하는 관용적 용어와는 다릅니다. CDK 구성 라이브러리는 다양한 AWS 서비스를 지원하는 개별 구성 라이브러리 모듈과 함께 하나의 Go 모듈로 제공되며, 이 모듈은 해당 모듈 내에서 Go 패키지로 제공됩니다.

일부 서비스의 AWS 구성 라이브러리 지원은 둘 이상의 구성 라이브러리 모듈 (Go 패키지)에서 제공됩니다. 예를 들어 Amazon Route 53에는 기본 `awsroute53` 패키지 외에도 이름이`awsroute53patterns`,, 인 세 개의 구성 라이브러리 모듈이 `awsroute53targets` 있습니다. `awsroute53resolver`

대부분의 AWS CDK 앱에 필요한 AWS CDK의 핵심 패키지는 Go 코드로 다음과 같이 가져옵니다`github.com/aws/aws-cdk-go/awscdk/v2`. AWS 구성 라이브러리의 다양한 서비스를 위한 패키지는 아래에 `github.com/aws/aws-cdk-go/awscdk/v2` 있습니다. 예를 들어, Amazon S3 모듈의 네임스페이스는 입니다. `github.com/aws/aws-cdk-go/awscdk/v2/awss3`

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

앱에서 사용하려는 서비스의 Construct Library 모듈 (Go 패키지) 을 가져온 후에는 예를 들어 를 사용하여 해당 모듈의 구문에 액세스할 수 있습니다. `awss3.Bucket`

종속성 관리: Go

Go에서는 종속성 버전이 에서 정의됩니다. `go.mod` `go.mod`기본값은 여기에 표시된 것과 비슷합니다.

```
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
)
```

패키지 이름 (Go 용어로 모듈) 은 필수 버전 번호가 추가된 URL로 지정됩니다. Go의 모듈 시스템은 버전 범위를 지원하지 않습니다.

go get명령을 실행하여 모든 필수 모듈을 설치하고 `go.mod` 업데이트합니다. 종속 항목에 사용할 수 있는 업데이트 목록을 보려면 를 실행하십시오. go list -m -u all

AWS CDK Go의 관용구

필드 및 메서드 이름

필드 및 메서드 이름은 CDK의 기원 언어인 TypeScript 카멜 대소문자 (`likeThis`) 를 사용합니다. Go에서는 이러한 규칙이 Go 규칙을 따르므로 Pascal-cased () 도 마찬가지입니다. `LikeThis`

정리

`main`메서드에서 CDK 앱이 `defer jsii.Close()` 저절로 정리되도록 하는 데 사용하십시오.

누락된 값 및 포인터 변환

Go에서 속성 번들과 같은 AWS CDK 객체의 누락된 값은 로 `nil` 표시됩니다. Go에는 null을 허용하는 형식이 없습니다. 포함할 `nil` 수 있는 형식은 포인터뿐입니다. 값을 선택 사항으로 허용하려면 기본 유형의 경우에도 모든 CDK 속성, 인수 및 반환 값이 포인터가 됩니다. 이는 필수 값뿐만 아니라 선택적 값에도 적용되므로 나중에 필수 값이 선택 사항이 되더라도 유형을 크게 변경할 필요가 없습니다.

리터럴 값이나 표현식을 전달할 때는 다음 도우미 함수를 사용하여 값에 대한 포인터를 만드십시오.

- `jsii.String`
- `jsii.Number`
- `jsii.Bool`
- `jsii.Time`

일관성을 위해 자체 구문을 정의할 때도 마찬가지로 포인터를 사용하는 것이 좋습니다. 예를 들어 구문을 문자열에 대한 포인터보다 문자열로 받는 것이 더 편리해 보일 수도 있습니다. `id`

프리미티브 값뿐만 아니라 복합 유형을 포함한 선택적 AWS CDK 값을 처리할 때는 포인터를 다루기 전에 포인터를 명시적으로 테스트하여 포인터가 올바르지 않은지 확인해야 합니다. `nil` Go에는 다른 언어처럼 비어 있거나 누락된 값을 처리하는 데 도움이 되는 "문법"이 없습니다. 하지만 속성 번들 및 유사한 구조에는 필수 값이 존재하므로 (그렇지 않으면 구성이 실패함) 이러한 값을 검사할 필요가 없습니다. `nil`

구성 및 소품

하나 이상의 AWS 리소스와 관련 속성을 나타내는 구문은 Go에서 인터페이스로 표시됩니다. 예를 들어, `awss3.Bucket` 는 인터페이스입니다. 모든 구문에는 해당하는 인터페이스를 구현하는 구조체를 반환하는 것과 `awss3.NewBucket` 같은 팩토리 함수가 있습니다.

모든 팩토리 함수는 세 개의 인수를 취합니다. 하나는 정의되는 구문 (구성 트리의 부모) `idprops`, 및 및 인수는 구문이 생성하는 리소스를 구성하는 데 사용하는 키/값 쌍의 번들입니다. `scope` "속성 번들" 패턴은 의 다른 곳에서도 사용됩니다. AWS CDK

Go에서 props는 각 구문의 특정 구조체 유형으로 표시됩니다. 예를 들어, an은 유형의 `awss3.Bucket` props 인수를 사용합니다. `awss3.BucketProps` 구조체 리터럴을 사용하여 props 인수를 작성합니다.

```
var bucket = awss3.NewBucket(stack, jsii.String("MyBucket"), &awss3.BucketProps{
    Versioned: jsii.Bool(true),
})
```

제네릭 구조체

일부 지역에서는 JavaScript 배열이나 형식이 지정되지 않은 객체를 메서드의 입력으로 AWS CDK 사용합니다. (예를 들어 AWS CodeBuild의 `BuildSpec.fromObject()`메서드를 참조하십시오.) Go에서 이러한 객체는 각각 슬라이스와 빈 인터페이스로 표시됩니다.

CDK는 기본 유형을 포함하는 슬라이스를 빌드하는 것과 `jsii.Strings` 같은 다양한 도우미 함수를 제공합니다.

```
jsii.Strings("One", "Two", "Three")
```

사용자 지정 구문 개발

Go에서는 일반적으로 기존 구문을 확장하는 것보다 새 구문을 작성하는 것이 더 간단합니다. 먼저 확장과 유사한 시맨틱이 필요한 경우 기존 유형을 하나 이상 익명으로 포함하여 새 구조체 유형을 정의합니다. 추가하려는 새 기능에 맞는 메서드와 필요한 데이터를 보관하는 데 필요한 필드를 작성하세요. 구문에 필요한 경우 props 인터페이스를 정의하세요. 마지막으로, 구문의 인스턴스를 `NewMyConstruct()` 반환하는 팩토리 함수를 작성하세요.

단순히 기존 구문의 일부 디폴트 값을 변경하거나 인스턴스화 시 간단한 동작을 추가하는 경우에는 이러한 모든 설정이 필요하지 않습니다. 대신 "확장"하려는 구문의 팩토리 함수를 호출하는 팩토리 함수를 작성하세요. 예를 들어, 다른 CDK 언어에서는 유형을 재정의하여 Amazon S3 버킷의 객체 유형을 적용하는 `TypedBucket` 구조를 만들고, 새 `s3.Bucket` 유형의 이니셜라이저에서 지정된 파일 이름 확장자만 버킷에 추가할 수 있는 버킷 정책을 추가할 수 있습니다. Go에서는 적절한 버킷 정책을 추가한 `s3.Bucket` (를 사용하여 인스턴스화한`s3.NewBucket`) 를 `NewTypedBucket` 반환하는 코드를 간단히 작성하는 것이 더 쉽습니다. 표준 버킷 구조에서 기능을 이미 사용할 수 있기 때문에 새 구문 유형이 필요하지 않습니다. 새로운 "구조"는 이를 구성하는 더 간단한 방법을 제공하기만 하면 됩니다.

빌드, 합성, 배포

는 앱을 실행하기 전에 AWS CDK 자동으로 컴파일합니다. 하지만 앱을 수동으로 빌드하여 오류를 확인하고 테스트를 실행하는 것이 유용할 수 있습니다. 프로젝트의 루트 디렉터리에 있는 동안 명령 프롬프트에서 명령을 실행하면 이 작업을 수행할 수 있습니다. `go build`

작성한 모든 테스트를 명령 프롬프트에서 실행하여 `go test` 실행하십시오.

AWS CDK 앱에 정의된 스택은 아래 명령어를 사용하여 개별적으로 또는 함께 합성하고 배포할 수 있습니다. 일반적으로 프로젝트를 실행할 때는 프로젝트의 기본 디렉터리에 있어야 합니다.

- `cdk synth`: 앱에 있는 하나 이상의 스택에서 AWS CloudFormation 템플릿을 합성합니다 AWS CDK .
- `cdk deploy`: 앱에 있는 하나 이상의 스택에서 정의한 리소스를 에 배포합니다. AWS CDK AWS

단일 명령으로 합성하거나 배포할 여러 스택의 이름을 지정할 수 있습니다. 앱이 스택을 하나만 정의하는 경우 스택을 지정할 필요가 없습니다.

```
cdk synth                 # app defines single stack
cdk deploy Happy Grumpy   # app defines two or more stacks; two are deployed
```

와일드카드* (원하는 수의 문자) 및? 를 사용할 수도 있습니다. (임의의 단일 문자) 를 사용하여 패턴별로 스택을 식별할 수 있습니다. 와일드카드를 사용할 때는 패턴을 따옴표로 묶으십시오. 그렇지 않으면 셸이 툴킷으로 전달되기 전에 현재 디렉토리의 파일 이름으로 확장하려고 할 수 있습니다. AWS CDK

```
cdk synth "Stack?"    # Stack1, StackA, etc.
cdk deploy "*Stack"   # PipeStack, LambdaStack, etc.
```

> **Tip**
>
> 스택을 배포하기 전에 명시적으로 합성할 필요는 없습니다. 이 단계를 `cdk deploy` 수행하면 최신 코드가 배포되도록 할 수 있습니다.

명령의 전체 설명서를 보려면 을 참조하십시오. `cdk` the section called “AWS CDK 툴킷”

AWS CDK 애플리케이션 개발

AWS Cloud Development Kit (AWS CDK) 애플리케이션 개발

주제

구성 라이브러리에서 구성 사용자 지정하기 AWS

이스케이프 해치, 원시 오버라이드, 사용자 지정 리소스를 통해 AWS 구성 라이브러리의 구성을 사용자 정의할 수 있습니다.

주제

이스케이프 해치 사용

AWS 구성 라이브러리는 다양한 추상화 수준의 구문을 제공합니다.

최상위 수준에서 보면 AWS CDK 애플리케이션과 그 안에 포함된 스택 자체가 전체 클라우드 인프라 또는 그 중 상당 부분을 추상화한 것입니다. 이를 파라미터화하여 다양한 환경이나 다양한 요구 사항에 맞게 배포할 수 있습니다.

추상화는 클라우드 애플리케이션을 설계하고 구현하기 위한 강력한 도구입니다. 를 AWS CDK 사용하면 추상화를 사용하여 구축할 수 있을 뿐만 아니라 새로운 추상화를 만들 수도 있습니다. 기존 오픈 소스 L2 및 L3 구문을 지침으로 사용하여 조직의 모범 사례와 의견을 반영하는 자체 L2 및 L3 구문을 구축할 수 있습니다.

완벽한 추상화는 없으며, 아무리 좋은 추상화라도 가능한 모든 사용 사례를 포괄할 수는 없습니다. 개발 중에 필요에 거의 맞는 구조를 찾을 수 있으며, 크든 작든 커스터마이징이 필요할 수도 있습니다.

이러한 이유로 에서는 구성 모델을 분리할 수 있는 방법을 AWS CDK 제공합니다. 여기에는 하위 수준의 추상화로 이동하거나 완전히 다른 모델로 이동하는 것도 포함됩니다. 이스케이프 해치를 사용하면 AWS CDK 패러다임에서 벗어나 필요에 맞게 사용자 정의할 수 있습니다. 그런 다음 변경 내용을 새 구조로 래핑하여 근본적인 복잡성을 추상화하고 다른 개발자에게 깔끔한 API를 제공할 수 있습니다.

다음은 이스케이프 해치를 사용할 수 있는 상황의 예시입니다.

- 를 통해 AWS 서비스 기능을 사용할 수 AWS CloudFormation있지만 이에 대한 L2 구조는 없습니다.
- 를 통해 AWS AWS CloudFormation서비스 기능을 사용할 수 있으며 서비스에 대한 L2 구문이 있지만 이러한 구조에는 아직 기능이 표시되지 않습니다. L2 구조는 CDK 팀에서 큐레이션했기 때문에 새 기능에 바로 사용할 수 없을 수도 있습니다.
- 이 기능은 아직 전혀 사용할 수 없습니다. AWS CloudFormation

  를 통해 AWS CloudFormation기능을 사용할 수 있는지 확인하려면 AWS 리소스 및 속성 유형 참조를 참조하십시오.

L1 구문을 위한 이스케이프 해치 개발

서비스에 L2 구문을 사용할 수 없는 경우 자동으로 생성된 L1 구문을 사용할 수 있습니다. 이러한 리소스는 로 `Cfn` 시작하는 이름으로 식별할 수 있습니다 (예: 또는). `CfnBucket` `CfnRole` AWS CloudFormation 동등한 리소스를 사용할 때와 똑같이 인스턴스화합니다.

예를 들어, 분석이 활성화된 상태에서 하위 수준 Amazon S3 버킷 L1을 인스턴스화하려면 다음과 같이 작성합니다.

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",
        // ...
    }
```

```
});
```

드물게 해당 클래스가 없는 리소스를 정의하려는 경우가 있을 수 있습니다. `CfnXxx` 리소스 사양에 아직 게시되지 않은 새 AWS CloudFormation 리소스 유형일 수 있습니다. 이런 경우에는 를 `cdk.CfnResource` 직접 인스턴스화하고 리소스 유형과 속성을 지정할 수 있습니다. 방법은 다음 예제와 같습니다.

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"
            }
        }
    }
});
```

L2 구문을 위한 이스케이프 해치를 개발하세요.

L2 구문에 기능이 없거나 문제를 해결하려는 경우 L2 구문으로 캡슐화된 L1 구문을 수정할 수 있습니다.

모든 L2 구문은 그 안에 해당하는 L1 구문을 포함합니다. 예를 들어, 상위 수준 `Bucket` 구문은 하위 수준 구문을 래핑합니다. `CfnBucket` 는 AWS CloudFormation 리소스에 직접 `CfnBucket` 대응되므로 이를 통해 사용할 수 있는 모든 기능이 노출됩니다. AWS CloudFormation

L1 구문에 액세스하는 기본 방법은 `construct.node.defaultChild` (Python:`default_child`)를 사용하고 필요한 경우 올바른 유형으로 캐스팅한 다음 속성을 수정하는 것입니다. 다시 a의 `Bucket` 예를 들어 보겠습니다.

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",
        // ...
    }
};
```

이 객체를 사용하여 `Metadata` 및 와 같은 AWS CloudFormation 옵션을 변경할 수도 `UpdatePolicy` 있습니다.

TypeScript

```
cfnBucket.cfnOptions.metadata = {
  MetadataKey: 'MetadataValue'
};
```

JavaScript

```
cfnBucket.cfnOptions.metadata = {
  MetadataKey: 'MetadataValue'
```

```
};
```

Python

```
cfn_bucket.cfn_options.metadata = {
    "MetadataKey": "MetadataValue"
}
```

Java

```
cfnBucket.getCfnOptions().setMetadata(java.util.Map.of(    // Java 9+
    "MetadataKey", "Metadatavalue"));
```

C#

```
cfnBucket.CfnOptions.Metadata = new Dictionary<string, object>
{
    ["MetadataKey"] = "Metadatavalue"
};
```

언이스케이프 해치

AWS CDK 또한 추상화 수준을 높일 수 있는 기능을 제공하는데, 이를 "이스케이프 해제" 해치라고 할 수 있습니다. 예를 들어 L1 구문이 있는 경우 새 L2 구문 (Bucket이 경우) 을 만들어 L1 구문을 래핑할 수 있습니다. CfnBucket

이 방법은 L1 리소스를 만들지만 L2 리소스가 필요한 구문과 함께 사용하려는 경우에 편리합니다. L1 구조에서는 사용할 수 .grantXxxxx() 없는 편리한 메서드를 사용하려는 경우에도 유용합니다.

L2 클래스의 정적 메서드 .fromCfnXxxxx() (예: Bucket.fromCfnBucket() Amazon S3 버킷) 를 사용하여 더 높은 추상화 수준으로 이동합니다. L1 리소스가 유일한 파라미터입니다.

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);
```

L1 구조에서 생성된 L2 구문은 L1 리소스를 참조하는 프록시 객체로, 리소스 이름, ARN 또는 조회에서 생성되는 것과 유사합니다. 이러한 구문을 수정해도 최종 합성 AWS CloudFormation 템플릿에는 영향을 주지 않습니다. 하지만 L1 리소스가 있으므로 대신 수정할 수 있습니다. 프록시 오브젝트에 대한 자세한 내용은 을 참조하십시오. the section called “계정의 리소스 참조 AWS”

혼동을 피하려면 동일한 L1 구문을 참조하는 L2 구문을 여러 개 만들지 마십시오. 예를 들어 이전 섹션의 `Bucket` 방법을 사용하여 `CfnBucket` a에서 를 추출하는 경우 이를 사용하여 a를 `Bucket.fromCfnBucket()` 호출하여 두 번째 `Bucket` 인스턴스를 만들면 안 됩니다. `CfnBucket` 실제로는 예상대로 작동하지만 (하나만 `AWS::S3::Bucket` 합성됨) 코드를 유지 관리하기가 더 어려워집니다.

Raw 오버라이드

L1 구문에 누락된 속성이 있는 경우 원시 재정의를 사용하여 모든 입력을 우회할 수 있습니다. 이렇게 하면 합성된 속성을 삭제할 수도 있습니다.

다음 예제와 같이 `addOverride` 메서드 (Python:`add_override`) 메서드 중 하나를 사용합니다.

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");
```

사용자 정의 리소스

API 직접 호출을 AWS CloudFormation통해서만 기능을 사용할 수 없는 경우에는 AWS CloudFormation 사용자 지정 리소스를 작성하여 필요한 API 호출을 수행해야 합니다. 를 사용하여 사용자 지정 리소스를 작성하고 이를 일반 구성 인터페이스로 래핑할 수 있습니다. AWS CDK 구문을 사용하는 소비자의 관점에서 보면 그 경험이 자연스럽게 느껴질 것입니다.

사용자 지정 리소스를 구축하려면 CREATE 리소스의UPDATE,, 수명 주기 이벤트에 응답하는 Lambda 함수를 작성해야 합니다. DELETE 사용자 지정 리소스에서 API 호출을 한 번만 수행해야 하는 경우 사용을 고려해 보십시오. AwsCustomResource 이렇게 하면 배포 중에 임의의 AWS CloudFormation SDK 호출을 수행할 수 있습니다. 그렇지 않으면 필요한 작업을 수행하도록 Lambda 함수를 직접 작성해야 합니다.

주제가 너무 광범위해서 여기서 완전히 다룰 수 없지만 다음 링크를 통해 시작할 수 있습니다.

- 사용자 지정 리소스
- 사용자 지정 리소스 예제
- 좀 더 완전한 예제를 보려면 CDK 표준 라이브러리의 DnsValidatedCertificate클래스를 참조하십시오. 이는 사용자 지정 리소스로 구현됩니다.

환경 변수에서 값 가져오기

환경 변수 값을 가져오려면 다음과 같은 코드를 사용합니다. 이 코드는 환경 변수의 값을 가져옵니다MYBUCKET.

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";
```

AWS CloudFormation 값 사용

에서 AWS CloudFormation 매개변수를 사용하는 방법에 the section called “파라미터” 대한 자세한 내용은 을 참조하십시오 AWS CDK.

기존 AWS CloudFormation 템플릿의 리소스에 대한 참조를 가져오려면 을 참조하십시오the section called “ AWS CloudFormation 템플릿 가져오기”.

기존 AWS CloudFormation 템플릿 가져오기

구조를 사용하여 리소스를 L1 `cloudformation-include.CfnInclude`구조로 변환하여 AWS CloudFormation 템플릿의 리소스를 AWS Cloud Development Kit (AWS CDK) 애플리케이션으로 가져옵니다.

가져온 후에는 원래 코드에 정의된 리소스와 동일한 방식으로 앱에서 이러한 리소스를 사용할 수 있습니다. AWS CDK 상위 AWS CDK 수준 구문 내에서 이러한 L1 구문을 사용할 수도 있습니다. 예를 들어 이렇게 하면 L2 권한 부여 메서드를 정의된 리소스와 함께 사용할 수 있습니다.

이 `cloudformation-include.CfnInclude` 구조는 기본적으로 템플릿의 모든 리소스에 AWS CDK API 래퍼를 추가합니다. AWS CloudFormation 이 기능을 사용하면 기존 AWS CloudFormation 템플릿을 AWS CDK 한 번에 하나씩 가져올 수 있습니다. 이렇게 하면 AWS CDK 구문을 사용하여 기존 리소스를 관리하여 상위 수준 추상화의 이점을 활용할 수 있습니다. 또한 이 기능을 사용하여 구성 API를 제공하여 AWS CDK 개발자에게 AWS CloudFormation 템플릿을 제공할 수 있습니다. AWS CDK

> **Note**
>
> AWS CDK 이전에 동일한 범용으로 사용되었던 v1도 포함되어 `aws-cdk-lib.CfnInclude`있습니다. 하지만 의 기능은 많이 부족합니다. `cloudformation-include.CfnInclude`

주제

템플릿 가져오기 AWS CloudFormation

다음은 이 항목에서 예제를 제공하는 데 사용할 샘플 AWS CloudFormation 템플릿입니다. 다음과 같이 템플릿을 `my-template.json` 복사하고 저장합니다. 이 예제를 모두 살펴본 후 기존에 배포한 AWS CloudFormation 템플릿을 사용하여 더 자세히 살펴볼 수 있습니다. AWS CloudFormation 콘솔에서 다운로드할 수 있습니다.

```
{
  "Resources": {
    "MyBucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {
        "BucketName": "MyBucket",
      }
    }
  }
}
```

JSON 또는 YAML 템플릿 중 하나로 작업할 수 있습니다. YAML 파서가 허용하는 범위가 약간 다를 수 있으므로 가능한 경우 JSON을 사용하는 것이 좋습니다.

다음은 를 사용하여 샘플 템플릿을 AWS CDK 앱으로 가져오는 방법의 예입니다. `cloudformation-include` 템플릿은 CDK 스택의 컨텍스트 내에서 가져옵니다.

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"
            });
        }
    }
}
```

기본적으로 리소스를 가져오면 템플릿에 있는 리소스의 원래 논리 ID가 보존됩니다. 이 동작은 논리적 ID를 유지해야 하는 로 AWS CloudFormation 템플릿을 가져오는 데 적합합니다. AWS CDK AWS

CloudFormation 가져온 리소스를 템플릿의 동일한 리소스로 인식하려면 이 정보가 필요합니다. AWS CloudFormation

다른 AWS CDK 개발자가 사용할 수 있도록 템플릿의 AWS CDK 구성 래퍼를 개발하는 경우 대신 새 리소스 ID를 AWS CDK 생성하도록 하세요. 이렇게 하면 이름 충돌 없이 스택에서 구문을 여러 번 사용할 수 있습니다. 이렇게 하려면 템플릿을 가져올 `false` 때 `preserveLogicalIds` 속성을 로 설정하십시오. 다음은 그 예제입니다.

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
});
```

가져온 리소스를 AWS CDK 앱 제어 하에 두려면 스택을 다음에 추가하세요. App

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import { MyStack } from '../lib/my-stack';

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

JavaScript

```
const cdk = require('aws-cdk-lib');
const { MyStack } = require('../lib/my-stack');

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

Python

```
import aws_cdk as cdk
from mystack.my_stack import MyStack

app = cdk.App()
MyStack(app, "MyStack")
```

Java

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

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

        new MyStack(app, "MyStack");
    }
}
```

C#

```
using Amazon.CDK;

namespace CdkApp
{
    sealed class Program
    {
        public static void Main(string[] args)
        {
            var app = new App();
            new MyStack(app, "MyStack");
        }
    }
}
```

스택의 AWS 리소스에 의도하지 않은 변경이 없는지 확인하기 위해 diff를 수행할 수 있습니다. AWS CDK CLIcdk diff명령을 사용하고 특정 메타데이터는 생략하세요 AWS CDK. 다음은 그 예제입니다.

```
cdk diff --no-version-reporting --no-path-metadata --no-asset-metadata
```

AWS CloudFormation 템플릿을 가져온 후에는 AWS CDK 앱이 가져온 리소스의 신뢰할 수 있는 소스가 되어야 합니다. 리소스를 변경하려면 AWS CDK 앱에서 리소스를 수정하고 AWS CDK CLI cdk deploy 명령을 사용하여 배포하세요.

가져온 리소스에 액세스

예제 `template` 코드의 이름은 가져온 AWS CloudFormation 템플릿을 나타냅니다. 리소스에서 리소스에 액세스하려면 객체의 `getResource()`메서드를 사용하십시오. 반환된 리소스에 특정 종류의 리소스로 액세스하려면 결과를 원하는 유형으로 캐스팅하십시오. Python 또는 에서는 이것이 필요하지 않습니다 JavaScript. 다음은 그 예제입니다.

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");
```

이 예제에서는 cfnBucket 이 aws-s3.CfnBucket클래스의 인스턴스가 되었습니다. 이는 해당 AWS CloudFormation 리소스를 나타내는 L1 구조입니다. 해당 유형의 다른 리소스처럼 취급할 수 있습니다. 예를 들어 속성을 사용하여 ARN 값을 가져올 수 있습니다. bucket.attrArn

L1 CfnBucket 리소스를 L2 aws-s3.Bucket인스턴스로 대신 래핑하려면 정적 메서드 fromBucketArn(), fromBucketAttributes()또는 를 사용하십시오. fromBucketName() 일반적으로 이 fromBucketName() 방법이 가장 편리합니다. 다음은 그 예제입니다.

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);
```

다른 L2 구문에도 기존 리소스에서 구문을 만드는 비슷한 방법이 있습니다.

L1 구문을 L2 구문으로 래핑하면 새 리소스가 생성되지 않습니다. 이 예시에서는 두 번째 S3; 버킷을 만들지 않습니다. 대신 새 `Bucket` 인스턴스는 기존 인스턴스를 캡슐화합니다. `CfnBucket`

예제에서 보면 이제 `bucket` 는 다른 L2 구문처럼 동작하는 L2 `Bucket` 구문입니다. 예를 들어, 버킷의 편리한 메서드를 사용하여 AWS Lambda 함수에 버킷에 대한 쓰기 액세스 권한을 부여할 수 있습니다. `grantWrite()` 필요한 AWS Identity and Access Management (IAM) 정책을 수동으로 정의할 필요는 없습니다. 다음은 그 예제입니다.

TypeScript

```
bucket.grantWrite(lambdaFunc);
```

JavaScript

```
bucket.grantWrite(lambdaFunc);
```

Python

```
bucket.grant_write(lambda_func)
```

Java

```
bucket.grantWrite(lambdaFunc);
```

C#

```
bucket.GrantWrite(lambdaFunc);
```

파라미터 교체

AWS CloudFormation 템플릿에 파라미터가 포함된 경우, 임포트 시 `parameters` 프로퍼티를 사용하여 빌드 시간 값으로 대체할 수 있습니다. 다음 예시에서는 `UploadBucket` 파라미터를 코드의 다른 곳에 정의된 버킷의 ARN으로 대체합니다. AWS CDK

TypeScript

```
const template = new cfninc.CfnInclude(this, 'Template', {
  templateFile: 'my-template.json',
  parameters: {
    'UploadBucket': bucket.bucketArn,
  },
});
```

JavaScript

```
const template = new cfninc.CfnInclude(this, 'Template', {
  templateFile: 'my-template.json',
  parameters: {
    'UploadBucket': bucket.bucketArn,
  },
});
```

Python

```
template = cfn_inc.CfnInclude(self, "Template",
    template_file="my-template.json",
    parameters=dict(UploadBucket=bucket.bucket_arn)
)
```

Java

```
CfnInclude template = CfnInclude.Builder.create(this, "Template")
 .templateFile("my-template.json")
 .parameters(java.util.Map.of(    // Map.of requires Java 9+
   "UploadBucket", bucket.getBucketArn()))
 .build();
```

C#

```
var template = new cfnInc.CfnInclude(this, "Template", new cfnInc.CfnIncludeProps
{
    TemplateFile = "my-template.json",
    Parameters = new Dictionary<string, string>
    {
        { "UploadBucket", bucket.BucketArn }
    }
});
```

기타 템플릿 요소

리소스뿐만 아니라 모든 AWS CloudFormation 템플릿 요소를 가져올 수 있습니다. 가져온 요소는 AWS CDK 스택의 일부가 됩니다. 이러한 요소를 가져오려면 `CfnInclude` 객체의 다음 메서드를 사용하십시오.

- `getCondition()`— AWS CloudFormation 조건.
- `getHook()`— 블루/그린 배포용 AWS CloudFormation 후크.
- `getMapping()`AWS CloudFormation— 매핑.
- `getOutput()`— 출력 AWS CloudFormation .
- `getParameter()`— AWS CloudFormation 파라미터.
- `getRule()`— AWS Service Catalog 템플릿 AWS CloudFormation 규칙.

각 메서드는 특정 유형의 AWS CloudFormation 요소를 나타내는 클래스의 인스턴스를 반환합니다. 이러한 객체는 변경 가능합니다. 변경 내용은 AWS CDK 스택에서 생성되는 템플릿에 표시됩니다. 다음은 템플릿에서 매개 변수를 가져와 기본값을 수정하는 예제입니다.

TypeScript

```
const param = template.getParameter('MyParameter');
param.default = "AWS CDK"
```

JavaScript

```
const param = template.getParameter('MyParameter');
param.default = "AWS CDK"
```

Python

```
param = template.get_parameter("MyParameter")
param.default = "AWS CDK"
```

Java

```
CfnParameter param = template.getParameter("MyParameter");
param.setDefaultValue("AWS CDK")
```

C#

```
var cfnBucket = (CfnBucket)template.GetResource("MyBucket");
var param = template.GetParameter("MyParameter");
param.Default = "AWS CDK";
```

중첩 스택

기본 템플릿을 가져올 때 또는 나중에 중첩 스택을 지정하여 중첩 스택을 가져올 수 있습니다. 중첩된 템플릿은 로컬 파일에 저장해야 하지만 기본 템플릿에서는 NestedStack 리소스로 참조해야 합니다. 또한 AWS CDK 코드에 사용된 리소스 이름은 기본 템플릿의 중첩 스택에 사용된 이름과 일치해야 합니다.

기본 템플릿에 이 리소스 정의가 있는 경우 다음 코드는 참조된 중첩 스택을 양방향으로 가져오는 방법을 보여줍니다.

```
"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'
});
```

두 방법 중 하나를 사용하여 여러 개의 중첩된 스택을 가져올 수 있습니다. 기본 템플릿을 가져올 때 각 중첩된 스택의 리소스 이름과 해당 템플릿 파일 간의 매핑을 제공합니다. 이 매핑에는 여러 개의 항목이 포함될 수 있습니다. 초기 가져오기 이후에 이 작업을 수행하려면 각 중첩된 스택에 대해 `loadNestedStack()` 한 번씩 호출하십시오.

중첩된 스택을 가져온 후에는 기본 템플릿의 메서드를 사용하여 액세스할 수 있습니다. `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;
```

`getNestedStack()`메서드는 인스턴스를 반환합니다. `IncludedNestedStack` 이 인스턴스에서는 예제와 같이 `stack` 속성을 통해 AWS CDK `NestedStack`인스턴스에 액세스할 수 있습니다. 또한 리소스 및 기타 AWS CloudFormation 요소를 로드할 수 있는 방법을 통해 `includedTemplate` 원본 AWS CloudFormation 템플릿 객체에 액세스할 수 있습니다.

Systems Manager 파라미터 저장소에서 값 가져오기

AWS Systems Manager 파라미터 스토어 속성 값을 검색할 AWS Cloud Development Kit (AWS CDK) 수 있습니다. 합성 중에 는 토큰을 AWS CDK 생성하며, 토큰은 배포 AWS CloudFormation 중에 확인됩니다.

는 일반 값과 보안 값 검색을 모두 AWS CDK 지원합니다. 어떤 종류의 값이든 특정 버전을 요청할 수 있습니다. 일반 값의 경우 요청에서 버전을 생략하여 최신 버전을 가져올 수 있습니다. 보안 값의 경우 보안 속성 값을 요청할 때 버전을 지정해야 합니다.

> **Note**
>
> 이 항목에서는 AWS Systems Manager 매개변수 저장소에서 속성을 읽는 방법을 보여줍니다. 에서 시크릿을 읽을 수도 있습니다 AWS Secrets Manager (참조에서 가치 가져오기 AWS Secrets Manager).

주제

배포 시 Systems Manager 값 읽기

Systems Manager 매개변수 저장소에서 값을 읽으려면 valueForString매개변수와 valueForSecureStringParameter메서드를 사용하십시오. 원하는 속성이 일반 문자열인지 보안 문자열 값인지에 따라 메서드를 선택합니다. 이러한 메서드는 실제 값이 아닌 토큰을 반환합니다. 값은 배포 AWS CloudFormation 중에 확인됩니다. 다음은 그 예제입니다.

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
```

현재 이 기능을 지원하는 AWS 서비스 수는 제한되어 있습니다.

합성 시 Systems Manager 값 읽기

때로는 합성 시 파라미터를 제공하는 것이 유용할 때가 있습니다. 이렇게 하면 AWS CloudFormation 템플릿이 배포 중에 값을 확인하는 대신 항상 같은 값을 사용하게 됩니다.

합성 시 Systems Manager 파라미터 저장소에서 값을 읽으려면 (Python:`value_from_lookup`) valueFromLookup메서드를 사용하십시오. 이 메서드는 매개변수의 실제 the section called “컨텍스트” 값을 값으로 반환합니다. 값이 아직 캐시되지 `cdk.json` 않았거나 명령줄에 전달되지 않은 경우 현재 AWS 계정에서 검색됩니다. 따라서 스택은 명시적인 AWS 환경 정보와 함께 합성되어야 합니다.

다음은 그 예제입니다.

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");
```

일반 Systems Manager 문자열만 검색할 수 있습니다. 보안 문자열은 검색할 수 없습니다. 항상 최신 버전이 반환됩니다. 특정 버전은 요청할 수 없습니다.

> **Important**
>
> 검색된 값은 합성된 AWS CloudFormation 템플릿에 저장됩니다. 템플릿에 액세스할 수 있는 사용자와 AWS CloudFormation 템플릿의 값의 종류에 따라 보안 위험이 발생할 수 있습니다. 일반적으로 비공개로 유지하려는 암호, 키 또는 기타 값에는 이 기능을 사용하지 마십시오.

Systems Manager에 값 쓰기

AWS CLI AWS Management Console, 또는 AWS SDK를 사용하여 Systems Manager 파라미터 값을 설정할 수 있습니다. 다음 예에서는 ssm put-parameter CLI 명령을 사용합니다.

```
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"
```

이미 있는 SSM 값을 업데이트할 때는 옵션도 포함하십시오. `--overwrite`

```
aws ssm put-parameter --overwrite --name "parameter-name" --type "String" --value
 "parameter-value"
aws ssm put-parameter --overwrite --name "secure-parameter-name" --type "SecureString"
 --value "secure-parameter-value"
```

에서 가치 가져오기 AWS Secrets Manager

AWS CDK 앱의 AWS Secrets Manager 값을 사용하려면 fromSecretAttributes() 메서드를 사용하세요. 이는 Secrets Manager에서 검색되어 AWS CloudFormation 배포 시 사용되는 값을 나타냅니다. 다음은 그 예제입니다.

TypeScript

```
import * as sm from "aws-cdk-lib/aws-secretsmanager";

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

    const secret = sm.Secret.fromSecretAttributes(this, "ImportedSecret", {
      secretCompleteArn:
        "arn:aws:secretsmanager:<region>:<account-id-number>:secret:<secret-name>-
<random-6-characters>"
      // If the secret is encrypted using a KMS-hosted CMK, either import or
 reference that key:
      // encryptionKey: ...
    });
```

JavaScript

```
const sm = require("aws-cdk-lib/aws-secretsmanager");

class SecretsManagerStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    const secret = sm.Secret.fromSecretAttributes(this, "ImportedSecret", {
      secretCompleteArn:
        "arn:aws:secretsmanager:<region>:<account-id-number>:secret:<secret-name>-
<random-6-characters>"
      // If the secret is encrypted using a KMS-hosted CMK, either import or
 reference that key:
      // encryptionKey: ...
    });
  }
}

module.exports = { SecretsManagerStack }
```

Python

```
import aws_cdk.aws_secretsmanager as sm

class SecretsManagerStack(cdk.Stack):
```

```
    def __init__(self, scope: cdk.App, id: str, **kwargs):
      super().__init__(scope, name, **kwargs)

      secret = sm.Secret.from_secret_attributes(self, "ImportedSecret",
          secret_complete_arn="arn:aws:secretsmanager:<region>:<account-id-
number>:secret:<secret-name>-<random-6-characters>",
          # If the secret is encrypted using a KMS-hosted CMK, either import or
 reference that key:
          # encryption_key=....
      )
```

Java

```
import software.amazon.awscdk.services.secretsmanager.Secret;
import software.amazon.awscdk.services.secretsmanager.SecretAttributes;

public class SecretsManagerStack extends Stack {
    public SecretsManagerStack(App scope, String id) {
        this(scope, id, null);
    }

    public SecretsManagerStack(App scope, String id, StackProps props) {
        super(scope, id, props);

        Secret secret = (Secret)Secret.fromSecretAttributes(this, "ImportedSecret",
 SecretAttributes.builder()
            .secretCompleteArn("arn:aws:secretsmanager:<region>:<account-id-
number>:secret:<secret-name>-<random-6-characters>")
             // If the secret is encrypted using a KMS-hosted CMK, either import or
 reference that key:
             // .encryptionKey(...)
             .build());
    }
}
```

C#

```
using Amazon.CDK.AWS.SecretsManager;

public class SecretsManagerStack : Stack
{
    public SecretsManagerStack(App scope, string id, StackProps props) : base(scope,
 id, props) {
```

```
        var secret = Secret.FromSecretAttributes(this, "ImportedSecret", new
 SecretAttributes {
            SecretCompleteArn = "arn:aws:secretsmanager:<region>:<account-id-
number>:secret:<secret-name>-<random-6-characters>"
            // If the secret is encrypted using a KMS-hosted CMK, either import or
 reference that key:
            // encryptionKey = ...,
        });
    }
```

Tip

AWS CLI create-secret CLI 명령을 사용하면 다음과 같이 명령줄에서 시크릿을 생성할 수 있습니다 (예: 테스트 시).

```
aws secretsmanager create-secret --name ImportedSecret --secret-string
 mygroovybucket
```

이 명령은 이전 예제와 함께 사용할 수 있는 ARN을 반환합니다.

인스턴스를 생성한 후에는 `Secret` 인스턴스의 속성에서 시크릿 값을 가져올 수 있습니다. `secretValue` 값은 특수 유형인 `SecretValue` 인스턴스로 표시됩니다the section called “토큰”. 토큰이기 때문에 확인을 거쳐야 의미가 있습니다. CDK 앱은 실제 가치에 접근할 필요가 없습니다. 대신 앱은 `SecretValue` 인스턴스 (또는 문자열 또는 숫자 표현) 를 값이 필요한 CDK 메서드에 전달할 수 있습니다.

CloudWatch 알람 설정

AWS-cloudwatch 패키지를 사용하여 지표에 대한 아마존 CloudWatch 경보를 설정할 수 있습니다. CloudWatch 사전 정의된 지표를 사용하거나 직접 지표를 생성할 수 있습니다.

주제

- 기존 지표 사용
- 자체 지표 생성
- 알람 생성

기존 지표 사용

많은 AWS Construct Library 모듈을 사용하면 메트릭이 있는 객체 인스턴스의 편의 메서드에 지표 이름을 전달하여 기존 지표에 경보를 설정할 수 있습니다. 예를 들어 Amazon SQS 대기열이 있는 경우 대기열의 metric () ApproximateNumberOfMessagesVisible메서드에서 지표를 가져올 수 있습니다.

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");
```

자체 지표 생성

다음과 같이 자체 지표를 생성합니다. 여기서 네임스페이스 값은 Amazon SQS 대기열의 AWS/SQS와 비슷해야 합니다. 또한 지표의 이름과 차원을 지정해야 합니다.

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" }
    }
});
```

알람 생성

기존 지표 또는 사용자가 정의한 지표가 있으면 경보를 생성할 수 있습니다. 이 예시에서는 최근 세 번의 평가 기간 중 두 기간에 지표가 100개를 넘으면 경보가 발생합니다. 속성을 통해 경보에서 작은

과 같은 비교 결과를 사용할 수 있습니다. `comparisonOperator` G reater-than-or-equal -to가 AWS CDK 기본값이므로 지정할 필요가 없습니다.

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
});
```

경보를 생성하는 또 다른 방법은 지표의 createAlarm () 메서드를 사용하는 것입니다. 이 메서드는 기본적으로 생성자와 동일한 속성을 사용합니다. `Alarm` 메트릭은 이미 알려져 있기 때문에 전달하지 않아도 됩니다.

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
});
```

컨텍스트 변수 값 저장 및 검색

cdk.json파일에서 AWS Cloud Development Kit (AWS CDK) CLI 또는 를 사용하여 컨텍스트 변수를 지정할 수 있습니다. 그런 다음 TryGetContext 메서드를 사용하여 값을 검색합니다.

주제

- 컨텍스트 변수를 지정하십시오.
- 컨텍스트 변수 값 검색

컨텍스트 변수를 지정하십시오.

컨텍스트 변수는 AWS CDK CLI 명령의 일부로 또는 에서 지정할 수 cdk.json 있습니다.

명령줄 컨텍스트 변수를 만들려면 다음 예제와 같이 --context (-c) 옵션을 사용합니다.

```
cdk synth -c bucket_name=mygroovybucket
```

cdk.json파일에서 동일한 컨텍스트 변수와 값을 지정하려면 다음 코드를 사용합니다.

```
{
  "context": {
    "bucket_name": "myotherbucket"
  }
}
```

AWS CDK CLI및 cdk.json 파일을 모두 사용하여 컨텍스트 변수를 지정하는 경우 AWS CDK CLI 값이 우선합니다.

컨텍스트 변수 값 검색

앱의 컨텍스트 변수 값을 가져오려면 구문의 컨텍스트에서 TryGetContext 메서드를 사용하십시오. (즉this, 또는 self Python에서 어떤 구문의 인스턴스가 되는 경우입니다.)

이 예제에서는 bucket_name 컨텍스트 변수의 값을 검색합니다. 요청된 값이 정의되지 않은 경우 예외를 발생시키지 않고 undefined (NonePython, Java 및 C #, nil Go에서) TryGetContext 반환합니다. null

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");
```

구문의 컨텍스트 외부에서는 다음과 같이 앱 객체에서 컨텍스트 변수에 액세스할 수 있습니다.

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");
```

컨텍스트 변수 작업에 대한 자세한 내용은 을 참조하십시오the section called “컨텍스트”.

AWS CloudFormation 퍼블릭 레지스트리의 리소스 사용

AWS CloudFormation 퍼블릭 레지스트리를 사용하면 에서 사용할 수 있는 리소스, 모듈, 후크 등 퍼블릭 및 프라이빗 확장을 모두 관리할 수 있습니다 AWS 계정. 구문과 함께 AWS Cloud Development Kit (AWS CDK) 애플리케이션에서 퍼블릭 리소스 확장을 사용할 수 `CfnResource`있습니다.

AWS CloudFormation 퍼블릭 레지스트리에 대한 자세한 내용은 사용 AWS CloudFormation 설명서의 AWS CloudFormation 레지스트리 사용을 참조하십시오.

에서 게시한 AWS 모든 공개 확장 프로그램은 사용자가 별도의 조치를 취하지 않아도 모든 지역의 모든 계정에서 사용할 수 있습니다. 하지만 사용하려는 각 계정 및 지역에서 사용하려는 타사 확장 프로그램을 각각 활성화해야 합니다.

> **Note**
>
> 타사 리소스 유형과 AWS CloudFormation 함께 사용하는 경우 요금이 부과됩니다. 요금은 매월 실행하는 핸들러 작업 수와 핸들러 작업 기간을 기준으로 합니다. 자세한 내용은 CloudFormation 요금을 참조하십시오.

공개 확장에 대해 자세히 알아보려면 사용 설명서의AWS CloudFormation 공개 확장 프로그램 사용을 참조하십시오. CloudFormation

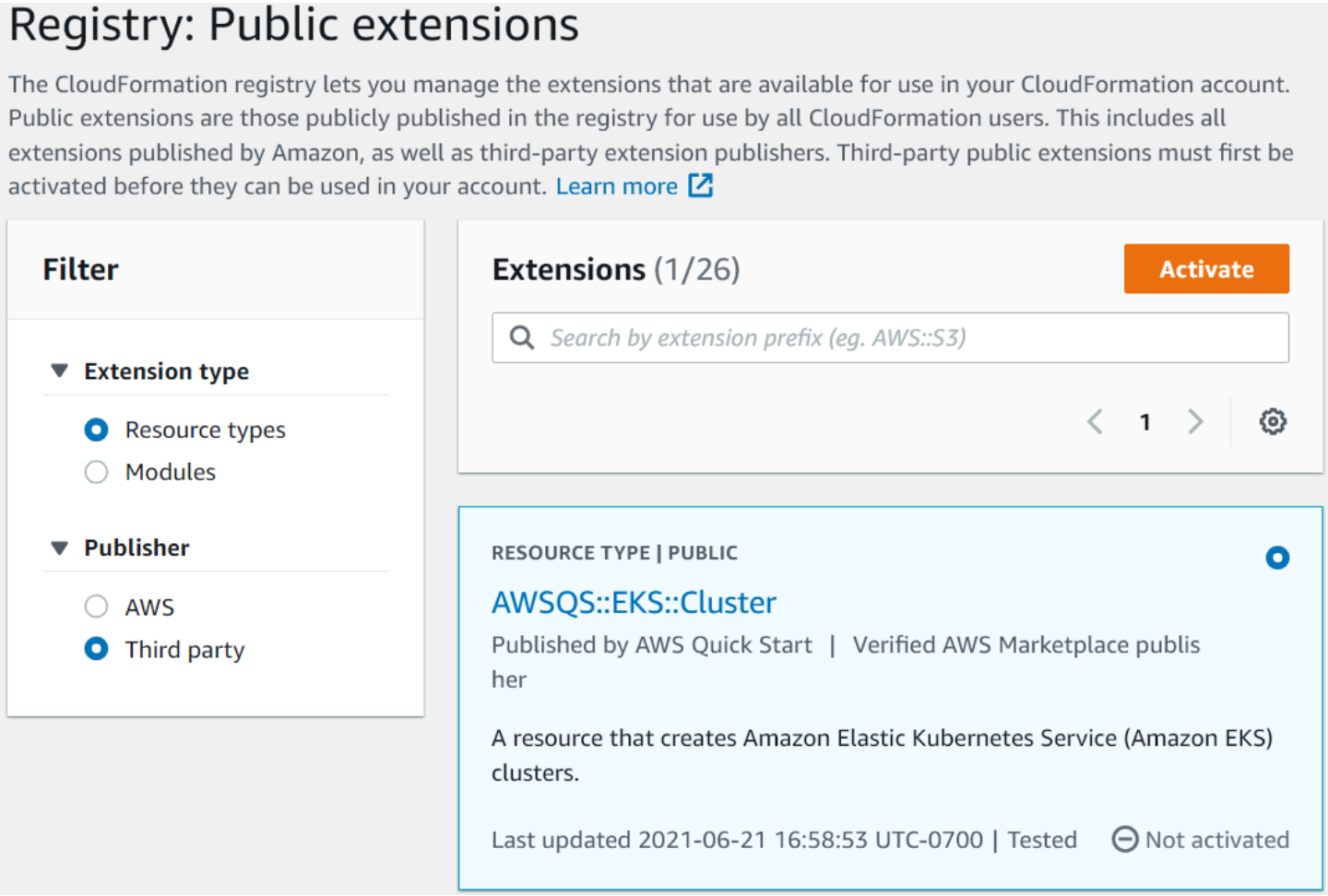
주제

- 계정 및 지역의 타사 리소스 활성화하기
- AWS CloudFormation 퍼블릭 레지스트리의 리소스를 CDK 앱에 추가

계정 및 지역의 타사 리소스 활성화하기

에서 게시한 확장 프로그램에는 활성화가 AWS 필요하지 않습니다. 모든 계정과 지역에서 항상 사용할 수 있습니다. 를 통해 AWS Management Console, 를 통해 또는 특수 AWS CloudFormation 리소스를 배포하여 타사 확장 프로그램을 활성화할 수 있습니다. AWS Command Line Interface

를 통해 타사 확장 프로그램을 활성화하려면 AWS Management Console 또는 사용 가능한 리소스를 확인하세요.



1. 확장 프로그램을 사용하려는 AWS 계정에 로그인한 다음 확장 프로그램을 사용하려는 지역으로 전환하십시오.
2. 서비스 메뉴를 통해 CloudFormation 콘솔로 이동합니다.
3. 탐색 표시줄에서 공개 확장을 선택한 다음 Publisher 아래의 타사 라디오 버튼을 활성화합니다. 사용 가능한 타사 공개 확장 목록이 표시됩니다. (활성화할 필요는 없지만 에서 게시한 AWS공개 확장 목록을 AWS표시하도록 선택할 수도 있습니다.)
4. 목록을 탐색하여 활성화하려는 확장 프로그램을 찾으세요. 또는 확장 프로그램을 검색한 다음 확장 프로그램 카드의 오른쪽 상단에 있는 라디오 버튼을 활성화하십시오.
5. 목록 상단의 활성화 버튼을 선택하여 선택한 확장 프로그램을 활성화합니다. 확장 프로그램의 활성화 페이지가 나타납니다.

6. 활성화 페이지에서 확장의 기본 이름을 재정의하고 실행 역할 및 로깅 구성을 지정할 수 있습니다. 새 버전이 출시될 때 확장 프로그램을 자동으로 업데이트할지 여부도 선택할 수 있습니다. 이러한 옵션을 원하는 대로 설정한 후 페이지 하단에서 확장 프로그램 활성화를 선택합니다.

를 사용하여 타사 확장 프로그램을 활성화하려면 AWS CLI

- `activate-type` 명령을 사용합니다. 표시된 곳에 사용하려는 사용자 지정 유형의 ARN을 대체하십시오.

  다음은 그 예제입니다.

  ```
  aws cloudformation activate-type --public-type-arn public_extension_ARN --auto-update-activated
  ```

CloudFormation 또는 CDK를 통해 타사 확장 프로그램을 활성화하려면

- 유형의 리소스를 `AWS::CloudFormation::TypeActivation` 배포하고 다음 속성을 지정하십시오.

  a. `TypeName`- 유형의 이름 (예:`AWSQS::EKS::Cluster`.

  b. `MajorVersion`- 원하는 확장의 메이저 버전 번호입니다. 최신 버전을 원하면 생략하세요.

  c. `AutoUpdate`- 게시자가 새 마이너 버전을 출시할 때 이 확장을 자동으로 업데이트할지 여부. (메이저 버전을 업데이트하려면 속성을 명시적으로 변경해야 합니다`MajorVersion`.)

  d. `ExecutionRoleArn`- 이 확장 프로그램이 실행될 IAM 역할의 ARN

  e. `LoggingConfig`- 확장 프로그램의 로깅 구성.

  CDK는 `CfnResource`구문을 사용하여 `TypeActivation` 리소스를 배포할 수 있습니다. 실제 확장에 대한 내용은 다음 섹션에 나와 있습니다.

AWS CloudFormation 퍼블릭 레지스트리의 리소스를 CDK 앱에 추가

`CfnResource`구문을 사용하여 AWS CloudFormation 퍼블릭 레지스트리의 리소스를 애플리케이션에 포함시키십시오. 이 구조는 CDK `aws-cdk-lib` 모듈에 있습니다.

예를 들어 애플리케이션에 사용하려는 퍼블릭 리소스가 `MY::S5::UltimateBucket` 있다고 가정해 보겠습니다. AWS CDK 이 리소스는 하나의 속성, 즉 버킷 이름을 사용합니다. 해당 `CfnResource` 인스턴스화는 다음과 같습니다.

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"
    }
});
```

애플리케이션 배포 AWS CDK

AWS Cloud Development Kit (AWS CDK) 애플리케이션 배포.

주제

AWS CDK 합성 시 정책 검증

주제

합성 시 정책 검증

사용자 또는 조직에서 OPA와 같은 AWS CloudFormation Guard정책 검증 도구를 사용하여 AWS CloudFormation 템플릿의 제약 조건을 정의하는 경우 통합 AWS CDK 시 에 통합할 수 있습니다. 적절한 정책 검증 플러그인을 사용하면 AWS CDK 애플리케이션이 합성 직후에 생성된 AWS CloudFormation 템플릿을 정책과 비교하여 확인하도록 할 수 있습니다. 위반 사항이 있는 경우 합성이 실패하고 보고서가 콘솔에 인쇄됩니다.

합성 AWS CDK 시 수행한 검증은 배포 수명 주기의 한 시점에서 제어를 검증하지만 합성 외부에서 발생하는 작업에는 영향을 미치지 않습니다. 콘솔에서 직접 또는 서비스 API를 통해 수행한 작업을 예로 들 수 있습니다. 합성 후 AWS CloudFormation 템플릿이 변경되는 것을 거부하지 않습니다. 동일한 규칙 세트를 더 신뢰할 수 있게 검증할 수 있는 다른 메커니즘 (예: 후크 또는) 은 독립적으로 설정해야 합니다.AWS CloudFormationAWS Config 하지만 개발 중에 규칙 세트를 평가하는 기능은 탐지 속도와 개발자 생산성을 향상시키므로 여전히 유용합니다. AWS CDK

AWS CDK 정책 검증의 목표는 개발 중에 필요한 설정의 양을 최소화하고 가능한 한 쉽게 만드는 것입니다.

> **Note**
>
> 이 기능은 실험용으로 간주되며 플러그인 API와 검증 보고서 형식은 향후 변경될 수 있습니다.

주제

애플리케이션 개발자용

애플리케이션에서 하나 이상의 유효성 검사 플러그인을 사용하려면 다음 `policyValidationBeta1` 속성을 사용하십시오`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: [...],
});
```

합성 직후 이 방법으로 등록된 모든 플러그인이 호출되어 정의한 범위에서 생성된 모든 템플릿의 유효성을 검사합니다. 특히, App 객체에 템플릿을 등록하면 모든 템플릿이 검증 대상이 됩니다.

> **Warning**
>
> 클라우드 어셈블리를 수정하는 것 외에 플러그인은 AWS CDK 애플리케이션이 할 수 있는 모든 작업을 수행할 수 있습니다. 파일 시스템에서 데이터를 읽고 네트워크에 액세스하는 등의 작업을 수행할 수 있습니다. 플러그인을 안전하게 사용할 수 있는지 확인하는 것은 플러그인 소비자인 귀하의 책임입니다.

AWS CloudFormation Guard 플러그인

CfnGuardValidator플러그인을 사용하면 정책 검증을 수행하는 AWS CloudFormation Guard데 사용할 수 있습니다. `CfnGuardValidator`플러그인에는 엄선된 AWS Control Tower 사전 예방 제어 세트가 내장되어 있습니다. 현재 규칙 세트는 프로젝트 문서에서 찾을 수 있습니다. 에서 언급한 것처럼 합성 시 정책 검증 조직에서는 후크를 사용하여AWS CloudFormation 보다 신뢰할 수 있는 검증 방법을 설정하는 것이 좋습니다.

AWS Control Tower고객의 경우 이와 동일한 사전 예방 제어 기능을 조직 전체에 배포할 수 있습니다. AWS Control Tower 환경에서 AWS Control Tower 사전 예방적 제어를 활성화하면 제어를 통해 배포된 규정을 준수하지 않는 리소스의 배포가 중단될 수 있습니다. AWS CloudFormation관리형 사전 제어 및 작동 방식에 대한 자세한 내용은 설명서를 참조하십시오.AWS Control Tower

이러한 AWS CDK 번들 제어와 관리형 AWS Control Tower 사전 예방적 제어는 함께 사용하는 것이 가장 좋습니다. 이 시나리오에서는 클라우드 환경에서 활성화된 것과 동일한 사전 예방적 제어를 사용하여 이 검증 플러그인을 구성할 수 있습니다 AWS Control Tower . 그러면 `cdk synth` 로컬에서 실행하여 AWS CDK 애플리케이션이 AWS Control Tower 제어를 통과할 것이라는 확신을 빠르게 확보할 수 있습니다.

검증 보고서

AWS CDK 앱을 합성하면 유효성 검사기 플러그인이 호출되고 결과가 인쇄됩니다. 예제 보고서는 다음과 같습니다.

```
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
```

기본적으로 보고서는 사람이 읽을 수 있는 형식으로 인쇄됩니다. JSON 형식의 보고서를 원하는 경우 CLI를 통해 활성화하거나 애플리케이션에 직접 전달하십시오. `@aws-cdk/core:validationReportJson`

```
const app = new App({
  context: { '@aws-cdk/core:validationReportJson': true },
});
```

또는 프로젝트 디렉터리의 `cdk.json` or `cdk.context.json` 파일을 사용하여 이 컨텍스트 키-값 쌍을 설정할 수 있습니다 (참조). 런타임 컨텍스트

JSON 형식을 선택하면 클라우드 어셈블리 AWS CDK `policy-validation-report.json` 디렉터리에 있는 파일에 정책 검증 보고서가 인쇄됩니다. 사람이 읽을 수 있는 기본 형식의 경우 보고서는 표준 출력으로 인쇄됩니다.

플러그인 작성자용

플러그인

AWS CDK 핵심 프레임워크는 플러그인을 등록하고 호출한 다음 형식이 지정된 검증 보고서를 표시하는 역할을 합니다. 플러그인의 책임은 AWS CDK 프레임워크와 정책 검증 도구 사이의 번역 계층 역할을 하는 것입니다. 에서 지원하는 모든 언어로 플러그인을 만들 수 AWS CDK있습니다. 여러 언어에서 사용할 수 있는 플러그인을 만드는 경우 JSII를 사용하여 각 AWS CDK 언어로 플러그인을 게시할 수 `TypeScript` 있도록 에서 플러그인을 만드는 것이 좋습니다.

플러그인 생성

AWS CDK 코어 모듈과 정책 도구 간의 통신 프로토콜은 `IPolicyValidationPluginBeta1` 인터페이스에 의해 정의됩니다. 새 플러그인을 만들려면 이 인터페이스를 구현하는 클래스를 작성해야 합니다. 두 가지를 구현해야 합니다. 바로 플러그인 이름 (`name`속성을 재정의하여) 과 메서드입니다. `validate()`

프레임워크는 `IValidationContextBeta1` 객체를 `validate()` 전달하여 호출합니다. 검증할 템플릿의 위치는 에 의해 지정됩니다. `templatePaths` 플러그인은 의 인스턴스를 반환해야 합니다. `ValidationPluginReportBeta1` 이 객체는 합성이 끝날 때 사용자가 받게 될 보고서를 나타냅니다.

```
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',
      }],
    }],
  };
}
```

참고로 플러그인은 클라우드 어셈블리의 어떤 내용도 수정할 수 없습니다. 그렇게 하려고 하면 합성에 실패하게 됩니다.

플러그인이 외부 도구에 의존하는 경우 일부 개발자의 워크스테이션에 해당 도구가 아직 설치되어 있지 않을 수 있다는 점을 염두에 두세요. 마찰을 최소화하려면 플러그인 패키지와 함께 설치 스크립트를 제공하여 전체 프로세스를 자동화하는 것이 좋습니다. 더 나은 방법은 해당 스크립트를 패키지 설치의 일부로 실행하는 것입니다. 예를 npm 들어 를 사용하여 `package.json` 파일의 `postinstall` 스크립트에 추가할 수 있습니다.

예외 처리

조직에 면제를 처리하는 메커니즘이 있는 경우 유효성 검사기 플러그인의 일부로 구현할 수 있습니다.

가능한 면제 메커니즘을 설명하는 예제 시나리오:

- 조직에는 특정 시나리오를 제외하고 퍼블릭 Amazon S3 버킷은 허용되지 않는다는 규칙이 있습니다.
- 개발자가 이러한 시나리오 중 하나에 해당하는 Amazon S3 버킷을 생성하고 면제를 요청합니다 (예: 티켓 생성).
- 보안 도구는 면제를 등록하는 내부 시스템에서 데이터를 읽는 방법을 알고 있습니다.

이 시나리오에서는 개발자가 내부 시스템에 예외를 요청한 다음 해당 예외를 “등록”할 방법이 필요합니다. 가드 플러그인 예제에 더해 내부 티켓팅 시스템에서 일치하는 면제가 있는 위반을 필터링하여 면제를 처리하는 플러그인을 만들 수 있습니다.

예제 구현은 기존 플러그인을 참조하십시오.

- @cdklabs/cdk-validator-cfnguard

CDK Pipeline을 사용한 지속적 통합 및 전달 (CI/CD)

구성 라이브러리의 CDK Pipelines 모듈을 사용하여 애플리케이션의 지속적 AWS 딜리버리를 구성할 수 있습니다. AWS CDK CDK 앱의 소스 코드를 AWS CodeCommitGitHub, 또는 AWS CodeStar에 커밋하면 CDK Pipelines에서 자동으로 새 버전을 빌드, 테스트, 배포할 수 있습니다.

CDK 파이프라인은 자동으로 업데이트됩니다. 애플리케이션 스테이지 또는 스택을 추가하면 파이프라인이 자동으로 재구성되어 새 스테이지 또는 스택을 배포합니다.

> **Note**
>
> CDK 파이프라인은 두 가지 API를 지원합니다. 하나는 CDK Pipelines 개발자 프리뷰에서 사용할 수 있는 오리지널 API입니다. 다른 하나는 프리뷰 단계에서 받은 CDK 고객의 피드백을 통합한 최신 API입니다. 이 항목의 예시에서는 최신 API를 사용합니다. 지원되는 두 API 간의 차이점에 대한 자세한 내용은 aws-cdk 리포지토리의 CDK Pipelines 원본 API를 참조하십시오. GitHub

주제

환경을 부트스트랩하세요. AWS

CDK Pipelines를 사용하려면 먼저 스택을 배포할 환경을 AWS 부트스트랩해야 합니다.

CDK 파이프라인에는 최소 두 개의 환경이 포함됩니다. 첫 번째 환경은 파이프라인이 프로비저닝되는 곳입니다. 두 번째 환경은 애플리케이션의 스택 또는 스테이지를 배포하려는 곳입니다 (스테이지는 관련 스택 그룹). 이러한 환경은 동일할 수 있지만 모범 사례 권장 사항은 서로 다른 환경에서 단계를 서로 격리하는 것입니다.

> **Note**
>
> 부트스트랩으로 생성되는 리소스의 종류 및 부트스트랩 스택을 사용자 지정하는 방법에 the section called “부트스트래핑” 대한 자세한 내용은 을 참조하십시오.

CDK Pipeline을 사용하여 지속적으로 배포하려면 CDK Toolkit 스택에 다음을 포함해야 합니다.

- Amazon Simple Storage Service(S3) 버킷

- 아마존 ECR 리포지토리.
- 파이프라인의 다양한 부분에 필요한 권한을 부여하는 IAM 역할.

CDK 툴킷은 기존 부트스트랩 스택을 업그레이드하거나 필요한 경우 새 부트스트랩 스택을 생성합니다.

AWS CDK 파이프라인을 프로비저닝할 수 있는 환경을 부트스트랩하려면 다음 `cdk bootstrap` 예와 같이 호출하십시오. 필요한 경우 `npx` 명령을 통해 AWS CDK 툴킷을 호출하면 툴킷이 임시로 설치됩니다. 또한 현재 프로젝트에 설치된 툴킷 버전 (있는 경우) 도 사용됩니다.

--cloudformation-execution-policies향후 CDK Pipelines 배포가 실행될 정책의 ARN을 지정합니다. 기본 `AdministratorAccess` 정책은 파이프라인이 모든 유형의 리소스를 배포할 수 있도록 합니다. AWS 이 정책을 사용하는 경우 AWS CDK 앱을 구성하는 모든 코드와 종속성을 신뢰해야 합니다.

대부분의 조직에서는 자동화를 통해 배포할 수 있는 리소스 종류에 대해 더 엄격한 제어를 요구합니다. 파이프라인에서 사용해야 하는 정책을 결정하려면 조직 내 해당 부서에 문의하세요.

기본 AWS 프로필에 필요한 인증 구성 및 AWS 리전이 포함되어 있는 경우 이 --profile 옵션을 생략할 수 있습니다.

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
```

파이프라인에서 AWS CDK 애플리케이션을 배포할 추가 환경을 부트스트랩하려면 다음 명령을 대신 사용하십시오. --trust옵션은 이 환경에 AWS CDK 애플리케이션을 배포할 수 있는 권한을 가져야 하는 다른 계정을 나타냅니다. 이 옵션의 경우 파이프라인의 AWS 계정 ID를 지정합니다.

다시 말하지만, 기본 AWS 프로필에 필요한 인증 구성 및 AWS 리전이 포함되어 있으면 이 --profile 옵션을 생략할 수 있습니다.

macOS/Linux

```
npx cdk bootstrap aws://ACCOUNT-NUMBER/REGION --profile ADMIN-PROFILE \
    --cloudformation-execution-policies arn:aws:iam::aws:policy/AdministratorAccess
 \
    --trust PIPELINE-ACCOUNT-NUMBER
```

Windows

```
npx cdk bootstrap aws://ACCOUNT-NUMBER/REGION --profile ADMIN-PROFILE ^
    --cloudformation-execution-policies arn:aws:iam::aws:policy/AdministratorAccess
 ^
    --trust PIPELINE-ACCOUNT-NUMBER
```

> **Tip**
>
> 관리 자격 증명은 초기 파이프라인을 부트스트랩하고 프로비저닝할 때만 사용하십시오. 이후에는 로컬 컴퓨터가 아닌 파이프라인 자체를 사용하여 변경 내용을 배포하세요.

기존 부트스트랩 환경을 업그레이드하는 경우 새 버킷이 생성되면 이전 Amazon S3 버킷이 분리됩니다. Amazon S3 콘솔을 사용하여 수동으로 삭제합니다.

프로젝트 초기화

비어 있는 새 GitHub 프로젝트를 만들고 디렉터리의 워크스테이션에 복제합니다`my-pipeline`. (이 항목의 코드 예제에서는 다음을 사용합니다 GitHub. AWS CodeStar 또는 를 사용할 수도 AWS CodeCommit있습니다.

```
git clone GITHUB-CLONE-URL my-pipeline
cd my-pipeline
```

> **Note**
>
> 앱의 기본 디렉터리가 아닌 `my-pipeline` 다른 이름을 사용할 수 있습니다. 하지만 이렇게 하면 이 주제 뒷부분에서 파일 및 클래스 이름을 수정해야 합니다. 이는 AWS CDK 툴킷이 일부 파일 및 클래스 이름을 기본 디렉토리 이름을 기반으로 하기 때문입니다.

복제한 후 평소와 같이 프로젝트를 초기화하십시오.

TypeScript

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

JavaScript

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

Python

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

앱을 만든 후 다음 두 명령도 입력합니다. 이는 앱의 Python 가상 환경을 활성화하고 AWS CDK 핵심 종속성을 설치합니다.

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

Java

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

IDE를 사용하는 경우 이제 프로젝트를 열거나 가져올 수 있습니다. 예를 들어 Eclipse에서는 [파일] > [가져오기] > [Maven] > [기존 Maven 프로젝트] 를 선택합니다. 프로젝트 설정이 Java 8 (1.8) 을 사용하도록 설정되어 있는지 확인하십시오.

C#

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

Visual Studio를 사용하는 경우 `src` 디렉터리에서 솔루션 파일을 여십시오.

Go

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

앱을 만든 후에는 다음 명령도 입력하여 앱에 필요한 AWS Construct Library 모듈을 설치합니다.

```
go get
```

> **Important**
>
> `cdk.json`및 `cdk.context.json` 파일을 소스 컨트롤에 커밋해야 합니다. 컨텍스트 정보 (예: AWS 계정에서 검색된 기능 플래그 및 캐시된 값) 는 프로젝트 상태의 일부입니다. 다른 환경에서는 값이 다를 수 있으며, 이로 인해 결과에 예상치 못한 변화가 발생할 수 있습니다. 자세한 설명은 the section called “컨텍스트” 섹션을 참조하세요.

파이프라인을 정의하세요.

CDK Pipelines 애플리케이션에는 적어도 두 개의 스택이 포함됩니다. 하나는 파이프라인 자체를 나타내는 스택이고 다른 하나는 파이프라인을 통해 배포된 애플리케이션을 나타내는 스택입니다. 또한 스택을 여러 단계로 그룹화하여 인프라 스택의 복사본을 여러 환경에 배포할 수 있습니다. 지금은 파이프라인을 살펴보고 나중에 배포할 애플리케이션을 자세히 살펴보겠습니다.

`CodePipeline`구문은 배포 AWS CodePipeline 엔진으로 사용하는 CDK 파이프라인을 나타내는 구조입니다. `CodePipeline`스택에서 인스턴스화할 때는 파이프라인의 소스 위치 (예: 리포지토리) 를 GitHub 정의합니다. 또한 앱을 빌드하기 위한 명령도 정의합니다.

예를 들어, 다음은 소스가 GitHub 리포지토리에 저장되는 파이프라인을 정의합니다. 또한 TypeScript CDK 애플리케이션을 위한 빌드 단계도 포함됩니다. 표시된 곳에 GitHub 리포지토리에 대한 정보를 입력하세요.

> **Note**
>
> 기본적으로 파이프라인은 Secrets Manager에 다음과 같은 이름으로 `github-token` 저장된 개인용 액세스 토큰을 GitHub 사용하여 인증합니다.

또한 AWS 계정과 지역을 지정하도록 파이프라인 스택의 인스턴스화를 업데이트해야 합니다.

TypeScript

에서 `lib/my-pipeline-stack.ts` (프로젝트 폴더 이름이 `my-pipeline` 지정되지 않은 경우 달라질 수 있음):

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

에서 `bin/my-pipeline.ts` (프로젝트 폴더 이름이 지정되지 않은 경우 다를 수 `my-pipeline` 있음):

```
#!/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

에서 `lib/my-pipeline-stack.js` (프로젝트 폴더 이름이 지정되지 않은 경우 다를 수 `my-pipeline` 있음):

```
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 }
```

에서 `bin/my-pipeline.js` (프로젝트 폴더 이름이 지정되지 않은 경우 다를 수 `my-pipeline` 있음):

```
#!/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

에서 `my-pipeline/my-pipeline-stack.py` (프로젝트 폴더 이름이 지정되지 않은 경우 다를 수 `my-pipeline` 있음):

```
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"]
                        )
                    )
```

app.py에서:

```
#!/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

에서 src/main/java/com/myorg/MyPipelineStack.java (프로젝트 폴더 이름이 지정되지 않은 경우 다를 수 my-pipeline 있음):

```
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();
    }
}
```

에서 `src/main/java/com/myorg/MyPipelineApp.java` (프로젝트 폴더 이름이 지정되지 않은 경우 다를 수 `my-pipeline` 있음):

```
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#

에서 src/MyPipeline/MyPipelineStack.cs (프로젝트 폴더 이름이 지정되지 않은 경우 다를 수 my-pipeline 있음):

```
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" }
                })
            });
        }
    }
}
```

에서 src/MyPipeline/Program.cs (프로젝트 폴더 이름이 지정되지 않은 경우 다를 수 my-pipeline 있음):

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

파이프라인을 수동으로 한 번 배포해야 합니다. 그 후에는 파이프라인이 소스 코드 리포지토리에서 최신 상태로 유지됩니다. 따라서 리포지토리의 코드가 배포하려는 코드인지 확인하세요. 변경 사항을 확인하고 GitHub 다음으로 푸시한 다음 배포하십시오.

```
git add --all
git commit -m "initial commit"
git push
cdk deploy
```

> **Tip**
>
> 이제 초기 배포가 완료되었으므로 로컬 AWS 계정에는 더 이상 관리 액세스 권한이 필요하지 않습니다. 앱의 모든 변경 사항이 파이프라인을 통해 배포되기 때문입니다. 푸시만 하면 GitHub 됩니다.

적용 단계

파이프라인에 한 번에 추가할 수 있는 멀티스택 AWS 애플리케이션을 정의하려면 의 서브클래스를 정의하십시오. `Stage` (이는 CDK Pipelines 모듈과 다릅니다`CdkStage`.)

스테이지에는 애플리케이션을 구성하는 스택이 포함됩니다. 스택 간에 종속성이 있는 경우 스택이 올바른 순서로 파이프라인에 자동으로 추가됩니다. 서로 종속되지 않는 스택은 병렬로 배포됩니다. 를 호출하여 스택 간의 종속성 관계를 추가할 수 있습니다. `stack1.addDependency(stack2)`

스테이지는 기본 env 인수를 받아들이며, 이 인수는 내부 스택의 기본 환경이 됩니다. (스택에는 여전히 자체 환경을 지정할 수 있습니다.)

의 Stage인스턴스를 addStage() 호출하여 파이프라인에 애플리케이션을 추가합니다. 단계를 인스턴스화하고 파이프라인에 여러 번 추가하여 DTAP 또는 다중 지역 애플리케이션 파이프라인의 여러 단계를 정의할 수 있습니다.

간단한 Lambda 함수를 포함하는 스택을 생성하고 해당 스택을 스테이지에 배치합니다. 그런 다음 파이프라인에 스테이지를 추가하여 배포할 수 있도록 하겠습니다.

TypeScript

Lambda 함수를 포함하는 애플리케이션 `lib/my-pipeline-lambda-stack.ts` 스택을 보관할 새 파일을 생성합니다.

```
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";')
      });
    }
}
```

스테이지를 보관할 새 `lib/my-pipeline-app-stage.ts` 파일을 생성하십시오.

```
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');
    }
}
```

`lib/my-pipeline-stack.ts`편집하여 파이프라인에 스테이지를 추가하세요.

```
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

Lambda 함수를 포함하는 애플리케이션 `lib/my-pipeline-lambda-stack.js` 스택을 보관할 새 파일을 생성합니다.

```
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 }
```

스테이지를 보관할 새 lib/my-pipeline-app-stage.js 파일을 생성하십시오.

```
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 };
```

lib/my-pipeline-stack.ts편집하여 파이프라인에 스테이지를 추가하세요.

```
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

Lambda 함수를 포함하는 애플리케이션 my_pipeline/my_pipeline_lambda_stack.py 스택을 보관할 새 파일을 생성합니다.

```
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';")
        )
```

스테이지를 보관할 새 my_pipeline/my_pipeline_app_stage.py 파일을 생성하십시오.

```
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")
```

my_pipeline/my-pipeline-stack.py편집하여 파이프라인에 스테이지를 추가하세요.

```
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

Lambda 함수를 포함하는 애플리케이션 src/main/java/com.myorg/MyPipelineLambdaStack.java 스택을 보관할 새 파일을 생성합니다.

```
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();

    }

}
```

스테이지를 보관할 새 src/main/java/com.myorg/MyPipelineAppStage.java 파일을 생성하십시오.

```
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");
    }

}
```

src/main/java/com.myorg/MyPipelineStack.java편집하여 파이프라인에 스테이지를 추가하세요.

```
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#

Lambda 함수를 포함하는 애플리케이션 `src/MyPipeline/MyPipelineLambdaStack.cs` 스택을 보관할 새 파일을 생성합니다.

```
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';")
            });
        }
    }
}
```

스테이지를 보관할 새 src/MyPipeline/MyPipelineAppStage.cs 파일을 생성하십시오.

```
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");
        }
    }
}
```

src/MyPipeline/MyPipelineStack.cs편집하여 파이프라인에 스테이지를 추가하세요.

```
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"
                }
            }));
        }
    }
}
```

에서 추가하는 모든 애플리케이션 단계는 해당 파이프라인 단계를 추가합니다. 이 단계는 addStage() 호출에서 반환된 StageDeployment인스턴스로 표시됩니다. addStage() 또는 메서드를 호출하여 배포 전 또는 배포 후 작업을 스테이지에 추가할 수 있습니다. addPre() addPost()

TypeScript

```
// import { ManualApprovalStep } from 'aws-cdk-lib/pipelines';

const testingStage = pipeline.addStage(new MyPipelineAppStage(this, 'testing', {
  env: { account: '111111111111', region: 'eu-west-1' }
}));

    testingStage.addPost(new ManualApprovalStep('approval'));
```

JavaScript

```
// const { ManualApprovalStep } = require('aws-cdk-lib/pipelines');

const testingStage = pipeline.addStage(new MyPipelineAppStage(this, 'testing', {
```

```
  env: { account: '111111111111', region: 'eu-west-1' }
}));

testingStage.addPost(new ManualApprovalStep('approval'));
```

Python

```
# from aws_cdk.pipelines import ManualApprovalStep

testing_stage = pipeline.add_stage(MyPipelineAppStage(self, "testing",
    env=cdk.Environment(account="111111111111", region="eu-west-1")))

testing_stage.add_post(ManualApprovalStep('approval'))
```

Java

```
// import software.amazon.awscdk.pipelines.StageDeployment;
// import software.amazon.awscdk.pipelines.ManualApprovalStep;

StageDeployment testingStage =
        pipeline.addStage(new MyPipelineAppStage(this, "test", StageProps.builder()
                .env(Environment.builder()
                        .account("111111111111")
                        .region("eu-west-1")
                        .build())
                .build()));

testingStage.addPost(new ManualApprovalStep("approval"));
```

C#

```
var testingStage = pipeline.AddStage(new MyPipelineAppStage(this, "test", new
 StageProps
{
    Env = new Environment
    {
        Account = "111111111111", Region = "eu-west-1"
    }
}));

testingStage.AddPost(new ManualApprovalStep("approval"));
```

예를 들어 스테이지를 여러 계정 또는 지역에 배포할 때 Wave에 스테이지를 추가하여 병렬로 배포할 수 있습니다.

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"
    }
}));
```

배포 테스트

CDK 파이프라인에 단계를 추가하여 수행 중인 배포를 검증할 수 있습니다. 예를 들어 CDK 파이프라인 라이브러리를 사용하여 다음과 같은 작업을 수행할 수 있습니다. `ShellStep`

- Lambda 함수가 지원하는 새로 배포된 Amazon API Gateway에 액세스하려고 합니다.
- 명령을 실행하여 배포된 리소스의 설정을 확인합니다. AWS CLI

가장 간단한 형태로 유효성 검사 작업을 추가하는 방법은 다음과 같습니다.

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'" }
}));
```

AWS CloudFormation 배포를 많이 하면 예측할 수 없는 이름을 가진 리소스가 생성됩니다. 따라서 CDK Pipeline은 배포 후 출력을 AWS CloudFormation 읽을 수 있는 방법을 제공합니다. 이를 통해 (예를 들어) 로드 밸런서의 생성된 URL을 테스트 작업에 전달할 수 있습니다.

출력을 사용하려면 관심 있는 `CfnOutput` 객체를 노출하세요. 그런 다음 단계 `envFromCfnOutputs` 속성에 전달하여 해당 단계 내에서 환경 변수로 사용할 수 있도록 하세요.

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" }
}));
```

에서 바로 간단한 검증 테스트를 작성할 수 있지만 테스트가 몇 줄 이상이면 이 방법을 다루기가 어려워집니다. `ShellStep` 더 복잡한 테스트의 경우 전체 셸 스크립트 또는 다른 언어로 작성된 프로그램과 같은 추가 파일을 `ShellStep` via 속성으로 가져올 수 있습니다. `inputs` 입력은 소스 (예: GitHub 리포지토리) 등을 포함하여 출력이 있는 모든 단계가 될 수 있습니다. `ShellStep`

파일을 테스트에서 직접 사용할 수 있는 경우 (예: 파일 자체가 실행 가능한 경우) 소스 리포지토리에서 파일을 가져오는 것이 좋습니다. 이 예시에서는 GitHub 저장소를 `source` (의 일부로 인라인으로 인스턴스화하는 대신) 로 선언합니다. `CodePipeline` 그런 다음 이 파일 세트를 파이프라인과 검증 테스트 모두에 전달합니다.

TypeScript

```
const source = CodePipelineSource.gitHub('OWNER/REPO', 'main');

const pipeline = new CodePipeline(this, 'Pipeline', {
```

```
  pipelineName: 'MyPipeline',
  synth: new ShellStep('Synth', {
    input: source,
    commands: ['npm ci', 'npm run build', 'npx cdk synth']
  })
});

const stage = pipeline.addStage(new MyPipelineAppStage(this, 'test', {
  env: { account: '111111111111', region: 'eu-west-1' }
}));

stage.addPost(new ShellStep('validate', {
  input: source,
  commands: ['sh ../tests/validate.sh']
}));
```

JavaScript

```
const source = CodePipelineSource.gitHub('OWNER/REPO', 'main');

const pipeline = new CodePipeline(this, 'Pipeline', {
  pipelineName: 'MyPipeline',
  synth: new ShellStep('Synth', {
    input: source,
    commands: ['npm ci', 'npm run build', 'npx cdk synth']
  })
});

const stage = pipeline.addStage(new MyPipelineAppStage(this, 'test', {
  env: { account: '111111111111', region: 'eu-west-1' }
}));

stage.addPost(new ShellStep('validate', {
  input: source,
  commands: ['sh ../tests/validate.sh']
}));
```

Python

```
source   = CodePipelineSource.git_hub("OWNER/REPO", "main")

pipeline =  CodePipeline(self, "Pipeline",
                pipeline_name="MyPipeline",
```

```
                synth=ShellStep("Synth",
                    input=source,
                    commands=["npm install -g aws-cdk",
                        "python -m pip install -r requirements.txt",
                        "cdk synth"]))

stage = pipeline.add_stage(MyApplicationStage(self, "test",
            env=cdk.Environment(account="111111111111", region="eu-west-1")))

stage.add_post(ShellStep("validate", input=source,
    commands=["sh ../tests/validate.sh"],
))
```

Java

```
final CodePipelineSource source = CodePipelineSource.gitHub("OWNER/REPO", "main");

final CodePipeline pipeline = CodePipeline.Builder.create(this, "pipeline")
        .pipelineName("MyPipeline")
        .synth(ShellStep.Builder.create("Synth")
                .input(source)
                .commands(Arrays.asList("npm install -g aws-cdk", "cdk synth"))
                .build())
        .build();

final StageDeployment stage =
        pipeline.addStage(new MyPipelineAppStage(this, "test", StageProps.builder()
                .env(Environment.builder()
                        .account("111111111111")
                        .region("eu-west-1")
                        .build())
                .build()));

stage.addPost(ShellStep.Builder.create("validate")
        .input(source)
        .commands(Arrays.asList("sh ../tests/validate.sh"))
        .build());
```

C#

```
var source = CodePipelineSource.GitHub("OWNER/REPO", "main");

var pipeline = new CodePipeline(this, "pipeline", new CodePipelineProps
```

```
{
    PipelineName = "MyPipeline",
    Synth = new ShellStep("Synth", new ShellStepProps
    {
        Input = source,
        Commands = new string[] { "npm install -g aws-cdk", "cdk synth" }
    })
});

var stage = pipeline.AddStage(new MyPipelineAppStage(this, "test", new StageProps
{
    Env = new Environment
    {
        Account = "111111111111", Region = "eu-west-1"
    }
}));

stage.AddPost(new ShellStep("validate", new ShellStepProps
{
    Input = source,
    Commands = new string[] { "sh ../tests/validate.sh" }
}));
```

합성의 일부로 수행되는 테스트를 컴파일해야 하는 경우에는 신디사이저 단계에서 추가 파일을 가져오는 것이 좋습니다.

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" }
}));
```

보안 참고 사항

모든 형태의 지속적 전송에는 보안 위험이 내재되어 있습니다. AWS 공동 책임 모델에 따라 클라우드 내 정보의 보안에 대한 책임은 귀하에게 있습니다. AWS CDK Pipelines 라이브러리는 보안 기본값과 모델링 모범 사례를 통합하여 한 발 앞서 나갈 수 있게 해줍니다.

그러나 라이브러리의 특성상 의도한 목적을 달성하기 위해 높은 수준의 액세스가 필요한 라이브러리는 완벽한 보안을 보장할 수 없습니다. 조직 외부에는 많은 공격 벡터가 AWS 있습니다.

특히 다음 사항에 유의하세요.

- 사용하고 있는 소프트웨어를 염두에 두세요. 파이프라인에서 실행하는 모든 타사 소프트웨어를 검토하세요. 배포되는 인프라가 변경될 수 있기 때문입니다.
- 종속성 잠금을 사용하여 실수로 인한 업그레이드를 방지하세요. CDK Pipelines는 사용자의 종속성을 `package-lock.json` `yarn.lock` 존중하고 사용자가 기대하는 것과 일치하는지 확인합니다.
- CDK Pipelines는 자체 계정에서 생성된 리소스에서 실행되며, 이러한 리소스의 구성은 파이프라인을 통해 코드를 제출하는 개발자가 제어합니다. 따라서 CDK Pipelines 자체로는 규정 준수 검사를 우회하려는 악의적인 개발자로부터 보호할 수 없습니다. 위협 모델에 개발자가 CDK 코드를 작성하는 것을 포함하는 경우 실행 역할에 비활성화할 권한이 없는 AWS CloudFormation Hooks (예방) 또는 AWS Config (반응형) 와 같은 외부 규정 준수 메커니즘을 마련해야 합니다. AWS CloudFormation

- 프로덕션 환경의 자격 증명은 수명이 짧아야 합니다. 부트스트래핑과 초기 프로비저닝 후에는 개발자가 계정 자격 증명을 보유할 필요가 전혀 없습니다. 파이프라인을 통해 변경 내용을 배포할 수 있습니다. 처음부터 자격 증명이 필요하지 않으므로 자격 증명이 유출될 가능성을 줄이세요.

문제 해결

CDK Pipelines를 시작할 때 일반적으로 발생하는 문제는 다음과 같습니다.

파이프라인: 내부 장애

```
CREATE_FAILED  | AWS::CodePipeline::Pipeline | Pipeline/Pipeline
Internal Failure
```

GitHub 액세스 토큰을 확인하세요. 누락되었거나 리포지토리에 액세스할 수 있는 권한이 없을 수 있습니다.

키: 정책에 잘못된 보안 주체가 하나 이상 포함된 명령문이 포함되어 있습니다.

```
CREATE_FAILED | AWS::KMS::Key | Pipeline/Pipeline/ArtifactsBucketEncryptionKey
Policy contains a statement with one or more invalid principals.
```

대상 환경 중 하나가 새 부트스트랩 스택으로 부트스트랩되지 않았습니다. 모든 대상 환경이 부트스트랩되었는지 확인하십시오.

스택이 ROLLBACK_COMPLETE 상태이므로 업데이트할 수 없습니다.

```
Stack STACK_NAME is in ROLLBACK_COMPLETE state and can not be updated. (Service:
AmazonCloudFormation; Status Code: 400; Error Code: ValidationError; Request
ID: ...)
```

스택이 이전 배포에 실패하여 재시도할 수 없는 상태입니다. AWS CloudFormation 콘솔에서 스택을 삭제하고 배포를 다시 시도합니다.

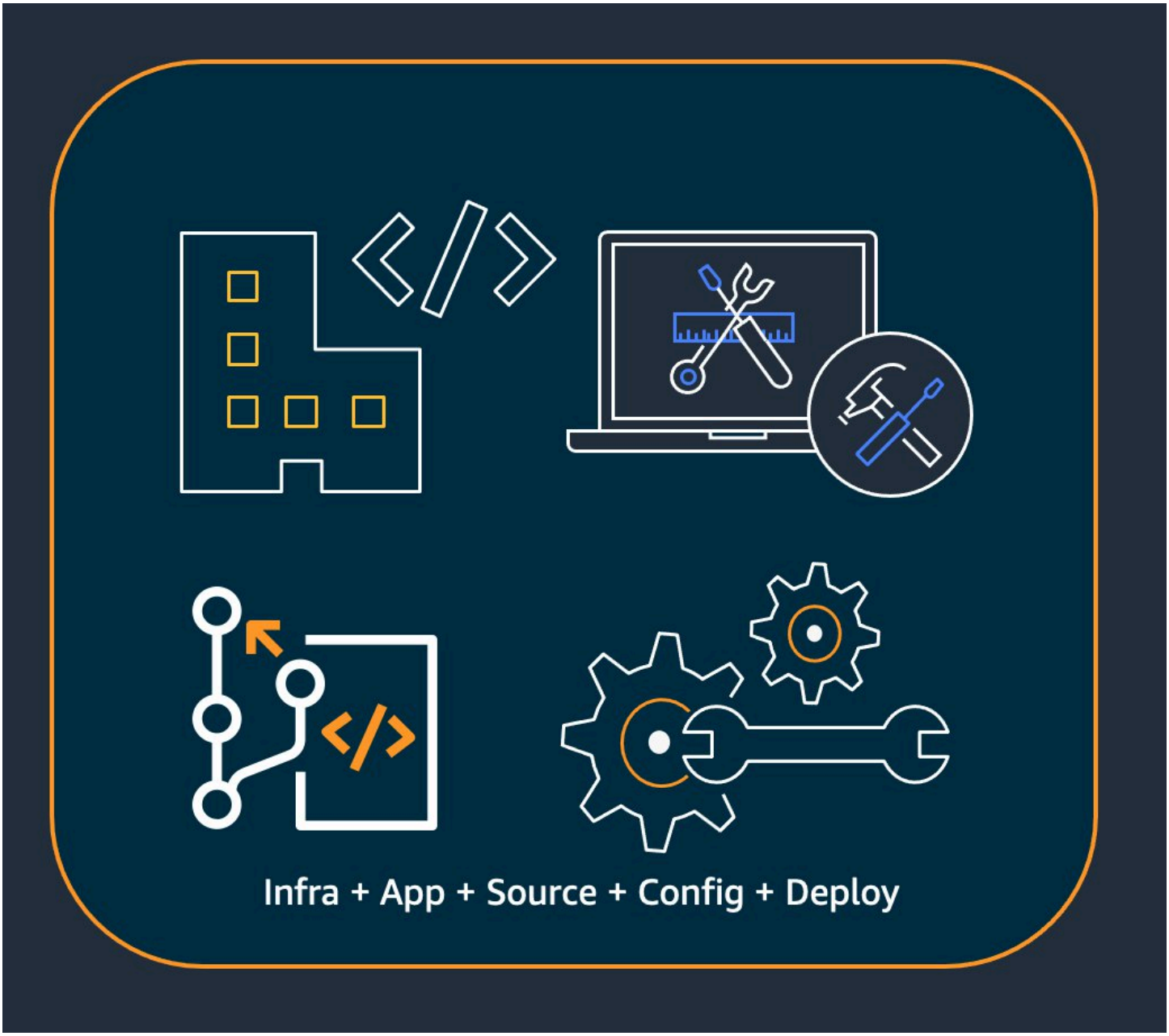
를 사용한 클라우드 인프라 개발 및 배포 모범 사례 AWS CDK

를 AWS CDK사용하면 개발자 또는 관리자가 지원되는 프로그래밍 언어를 사용하여 클라우드 인프라를 정의할 수 있습니다. CDK 애플리케이션은 API, 데이터베이스, 모니터링 리소스와 같은 논리적 단위로 구성되어야 하며, 선택적으로 자동 배포를 위한 파이프라인이 있어야 합니다. 논리 유닛은 다음을 포함하는 구성으로 구현되어야 합니다.

- 인프라 (예: 아마존 S3 버킷, 아마존 RDS 데이터베이스 또는 아마존 VPC 네트워크)
- 런타임 코드 (예: 함수) AWS Lambda
- 구성 코드

스택은 이러한 논리 유닛의 배포 모델을 정의합니다. CDK의 기본 개념에 대한 자세한 소개는 를 참조하십시오. 시작하기

이는 복잡한 클라우드 애플리케이션의 배포 및 지속적인 유지 관리 과정에서 자주 발생하는 장애 패턴과 고객 및 내부 팀의 요구 사항에 대한 세심한 고려를 AWS CDK 반영합니다. 우리는 장애가 구성 변경과 같이 완전히 테스트되지 않은 응용 프로그램의 변경 out-of-band ""“과 관련된 경우가 많다는 것을 발견했습니다. 따라서 비즈니스 로직뿐만 아니라 인프라 및 구성 등 전체 애플리케이션이 코드로 정의되는 모델을 AWS CDK 중심으로 개발했습니다. 이렇게 하면 제안된 변경 사항을 주의 깊게 검토하고, 다양한 수준의 프로덕션 환경과 유사한 환경에서 종합적으로 테스트하고, 문제가 발생할 경우 완전히 롤백할 수 있습니다.



배포 시 는 다음을 포함하는 클라우드 어셈블리를 AWS CDK 합성합니다.

- AWS CloudFormation 모든 대상 환경의 인프라를 설명하는 템플릿
- 런타임 코드와 해당 지원 파일이 포함된 파일 자산

CDK를 사용하면 애플리케이션의 기본 버전 관리 브랜치에 있는 모든 커밋을 완전하고 일관되며 배포 가능한 애플리케이션 버전으로 표현할 수 있습니다. 그러면 변경 사항이 있을 때마다 애플리케이션을 자동으로 배포할 수 있습니다.

이를 뒷받침하는 철학은 권장 모범 사례로 AWS CDK 이어지며, 이를 크게 네 가지 범주로 나누었습니다.

- the section called “조직 모범 사례”
- the section called “코딩 모범 사례”
- the section called “모범 사례 구축”
- the section called “애플리케이션 모범 사례”

> **Tip**
>
> 또한 CDK 정의 인프라에 적용할 수 있는 경우 해당 모범 AWS CloudFormation사례와 사용하는 개별 AWS 서비스를 고려하세요.

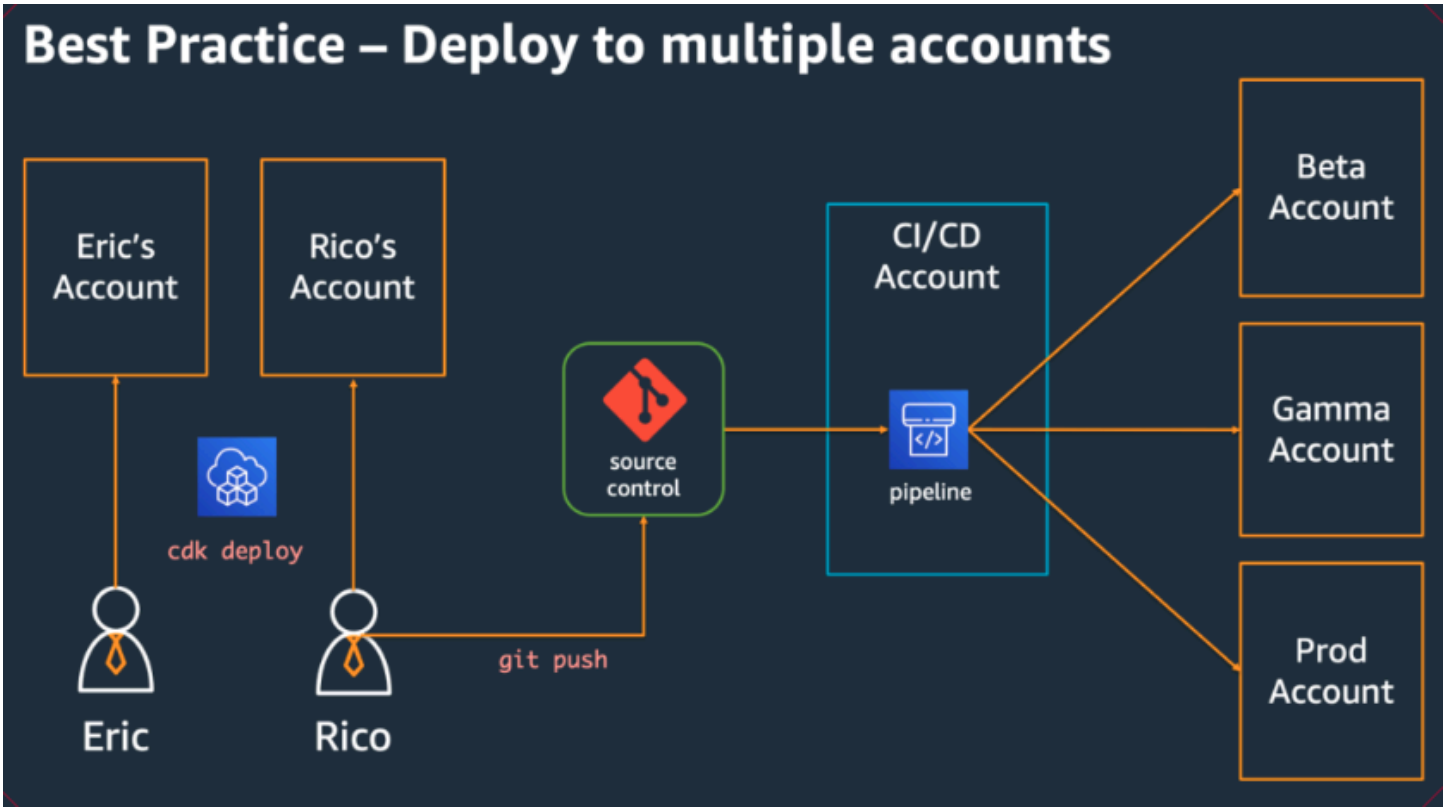
조직 모범 사례

AWS CDK 도입 초기 단계에서는 조직을 성공으로 이끄는 방법을 고려하는 것이 중요합니다. CDK를 채택하는 과정에서 회사의 나머지 구성원을 교육하고 안내할 책임이 있는 전문가 팀을 두는 것이 가장 좋습니다. 이 팀의 규모는 소규모 회사의 경우 한두 명에서 대규모 회사의 본격적인 Cloud Center of Excellence (CCoE) 에 이르기까지 다양할 수 있습니다. 이 팀은 회사의 클라우드 인프라에 대한 표준과 정책을 수립하고 개발자를 교육하고 멘토링하는 일을 담당합니다.

CCoE는 클라우드 인프라에 사용해야 하는 프로그래밍 언어에 대한 지침을 제공할 수 있습니다. 세부 사항은 조직마다 다르지만 올바른 정책은 개발자가 회사의 클라우드 인프라를 이해하고 유지 관리할 수 있도록 하는 데 도움이 됩니다.

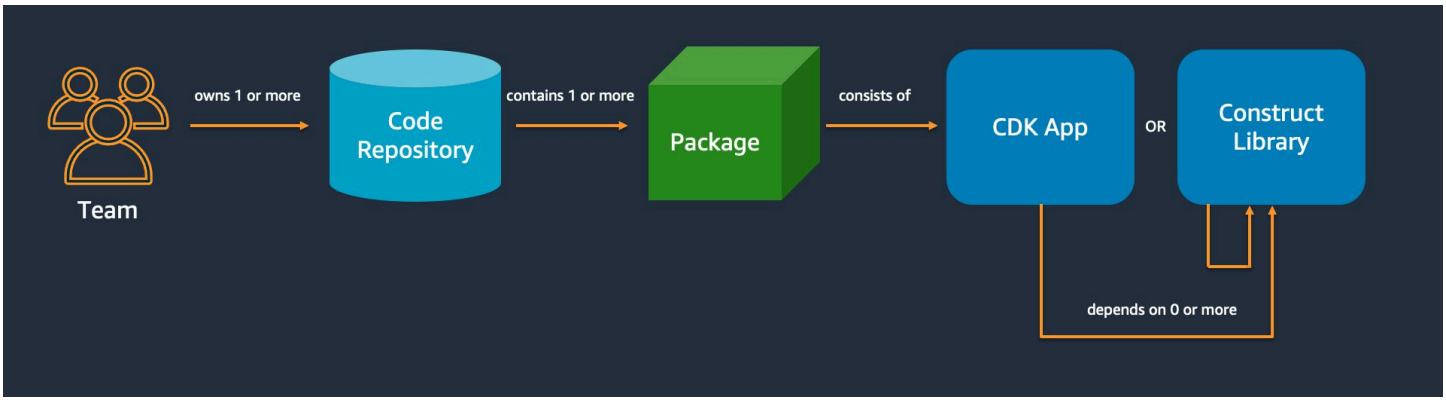
CCoE는 또한 조직 단위를 정의하는 “랜딩 존 (landing zone)" 을 생성합니다. AWS Landing Zone은 모범 사례 청사진을 기반으로 사전 구성되고 안전하며 확장 가능한 다중 계정 AWS 환경입니다. Landing Zone을 구성하는 서비스를 통합하려면 단일 사용자 인터페이스에서 전체 다중 계정 시스템을 구성하고 관리하는 방법을 사용할 AWS Control Tower수 있습니다.

개발팀은 필요에 따라 자체 계정을 사용하여 이러한 계정에 새 리소스를 테스트하고 배포할 수 있어야 합니다. 개별 개발자는 이러한 리소스를 자체 개발 워크스테이션의 확장으로 취급할 수 있습니다. CDK Pipelines를 사용하면 CI/CD 계정을 통해 애플리케이션을 테스트, 통합 및 프로덕션 환경 (각각 자체 AWS 지역 또는 계정으로 격리됨) 에 배포할 수 있습니다. AWS CDK 이는 개발자 코드를 조직의 표준 리포지토리에 병합하여 수행됩니다.



코딩 모범 사례

이 섹션에서는 AWS CDK 코드를 구성하는 모범 사례를 제공합니다. 다음 다이어그램은 팀과 해당 팀의 코드 리포지토리, 패키지, 응용 프로그램 및 구성 라이브러리 간의 관계를 보여줍니다.



간단하게 시작해서 필요할 때만 복잡성을 더하세요.

대부분의 모범 사례의 기본 원칙은 일을 최대한 단순하게 유지하되 단순하게 유지하는 것입니다. 요구 사항에 따라 더 복잡한 솔루션이 필요한 경우에만 복잡성을 추가하십시오. 를 사용하면 필요에 따라

AWS CDK코드를 리팩터링하여 새 요구 사항을 지원할 수 있습니다. 가능한 모든 시나리오를 미리 설계할 필요는 없습니다.

AWS Well-Architected 프레임워크에 맞춰 조정

AWS Well-Architected 프레임워크는 구성 요소를 요구 사항에 부합하는 코드, 구성 AWS 및 리소스로 정의합니다. 구성 요소는 종종 기술 소유권의 단위이며 다른 구성 요소와 분리됩니다. 워크로드라는 용어는 함께 비즈니스 가치를 제공하는 구성 요소 집합을 식별하는 데 사용됩니다. 워크로드는 일반적으로 비즈니스 및 기술 리더가 소통하는 세부 수준입니다.

AWS CDK 응용 프로그램은 AWS Well-Architected 프레임워크에 정의된 구성 요소에 매핑됩니다. AWS CDK 앱은 Well-Architected 클라우드 애플리케이션 모범 사례를 체계화하고 제공하는 메커니즘입니다. 다음과 같은 아티팩트 리포지토리를 통해 구성 요소를 재사용 가능한 코드 라이브러리로 만들고 공유할 수도 있습니다. AWS CodeArtifact

모든 애플리케이션은 단일 리포지토리의 단일 패키지로 시작됩니다.

단일 패키지는 AWS CDK 앱의 진입점입니다. 여기서는 애플리케이션의 다양한 논리 유닛을 배포하는 방법과 위치를 정의합니다. 또한 애플리케이션을 배포하기 위한 CI/CD 파이프라인을 정의합니다. 앱 구조는 솔루션의 논리적 단위를 정의합니다.

둘 이상의 응용 프로그램에서 사용하는 구문에는 추가 패키지를 사용하십시오. (또한 공유 구조에는 고유한 수명 주기 및 테스트 전략이 있어야 합니다.) 동일한 리포지토리에 있는 패키지 간의 종속성은 리포지토리의 빌드 도구를 통해 관리됩니다.

가능하긴 하지만, 특히 자동화된 배포 파이프라인을 사용하는 경우에는 여러 애플리케이션을 동일한 리포지토리에 두지 않는 것이 좋습니다. 이렇게 하면 배포 중 변경 사항의 “폭발 반경”이 늘어납니다. 저장소에 여러 응용 프로그램이 있는 경우 다른 응용 프로그램이 변경되지 않았더라도 한 응용 프로그램을 변경하면 다른 응용 프로그램도 배포됩니다. 또한 한 응용 프로그램이 손상되면 다른 응용 프로그램을 배포할 수 없습니다.

코드 수명 주기 또는 팀 소유권을 기반으로 코드를 리포지토리로 이동합니다.

패키지가 여러 애플리케이션에서 사용되기 시작하면 패키지를 자체 리포지토리로 이동하십시오. 이렇게 하면 패키지를 사용하는 애플리케이션 빌드 시스템에서 패키지를 참조할 수 있으며 애플리케이션 수명 주기와 관계없이 케이던스에 따라 패키지를 업데이트할 수도 있습니다. 하지만 처음에는 모든 공유 구문을 하나의 리포지토리에 넣는 것이 합리적일 수 있습니다.

또한 여러 팀에서 작업하는 경우 패키지를 자체 저장소로 옮기세요. 이렇게 하면 액세스 제어를 강화하는 데 도움이 됩니다.

여러 리포지토리에서 패키지를 사용하려면 NPM 또는 Maven Central과 비슷하지만 조직 내부에 있는 개인 패키지 리포지토리가 필요합니다. PyPi 또한 패키지를 빌드하고 테스트하여 개인 패키지 저장소에 게시하는 릴리스 프로세스도 필요합니다. CodeArtifact가장 많이 사용되는 프로그래밍 언어의 패키지를 호스팅할 수 있습니다.

패키지 저장소의 패키지 종속성은 해당 언어의 패키지 관리자 (예: NPM for TypeScript 또는 JavaScript application) 에서 관리합니다. 패키지 관리자는 빌드가 반복 가능한지 확인하는 데 도움을 줍니다. 애플리케이션이 사용하는 모든 패키지의 특정 버전을 기록하여 이를 수행합니다. 또한 제어된 방식으로 이러한 종속성을 업그레이드할 수 있습니다.

공유 패키지에는 다른 테스트 전략이 필요합니다. 단일 애플리케이션의 경우 애플리케이션을 테스트 환경에 배포하고 제대로 작동하는지 확인하는 것으로 충분할 수 있습니다. 하지만 공유 패키지는 일반 대중에게 공개되는 것처럼 사용 중인 애플리케이션과 독립적으로 테스트해야 합니다. (조직에서 일부 공유 패키지를 실제로 공개하도록 선택할 수도 있습니다.)

구문은 임의로 단순하거나 복잡할 수 있다는 점을 염두에 두세요. `BucketA`는 구조이지만 구조체일 `CameraShopWebsite` 수도 있습니다.

인프라 및 런타임 코드는 동일한 패키지에 있습니다.

인프라 배포를 위한 AWS CloudFormation 템플릿을 생성하는 것 외에도 Lambda 함수 및 Docker 이미지와 같은 런타임 자산을 번들로 제공하여 인프라와 함께 배포합니다. AWS CDK 따라서 인프라를 정의하는 코드와 런타임 로직을 구현하는 코드를 단일 구조로 결합할 수 있습니다. 이렇게 하는 것이 가장 좋습니다. 이 두 종류의 코드는 별도의 리포지토리나 패키지에 있을 필요가 없습니다.

두 종류의 코드를 함께 발전시키려면 인프라 및 로직을 포함한 기능을 완전히 설명하는 독립형 구문을 사용할 수 있습니다. 독립형 구문을 사용하면 두 종류의 코드를 분리하여 테스트하고, 여러 프로젝트에서 코드를 공유 및 재사용하고, 모든 코드의 버전을 동기화할 수 있습니다.

모범 사례 구축

이 섹션에는 구문 개발을 위한 모범 사례가 포함되어 있습니다. 구문은 리소스를 캡슐화하는 재사용 가능한 구성 가능한 모듈입니다. 이는 앱의 구성 요소입니다. AWS CDK

구문을 이용한 모델링, 스택으로 배포

스택은 배포 단위입니다. 스택의 모든 항목이 함께 배포됩니다. 따라서 여러 AWS 리소스에서 애플리케이션의 상위 수준 논리 유닛을 구축할 때는 각 논리 유닛을 a가 아닌 `Construct`a로 표현해야 합니다. `Stack` 스택은 다양한 배포 시나리오에서 구문을 구성하고 연결하는 방법을 설명하는 용도로만 사용하십시오.

예를 들어, 논리 유닛 중 하나가 웹 사이트인 경우 이를 구성하는 구조 (예: Amazon S3 버킷, API Gateway, Lambda 함수 또는 Amazon RDS 테이블) 는 하나의 상위 수준 구조로 구성되어야 합니다. 그런 다음 배포를 위해 해당 구조를 하나 이상의 스택에 인스턴스화해야 합니다.

구축용 구성과 배포용 스택을 사용하면 인프라의 재사용 가능성을 높이고 배포 방식을 보다 유연하게 조정할 수 있습니다.

환경 변수가 아닌 속성과 메서드를 사용하여 구성합니다.

구문과 스택 내부의 환경 변수 조회는 일반적인 안티패턴입니다. 코드에서 완전히 구성할 수 있으려면 구문과 스택 모두 properties 객체를 받아들여야 합니다. 그렇지 않으면 코드가 실행될 시스템에 종속성이 생겨 추적하고 관리해야 하는 구성 정보가 더 많이 생성됩니다.

일반적으로 환경 변수 조회는 앱의 최상위 수준으로 제한되어야 합니다. AWS CDK 또한 개발 환경에서 실행하는 데 필요한 정보를 전달하는 데에도 사용해야 합니다. 자세한 설명은 the section called “환경” 섹션을 참조하세요.

인프라를 단위 테스트하세요.

모든 환경에서 빌드 시 전체 유닛 테스트 세트를 일관되게 실행하려면 합성 중에 네트워크 조회를 피하고 모든 프로덕션 단계를 코드로 모델링하세요. (이러한 모범 사례는 나중에 설명합니다.) 단일 커밋으로 인해 항상 동일한 템플릿이 생성되는 경우 생성된 템플릿이 예상대로 표시되는지 확인하기 위해 작성하는 단위 테스트를 신뢰할 수 있습니다. 자세한 설명은 테스트 구성 섹션을 참조하세요.

스테이트풀 리소스의 논리적 ID는 변경하지 마세요.

리소스의 논리 ID를 변경하면 다음 배포 시 리소스가 새 리소스로 교체됩니다. 데이터베이스 및 S3 버킷과 같은 스테이트풀 리소스나 Amazon VPC와 같은 영구 인프라의 경우 이는 거의 필요하지 않습니다. ID를 변경할 수 있는 AWS CDK 코드 리팩토링에 주의하십시오. 스테이트풀 리소스의 논리적 ID가 정적으로 유지되는지 확인하는 단위 테스트를 작성하세요. 논리적 ID는 구문을 인스턴스화할 때 `id` 지정한 값과 구문 트리에서의 구문 위치에서 파생됩니다. 자세한 설명은 the section called “논리적 ID” 섹션을 참조하세요.

구문은 규정을 준수하기에 충분하지 않습니다.

많은 기업 고객이 L2 구성 (기본 제공 기본 제공 및 모범 사례로 개별 AWS 리소스를 나타내는 "큐레이션된" 구조) 에 대한 자체 래퍼를 작성합니다. 이러한 래퍼는 정적 암호화 및 특정 IAM 정책과 같은 보안 모범 사례를 적용합니다. 예를 들어, 일반적인 Amazon S3 `Bucket` 구조 대신 생성하여 애플리케이션에서 사용할 수 있습니다. `MyCompanyBucket` 이 패턴은 소프트웨어 개발 수명 주기 초기에 보안 지침을 제시하는 데 유용하지만, 이를 유일한 적용 수단으로 삼지는 마십시오.

대신 서비스 제어 정책 및 권한 경계와 같은 AWS 기능을 사용하여 조직 수준에서 보안 가드레일을 적용하십시오. 배포 the section called "속성" 전에 CloudFormation Guard와 같은 도구를 사용하여 인프라 요소의 보안 속성을 확인해 보세요. 가장 AWS CDK 효과적인 용도로 사용하세요.

마지막으로, 직접 "L2+" 구문을 작성하면 개발자가 AWS 솔루션 구문 또는 Construct Hub의 타사 구문과 같은 AWS CDK 패키지를 활용하지 못할 수 있다는 점을 염두에 두세요. 이러한 패키지는 일반적으로 표준 구문을 기반으로 구축되며 AWS CDK 래퍼 구문을 사용할 수 없습니다.

애플리케이션 모범 사례

이 섹션에서는 구문을 결합하여 AWS 리소스가 연결되는 방식을 정의하여 AWS CDK 애플리케이션을 작성하는 방법을 설명합니다.

통합 시점에 결정을 내리세요.

배포 시 (`Conditions{ Fn::If }`, 및 사용`Parameters`) 결정을 내릴 수 있고 이러한 메커니즘에 어느 정도 AWS CDK 액세스할 수 있지만 AWS CloudFormation 사용하지 않는 것이 좋습니다. 사용할 수 있는 값 유형과 해당 값에 대해 수행할 수 있는 작업 유형은 범용 프로그래밍 언어에서 사용할 수 있는 것과 비교할 때 제한적입니다.

대신 프로그래밍 언어의 `if` 명령문 및 기타 기능을 사용하여 AWS CDK 응용 프로그램에서 인스턴스화할 구문과 같은 모든 결정을 내려보세요. 예를 들어, 목록을 반복하고 목록에 있는 각 항목의 값으로 구문을 인스턴스화하는 일반적인 CDK 관용구는 표현식을 사용해서는 불가능합니다. AWS CloudFormation

강력한 클라우드 배포에 AWS CDK 사용하는 구현 세부 AWS CloudFormation 정보로 취급할 뿐, 언어 대상으로 취급하지 마십시오. 여러분은 TypeScript Python이나 Python으로 AWS CloudFormation 템플릿을 작성하는 것이 아니라 CloudFormation 배포에 사용되는 CDK 코드를 작성하는 것입니다.

물리적 이름이 아닌 생성된 리소스 이름을 사용하세요.

이름은 소중한 자원입니다. 각 이름은 한 번만 사용할 수 있습니다. 따라서 테이블 이름이나 버킷 이름을 인프라와 애플리케이션에 하드코딩하는 경우 동일한 계정에 해당 인프라를 두 번 배포할 수 없습니다. (여기서 말하는 이름은 예를 들어 Amazon S3 버킷 구조의 `bucketName` 속성에 의해 지정된 이름입니다.)

더 나쁜 것은 교체가 필요한 리소스를 변경할 수 없다는 것입니다. Amazon DynamoDB 테이블과 같이 리소스 생성 시에만 속성을 설정할 수 있는 경우 해당 속성은 변경할 수 없습니다. `KeySchema` 이 속성을 변경하려면 새 리소스가 필요합니다. 하지만 새 리소스가 진정한 대체품이 되려면 새 리소스의 이름이 같아야 합니다. 하지만 기존 리소스가 여전히 해당 이름을 사용하고 있는 동안에는 동일한 이름을 가질 수 없습니다.

가능한 한 적은 수의 이름을 지정하는 것이 더 좋습니다. 리소스 이름을 AWS CDK 생략하면 문제가 발생하지 않는 방식으로 리소스 이름이 자동으로 생성됩니다. 테이블이 리소스로 사용된다고 가정해 보겠습니다. 그런 다음 생성된 테이블 이름을 환경 변수로 AWS Lambda 함수에 전달할 수 있습니다. AWS CDK 애플리케이션에서 테이블 이름을 로 참조할 수 `table.tableName` 있습니다. 또는 시작 시 Amazon EC2 인스턴스에서 구성 파일을 생성하거나 실제 테이블 이름을 AWS Systems Manager Parameter Store에 기록하여 애플리케이션이 해당 테이블을 읽을 수 있도록 할 수 있습니다.

필요한 위치가 다른 AWS CDK 스택인 경우 훨씬 더 간단합니다. 한 스택이 리소스를 정의하고 다른 스택이 리소스를 사용해야 한다고 가정하면 다음이 적용됩니다.

- 두 스택이 같은 AWS CDK 앱에 있는 경우 두 스택 사이에 참조를 전달하세요. 예를 들어 리소스 구성에 대한 참조를 정의 스택 () `this.stack.uploadBucket = myBucket` 의 속성으로 저장합니다. 그런 다음 리소스가 필요한 스택의 생성자에 해당 어트리뷰트를 전달합니다.
- 두 스택이 서로 다른 AWS CDK 앱에 있는 경우 정적 `from` 메서드를 사용하여 ARN, 이름 또는 기타 속성을 기반으로 외부에서 정의된 리소스를 사용하십시오. (예를 들어, DynamoDB 테이블에 사용`Table.fromArn()`). `CfnOutput`구문을 사용하여 출력에 ARN 또는 기타 필수 값을 인쇄하거나 에서 `cdk deploy` 확인할 수 있습니다. AWS Management Console또는 두 번째 앱이 첫 번째 앱에서 생성된 CloudFormation 템플릿을 읽고 `Outputs` 섹션에서 해당 값을 검색할 수도 있습니다.

제거 정책 및 로그 보존을 정의합니다.

생성한 모든 내용을 보존하는 정책을 기본값으로 설정하여 데이터 손실을 AWS CDK 방지하려는 시도입니다. 예를 들어 데이터가 포함된 리소스 (예: Amazon S3 버킷 및 데이터베이스 테이블) 에 대한 기본 제거 정책은 스택에서 제거될 때 리소스를 삭제하지 않는 것입니다. 대신 리소스는 스택에서 분리됩니다. 마찬가지로 CDK의 기본값은 모든 로그를 영구 보존하는 것입니다. 프로덕션 환경에서 이러한 기

본값을 설정하면 실제로 필요하지 않은 대량의 데이터와 그에 AWS 상응하는 요금이 빠르게 저장될 수 있습니다.

각 프로덕션 리소스에 적용할 정책을 신중하게 고려하고 그에 따라 정책을 지정하세요. 스택의 제거 및 로깅 정책을 검증하는 the section called “속성” 데 사용합니다.

배포 요구 사항에 따라 애플리케이션을 여러 스택으로 분리합니다.

애플리케이션에 필요한 스택 수에 대한 엄격한 규칙은 없습니다. 일반적으로 배포 패턴을 기반으로 결정을 내리게 됩니다. 다음 지침을 염두에 두십시오.

- 일반적으로 동일한 스택에 최대한 많은 리소스를 보관하는 것이 더 간단하므로 분리하고 싶은 경우가 아니면 함께 보관하세요.
- 상태 저장 리소스 (예: 데이터베이스) 를 상태 비저장 리소스와는 별도의 스택에 보관하는 것을 고려해 보세요. 그런 다음 스테이트풀 스택에서 종료 보호를 켤 수 있습니다. 이렇게 하면 데이터 손실 위험 없이 스테이트리스 스택의 여러 복사본을 자유롭게 제거하거나 생성할 수 있습니다.
- 스테이트풀 리소스는 구조 이름 변경에 더 민감합니다. 이름을 바꾸면 리소스가 교체되기 때문입니다. 따라서 이동하거나 이름이 변경될 가능성이 있는 구조 안에 스테이트풀 리소스를 중첩하지 마세요. 단, 캐시와 같이 상태가 손실된 경우 다시 빌드할 수 있는 경우는 예외입니다. 이는 스테이트풀 리소스를 자체 스택에 넣어야 하는 또 다른 좋은 이유입니다.

비결정적 행동을 `cdk.context.json` 피하겠다고 약속하세요.

결정성은 성공적인 배포의 핵심입니다. AWS CDK AWS CDK 앱은 주어진 환경에 배포될 때마다 기본적으로 동일한 결과를 보여야 합니다.

AWS CDK 앱은 범용 프로그래밍 언어로 작성되었으므로 임의의 코드를 실행하고, 임의의 라이브러리를 사용하고, 임의의 네트워크 호출을 할 수 있습니다. 예를 들어 앱을 합성하는 동안 AWS SDK를 사용하여 AWS 계정에서 일부 정보를 검색할 수 있습니다. 이렇게 하면 자격 증명 설정 요구 사항이 추가되고, 지연 시간이 늘어나며, 실행할 때마다 실패할 가능성은 작지만 발생할 수 있다는 점을 기억하세요. `cdk synth`

통합 중에는 절대 AWS 계정이나 리소스를 수정하지 마십시오. 앱을 합성할 때 부작용이 없어야 합니다. 인프라 변경은 AWS CloudFormation 템플릿이 생성된 후 배포 단계에서만 이루어져야 합니다. 이렇게 하면 문제가 발생할 경우 변경 내용을 자동으로 AWS CloudFormation 롤백할 수 있습니다. AWS CDK 프레임워크 내에서 쉽게 변경할 수 없는 변경 사항을 적용하려면 사용자 지정 리소스를 사용하여 배포 시 임의의 코드를 실행하세요.

엄격한 읽기 전용 호출도 반드시 안전한 것은 아닙니다. 네트워크 호출에서 반환되는 값이 변경되면 어떻게 되는지 생각해 보십시오. 인프라의 어떤 부분에 영향을 미치나요? 이미 배포된 리소스는 어떻게 되나요? 다음은 급격한 값 변경으로 인해 문제가 발생할 수 있는 두 가지 상황의 예입니다.

- Amazon VPC를 특정 지역의 모든 가용 영역에 프로비저닝하고 배포일에 AZ 수가 2개인 경우 IP 공간이 절반으로 분할됩니다. 다음 날 새 가용 영역을 AWS 시작하면 다음 배포 시 IP 공간을 3분의 1로 분할하려고 하므로 모든 서브넷을 다시 생성해야 합니다. Amazon EC2 인스턴스가 아직 실행 중이기 때문에 가능하지 않을 수 있으며 수동으로 정리해야 합니다.
- 최신 Amazon Linux 머신 이미지를 쿼리하여 Amazon EC2 인스턴스를 배포하고 다음 날 새 이미지가 릴리스되면 후속 배포에서 새 AMI를 선택하여 모든 인스턴스를 대체합니다. 예상했던 일이 아닐 수도 있습니다.

배포를 성공적으로 완료한 지 몇 개월 또는 몇 년이 지나면 AWS-side 변경이 발생할 수 있으므로 이러한 상황은 치명적일 수 있습니다. 갑자기 배포가 "이유 없이" 실패하고 오래 전에 수행한 작업과 이유를 잊어버린 것입니다.

다행히도 AWS CDK 여기에는 비결정적 값의 스냅샷을 기록하는 컨텍스트 제공자라는 메커니즘이 포함되어 있습니다. 이를 통해 향후 합성 작업에서도 처음 배포했을 때와 똑같은 템플릿을 생성할 수 있습니다. 새 템플릿에서 변경한 내용은 코드에서 변경한 내용뿐입니다. 구문의 `.fromLookup()` 메서드를 사용하면 호출 결과가 캐시됩니다. `cdk.context.json` 이 코드를 나머지 코드와 함께 버전 관리에 커밋하여 향후 CDK 앱 실행 시 동일한 값을 사용하도록 해야 합니다. CDK 툴킷에는 컨텍스트 캐시를 관리하는 명령이 포함되어 있으므로 필요할 때 특정 항목을 새로 고칠 수 있습니다. 자세한 설명은 the section called "컨텍스트" 섹션을 참조하세요.

네이티브 CDK 컨텍스트 제공자가 없는 일부 값 (AWS 또는 다른 곳에서) 이 필요한 경우 별도의 스크립트를 작성하는 것이 좋습니다. 스크립트는 값을 가져와 파일에 쓴 다음 CDK 앱에서 해당 파일을 읽어야 합니다. 일반 빌드 프로세스의 일부가 아니라 저장된 값을 새로 고치고 싶은 경우에만 스크립트를 실행하세요.

역할 및 보안 그룹을 AWS CDK 관리하도록 하세요.

AWS CDK 구성 라이브러리의 `grant()` 편리한 메서드를 사용하면 최소 범위의 권한을 사용하여 한 리소스에 다른 리소스에 대한 액세스 권한을 부여하는 AWS Identity and Access Management 역할을 생성할 수 있습니다. 예를 들어 다음과 같은 줄을 생각해 보십시오.

```
myBucket.grantRead(myLambda)
```

이 한 줄은 Lambda 함수의 역할 (사용자를 위해 생성되기도 함) 에 정책을 추가합니다. 이 역할과 해당 정책에는 직접 작성하지 않아도 CloudFormation 되는 십여 줄이 넘습니다. 는 함수가 버킷에서 읽는 데 필요한 최소한의 권한만 AWS CDK 부여합니다.

개발자가 항상 보안팀에서 만든 사전 정의된 역할을 사용하도록 요구하면 AWS CDK 코딩이 훨씬 더 복잡해집니다. 팀이 애플리케이션을 설계하는 과정에서 유연성을 크게 잃을 수 있습니다. 더 나은 대안은 서비스 제어 정책과 권한 경계를 사용하여 개발자가 가드레일 내에서 벗어나지 않도록 하는 것입니다.

모든 제작 단계를 코드로 모델링합니다.

기존 AWS CloudFormation 시나리오의 목표는 다양한 대상 환경에 특정한 구성 값을 적용한 후 다양한 대상 환경에 배포할 수 있도록 매개 변수화된 단일 아티팩트를 생성하는 것입니다. CDK에서 해당 구성을 소스 코드에 빌드할 수 있으며, 그렇게 해야 합니다. 프로덕션 환경을 위한 스택을 만들고 다른 각 단계에 대해 별도의 스택을 생성하세요. 그런 다음 각 스택의 구성 값을 코드에 입력합니다. 소스 제어에 체크인하고 싶지 않은 민감한 값에 대해서는 해당 리소스의 이름 또는 ARN을 사용하여 Secrets Manager 및 Systems Manager Parameter Store와 같은 서비스를 사용하십시오.

애플리케이션을 합성할 때 `cdk.out` 폴더에 생성된 클라우드 어셈블리에는 각 환경에 대한 별도의 템플릿이 포함됩니다. 전체 빌드는 결정적입니다. 애플리케이션에는 out-of-band 변경 사항이 없으며, 커밋을 수행해도 항상 똑같은 AWS CloudFormation 템플릿과 함께 제공되는 에셋이 생성됩니다. 따라서 유닛 테스트의 신뢰성이 훨씬 높아집니다.

모든 것을 측정하세요

사람의 개입 없이 완전하고 지속적인 배포라는 목표를 달성하려면 높은 수준의 자동화가 필요합니다. 이러한 자동화는 광범위한 모니터링을 통해서만 가능합니다. 배포된 리소스의 모든 측면을 측정하려면 지표, 경보 및 대시보드를 생성하십시오. CPU 사용량, 디스크 공간 등을 측정하는 데 그치지 마세요. 또한 비즈니스 지표를 기록하고 이러한 측정치를 사용하여 롤백과 같은 배포 결정을 자동화하세요. 에 있는 대부분의 L2 AWS CDK 구조에는 DynamoDB.Table 클래스의 메서드와 같이 메트릭을 생성하는 데 도움이 되는 편리한 `metricUserErrors()` 메서드가 있습니다.

AWS CDK 참고

이 섹션에는 에 대한 참조 정보가 들어 AWS Cloud Development Kit (AWS CDK)있습니다.

주제

- API 참조
- AWS CDK 버전 관리

API 참조

API 참조에는 AWS 구성 라이브러리 및 에서 제공하는 기타 API에 대한 정보가 들어 있습니다. AWS Cloud Development Kit (AWS CDK)대부분의 AWS 구성 라이브러리는 TypeScript 이름이 다음과 `aws-cdk-lib` 같은 단일 패키지에 들어 있습니다. 실제 패키지 이름은 언어마다 다릅니다. 지원되는 각 프로그래밍 언어에 대해 별도의 API 참조 버전이 제공됩니다.

CDK API 참조는 하위 모듈로 구성되어 있습니다. 각 모듈에는 하나 이상의 하위 모듈이 있습니다. AWS 서비스

각 하위 모듈에는 API 사용 방법에 대한 정보가 포함된 개요가 있습니다. 예를 들어, S3 개요는 Amazon Simple Storage Service (Amazon S3) 버킷에서 기본 암호화를 설정하는 방법을 보여줍니다.

AWS CDK 버전 관리

이 항목에서는 에서 버전 관리를 AWS Cloud Development Kit (AWS CDK) 처리하는 방법에 대한 참조 정보를 제공합니다.

버전 번호는 메이저 버전이라는 세 개의 숫자 버전 부분으로 구성됩니다. 마이너. 패치를 적용하고 시맨틱 버전 관리 모델을 엄격하게 준수하십시오. 즉, 안정적인 API에 대한 주요 변경 사항은 메이저 릴리스에만 국한됩니다.

마이너 릴리스와 패치 릴리스는 이전 버전과 호환됩니다. 동일한 메이저 버전의 이전 버전에서 작성된 코드를 동일한 메이저 버전 내의 새 버전으로 업그레이드할 수 있습니다. 또한 계속해서 빌드 및 실행되어 동일한 출력을 생성합니다.

주제

- AWS CDKCLI호환성

- AWS 라이브러리 버전 관리 구성
- 언어 바인딩 안정성

AWS CDKCLI호환성

AWS CDK CLI는 의미상 더 낮거나 같은 버전 번호의 구조 라이브러리와 항상 호환됩니다. 따라서 동일한 메이저 버전 AWS CDK CLI 내에서 업그레이드하는 것이 항상 안전합니다.

AWS CDK CLI는 의미상 상위 버전의 구성 라이브러리와 항상 호환되는 것은 아닙니다. 호환성은 두 구성 요소가 동일한 클라우드 어셈블리 스키마 버전을 사용하는지 여부에 따라 달라집니다. AWS CDK 프레임워크는 합성 중에 클라우드 어셈블리를 생성한 AWS CDK CLI 다음 배포에 사용합니다. 클라우드 어셈블리의 형식을 정의하는 스키마는 엄격하게 지정되고 버전이 지정됩니다.

AWS 지정된 클라우드 어셈블리 스키마 버전을 사용하는 구성 라이브러리는 해당 스키마 버전 AWS CDK CLI 이상을 사용하는 버전과 호환됩니다. 여기에는 지정된 구성 라이브러리 릴리스보다 이전의 릴리스가 포함될 수 있습니다. AWS CDK CLI

구성 라이브러리에 필요한 클라우드 어셈블리 버전이 에서 지원하는 버전과 호환되지 않는 AWS CDK CLI 경우 다음과 같은 오류 메시지가 표시됩니다.

```
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.
```

이 오류를 AWS CDK CLI 해결하려면 필수 클라우드 어셈블리 버전과 호환되는 버전 또는 사용 가능한 최신 버전으로 업데이트하십시오. 대안 (앱에서 사용하는 구성 라이브러리 모듈을 다운그레이드하는 것) 은 일반적으로 권장되지 않습니다.

> **Note**
>
> 클라우드 어셈블리 스키마에 대한 자세한 내용은 클라우드 어셈블리 버전 관리를 참조하십시오.

AWS 라이브러리 버전 관리 구성

AWS 구성 라이브러리의 모듈은 개념에서 완성된 API로 개발되면서 다양한 단계를 거칩니다. 각 단계는 후속 버전에서 다양한 수준의 API 안정성을 제공합니다. AWS CDK

기본 AWS CDK 라이브러리의 API는 `aws-cdk-lib` 안정적이며 라이브러리는 완전히 의미론적으로 버전이 관리됩니다. 이 패키지에는 모든 AWS 서비스에 대한 AWS CloudFormation (L1) 구문과 모든 안정적인 상위 수준 (L2 및 L3) 모듈이 포함되어 있습니다. (여기에는 및 와 같은 핵심 CDK 클래스도 포함됩니다.) App Stack API는 CDK의 다음 주요 릴리스가 나올 때까지는 이 패키지에서 제거되지 않을 것입니다 (지원 중단될 수도 있음). 어떤 개별 API도 주요 변경 사항을 적용하지는 않을 것입니다. 주요 변경이 필요한 경우 완전히 새로운 API가 추가됩니다.

이미 통합된 서비스를 위해 개발 중인 새 API는 접미사를 사용하여 `aws-cdk-lib` 식별됩니다. Beta*N* 접미사는 1에서 N 시작하여 새 API가 크게 변경될 때마다 증가합니다. Beta*N* API는 제거되지 않고 더 이상 사용되지 않으므로 기존 앱은 최신 버전에서 계속 작동합니다. `aws-cdk-lib` API가 안정적이라고 판단되면 접미사가 없는 새 API가 추가됩니다. Beta*N*

이전에는 L1 API만 있었던 AWS 서비스를 위해 상위 수준 (L2 또는 L3) API가 개발되기 시작하면 이러한 API는 처음에 별도의 패키지로 배포됩니다. 이러한 패키지 이름에는 접미사가 “Alpha”이고 해당 버전이 호환되는 첫 번째 버전 (하위 버전 포함) 과 일치합니다. `aws-cdk-lib` `alpha` 모듈이 의도한 사용 사례를 지원하면 해당 API가 추가됩니다. `aws-cdk-lib`

언어 바인딩 안정성

시간이 지나면 추가 프로그래밍 언어에 AWS CDK 대한 지원을 추가할 수 있습니다. 모든 언어에서 설명하는 API는 동일하지만 API가 표현되는 방식은 언어마다 다르며 언어 지원이 발전함에 따라 달라질 수 있습니다. 이러한 이유로 언어 바인딩은 프로덕션 환경에서 사용할 준비가 된 것으로 간주되기 전까지는 한동안 실험적인 것으로 간주됩니다.

Language	Stability
TypeScript	Stable
JavaScript	Stable
Python	Stable
Java	Stable
C#/.NET	Stable
Go	Stable

AWS CDK 튜토리얼

이 섹션에는 에 대한 자습서가 포함되어 있습니다. AWS Cloud Development Kit (AWS CDK)

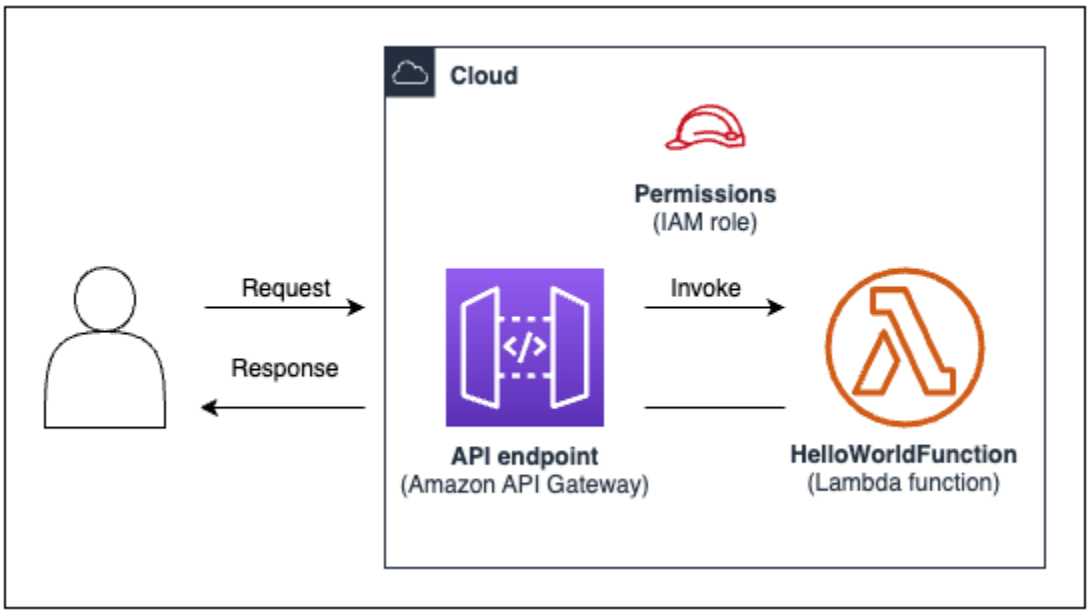
주제

서버리스 헬로 월드 애플리케이션 만들기

이 자습서에서는 를 사용하여 다음을 AWS Cloud Development Kit (AWS CDK) 생성하여 기본 API 백엔드를 구현하는 간단한 서버리스 Hello World 애플리케이션을 만듭니다.

- Amazon API Gateway REST API — HTTP GET 요청을 통해 함수를 호출하는 데 사용되는 HTTP 엔드포인트를 제공합니다.
- AWS Lambda 함수 - 엔드포인트와 함께 호출될 때 `Hello World!` 메시지를 반환하는 함수입니다. HTTP
- 통합 및 권한 — 리소스가 서로 상호 작용하고 CloudWatch Amazon에 로그를 쓰는 등의 작업을 수행할 수 있는 구성 세부 정보 및 권한.

다음 다이어그램은 이 애플리케이션의 구성 요소를 보여줍니다.



이 자습서에서는 다음을 완료합니다.

1. AWS CDK 프로젝트 만들기.
2. 구성 라이브러리의 L2 구조를 사용하여 Lambda 함수와 API Gateway REST API를 정의합니다. AWS
3. 에 애플리케이션을 배포하십시오. AWS 클라우드
4. 에서 애플리케이션과 상호 작용하십시오 AWS 클라우드.
5. AWS 클라우드에서 샘플 애플리케이션을 삭제합니다.

필수 조건

이 자습서를 시작하기 전에 다음을 완료하십시오.

- 를 AWS 계정 만들고 AWS Command Line Interface (AWS CLI) 를 설치 및 구성하십시오.
- 설치 Node.js 및npm.
- 를 사용하여 전 세계에 CDK 툴킷을 설치합니다. `npm install -g aws-cdk`

자세한 정보는 시작하기 AWS CDK을 참조하세요.

또한 다음 사항에 대한 기본적인 이해를 권장합니다.

- 이게 뭐야 AWS CDK?에 대한 기본 소개는 다음과 같습니다 AWS CDK.
- AWS CDK 개념의 핵심 개념에 대한 개요를 참조하십시오 AWS CDK.

1단계: CDK 프로젝트 만들기

이 단계에서는 명령어를 사용하여 새 CDK 프로젝트를 만듭니다. AWS CDK CLI `cdk init`

CDK 프로젝트를 만들려면

1. 선택한 시작 디렉터리에서 사용자 `cdk-hello-world` 컴퓨터에 이름이 지정된 프로젝트 디렉터리를 만들고 탐색합니다.

   ```
   $ mkdir cdk-hello-world && cd cdk-hello-world
   ```

2. `cdk init`명령을 사용하여 원하는 프로그래밍 언어로 새 프로젝트를 만드십시오.

TypeScript

```
$ cdk init --language typescript
```

AWS CDK 라이브러리 설치:

```
$ npm install aws-cdk-lib constructs
```

JavaScript

```
$ cdk init --language javascript
```

AWS CDK 라이브러리 설치:

```
$ npm install aws-cdk-lib constructs
```

Python

```
$ cdk init --language python
```

가상 환경 활성화:

```
$ source .venv/bin/activate
```

AWS CDK 라이브러리 및 프로젝트 종속성 설치:

```
(.venv)$ python3 -m pip install -r requirements.txt
```

Java

```
$ cdk init --language java
```

AWS CDK 라이브러리 및 프로젝트 종속성 설치:

```
$ mvn package
```

C#

```
$ cdk init --language csharp
```

AWS CDK 라이브러리 및 프로젝트 종속성 설치:

```
$ dotnet restore src
```

Go

```
$ cdk init --language go
```

프로젝트 종속성 설치:

```
$ go mod tidy
```

CLICDK는 다음과 같은 구조의 프로젝트를 생성합니다.

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
```

CDK는 단일 스택이 포함된 CDK 앱을 CLI 자동으로 생성합니다. CDK 앱 인스턴스는 클래스에서 생성됩니다. App 다음은 CDK 애플리케이션 파일의 일부입니다.

TypeScript

위치: `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

위치`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

위치`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

위치`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#

위치src/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

위치cdk-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
}
```

2단계: Lambda 함수 생성

CDK 프로젝트 내에서 새 파일이 포함된 lambda 디렉토리를 생성하십시오. hello.js 다음은 그 예제입니다.

TypeScript

프로젝트 루트에서 다음을 실행합니다.

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

이제 다음을 CDK 프로젝트에 추가해야 합니다.

```
cdk-hello-world
### lambda
    ### hello.js
```

JavaScript

프로젝트 루트에서 다음을 실행하세요.

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

이제 다음을 CDK 프로젝트에 추가해야 합니다.

```
cdk-hello-world
### lambda
    ### hello.js
```

Python

프로젝트 루트에서 다음을 실행하세요.

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

이제 다음을 CDK 프로젝트에 추가해야 합니다.

```
cdk-hello-world
### lambda
    ### hello.js
```

Java

프로젝트 루트에서 다음을 실행하세요.

```
$ mkdir -p src/main/resources/lambda
$ cd src/main/resources/lambda
$ touch hello.js
```

이제 다음을 CDK 프로젝트에 추가해야 합니다.

```
cdk-hello-world
### src
    ### main
        ###resources
            ###lambda
                ###hello.js
```

C#

프로젝트 루트에서 다음을 실행하세요.

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

이제 다음을 CDK 프로젝트에 추가해야 합니다.

```
cdk-hello-world
### lambda
    ### hello.js
```

Go

프로젝트 루트에서 다음을 실행하세요.

```
$ mkdir lambda && cd lambda
$ touch hello.js
```

이제 다음을 CDK 프로젝트에 추가해야 합니다.

```
cdk-hello-world
### lambda
    ### hello.js
```

> **Note**
>
> 이 자습서를 간단하게 하기 위해 모든 CDK 프로그래밍 JavaScript 언어에 Lambda 함수를 사용합니다.

새로 생성된 파일에 다음을 추가하여 Lambda 함수를 정의합니다.

```
exports.handler = async (event) => {
    return {
        statusCode: 200,
        headers: { "Content-Type": "text/plain" },
        body: JSON.stringify({ message: "Hello, World!" }),
    };
};
```

3단계: 구문 정의

이 단계에서는 L2 구조를 사용하여 Lambda 및 API Gateway 리소스를 AWS CDK 정의합니다.

CDK 스택을 정의하는 프로젝트 파일을 엽니다. 이 파일을 수정하여 구문을 정의합니다. 다음은 시작 스택 파일의 예시입니다.

TypeScript

위치`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

위치`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

위치cdk_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

위치src/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#

위치src/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

위치cdk-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
}
```

이 파일에서 AWS CDK 는 다음과 같은 작업을 수행합니다.

- CDK 스택 인스턴스는 클래스에서 인스턴스화됩니다. `Stack`
- `Constructs`기본 클래스를 가져와서 스택 인스턴스의 범위 또는 상위 클래스로 제공합니다.

Lambda 함수 리소스를 정의하십시오.

Lambda 함수 리소스를 정의하려면 구성 라이브러리에서 L2 구문을 가져와서 사용합니다`aws-lambda`. AWS

다음과 같이 스택 파일을 수정하십시오.

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**
>
> aws_lambda모듈을 임포트하는 lambda 이유는 내장 식별자가 있기 _lambda 때문입니다. Python

Java

```
// ...
// Import Lambda L2 construct
import software.amazon.awscdk.services.lambda.Code;
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.Runtime;

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

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

        // Define the Lambda function resource
        Function helloWorldFunction = Function.Builder.create(this,
 "HelloWorldFunction")
                .runtime(Runtime.NODEJS_20_X)  // Choose any supported Node.js
 runtime
                .code(Code.fromAsset("src/main/resources/lambda")) // Points to the
 lambda directory
                .handler("hello.handler")  // Points to the 'hello' file in the
 lambda directory
                .build();
```

```
    }
}
```

C#

```
// ...
// Import Lambda L2 construct
using Amazon.CDK.AWS.Lambda;

namespace CdkHelloWorld
{
    public class CdkHelloWorldStack : Stack
    {
        internal CdkHelloWorldStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            // Define the Lambda function resource
            var helloWorldFunction = new Function(this, "HelloWorldFunction", new
 FunctionProps
            {
                Runtime = Runtime.NODEJS_20_X, // Choose any supported Node.js
 runtime
                Code = Code.FromAsset("lambda"), // Points to the lambda directory
                Handler = "hello.handler" // Points to the 'hello' file in the
 lambda directory
            });
        }
    }
}
```

Go

```
package main

import (
    // ...
    // Import Lambda L2 construct
    "github.com/aws/aws-cdk-go/awscdk/v2/awslambda"
    // ...
)

// ...
```

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

    // Define the Lambda function resource
    helloWorldFunction := awslambda.NewFunction(stack,
 jsii.String("HelloWorldFunction"), &awslambda.FunctionProps{
        Runtime: awslambda.Runtime_NODEJS_20_X(), // Choose any supported Node.js
 runtime
        Code:    awslambda.Code_FromAsset(jsii.String("lambda")), // Points to the
 lambda directory
        Handler: jsii.String("hello"), // Points to the 'hello' file in the lambda
 directory
    })

    return stack
}

// ...
```

여기에서 Lambda 함수 리소스를 생성하고 다음 속성을 정의합니다.

- `runtime`— 함수가 실행되는 환경. 여기서는 Node.js 버전을 사용합니다20.x.
- `code`— 로컬 컴퓨터에 있는 함수 코드의 경로.
- `handler`— 함수 코드가 들어 있는 특정 파일의 이름.

API Gateway REST API 리소스 정의

API Gateway REST API 리소스를 정의하려면 AWS 구성 라이브러리에서 `aws-apigateway` L2 구문을 가져와서 사용합니다.

다음과 같이 스택 파일을 수정합니다.

TypeScript

```
// ...
//Import API Gateway L2 construct
```

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


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

    // ...

    // Define the API Gateway resource
    const api = new apigateway.LambdaRestApi(this, 'HelloWorldApi', {
      handler: helloWorldFunction,
      proxy: false,
    });

    // Define the '/hello' resource with a GET method
    const helloResource = api.root.addResource('hello');
    helloResource.addMethod('GET');
  }
}
```

JavaScript

```
// ...
// Import API Gateway L2 construct
const apigateway = require('aws-cdk-lib/aws-apigateway');


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

    // ...

    // Define the API Gateway resource
    const api = new apigateway.LambdaRestApi(this, 'HelloWorldApi', {
      handler: helloWorldFunction,
      proxy: false,
    });

    // Define the '/hello' resource with a GET method
    const helloResource = api.root.addResource('hello');
    helloResource.addMethod('GET');
```

```
  };
};

// ...
```

Python

```
from aws_cdk import (
    # ...
    # Import API Gateway L2 construct
    aws_apigateway as apigateway,
)
from constructs import Construct

class CdkHelloWorldStack(Stack):

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

        # ...

        # Define the API Gateway resource
        api = apigateway.LambdaRestApi(
            self,
            "HelloWorldApi",
            handler = hello_world_function,
            proxy = False,
        )

        # Define the '/hello' resource with a GET method
        hello_resource = api.root.add_resource("hello")
        hello_resource.add_method("GET")
```

Java

```
// ...
// Import API Gateway L2 construct
import software.amazon.awscdk.services.apigateway.LambdaRestApi;
import software.amazon.awscdk.services.apigateway.Resource;

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

```
    }

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

        // ...

        // Define the API Gateway resource
        LambdaRestApi api = LambdaRestApi.Builder.create(this, "HelloWorldApi")
                .handler(helloWorldFunction)
                .proxy(false) // Turn off default proxy integration
                .build();

        // Define the '/hello' resource and its GET method
        Resource helloResource = api.getRoot().addResource("hello");
        helloResource.addMethod("GET");
    }
}
```

C#

```
// ...
// Import API Gateway L2 construct
using Amazon.CDK.AWS.APIGateway;

namespace CdkHelloWorld
{
    public class CdkHelloWorldStack : Stack
    {
        internal CdkHelloWorldStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
           // ...

            // Define the API Gateway resource
            var api = new LambdaRestApi(this, "HelloWorldApi", new
 LambdaRestApiProps
            {
                Handler = helloWorldFunction,
                Proxy = false
            });
```

```
            // Add a '/hello' resource with a GET method
            var helloResource = api.Root.AddResource("hello");
            helloResource.AddMethod("GET");
        }
    }
}
```

Go

```
// ...

import (
    // ...
    // Import Api Gateway L2 construct
    "github.com/aws/aws-cdk-go/awscdk/v2/awsapigateway"
    // ...
)

// ...

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

    // Define the Lambda function resource
    // ...

    // Define the API Gatweay resource
    api := awsapigateway.NewLambdaRestApi(stack, jsii.String("HelloWorldApi"),
 &awsapigateway.LambdaRestApiProps{
        Handler: helloWorldFunction,
        Proxy: jsii.Bool(false),
    })

    // Add a '/hello' resource with a GET method
    helloResource := api.Root().AddResource(jsii.String("hello"))
    helloResource.AddMethod(jsii.String("GET"))

    return stack
```

```
}

// ...
```

여기에서 다음과 함께 API Gateway REST API 리소스를 생성합니다.

- Lambda 함수와 Lambda 함수를 통합하여 API가 함수를 호출할 수 있도록 합니다. REST API 여기에는 Lambda 권한 리소스 생성이 포함됩니다.
- API 엔드포인트의 루트에 `hello` 추가되는 이름이 지정된 새 리소스 또는 경로. 그러면 기본 엔드포인트가 `/hello` 추가되는 새 엔드포인트가 생성됩니다URL.
- `hello`리소스의 GET 메서드. GET 요청이 `/hello` 엔드포인트로 전송되면 Lambda 함수가 호출되고 해당 응답이 반환됩니다.

4단계: 배포를 위한 애플리케이션 준비

이 단계에서는 필요한 경우 AWS CDK CLI `cdk synth` 명령을 사용하여 빌드하고 기본 검증을 수행하여 배포할 애플리케이션을 준비합니다.

필요한 경우 애플리케이션을 빌드하세요.

TypeScript

프로젝트 루트에서 다음을 실행합니다.

```
$ npm run build
```

JavaScript

빌드는 필요하지 않습니다.

Python

건물은 필요하지 않습니다.

Java

프로젝트 루트에서 다음을 실행하세요.

```
$ mvn package
```

C#

프로젝트 루트에서 다음을 실행합니다.

```
$ dotnet build src
```

Go

프로젝트 루트에서 다음을 실행합니다.

```
$ go build
```

`cdk synth`를 실행하여 CDK 코드에서 AWS CloudFormation 템플릿을 합성합니다. L2 구조를 사용하면 Lambda 함수와 Lambda 함수 간의 상호 작용을 촉진하는 AWS CloudFormation 데 필요한 많은 구성 세부 정보가 에서 자동으로 REST API 프로비저닝됩니다. AWS CDK

프로젝트의 루트에서 다음을 실행합니다.

```
$ cdk synth
```

> **Note**
>
> 다음과 같은 오류가 발생하는 경우, `cdk-hello-world` 디렉터리에 있는지 확인하고 다시 시도하세요.
>
> ```
> --app is required either in command-line, in cdk.json or in ~/.cdk.json
> ```

성공하면 명령 프롬프트에 AWS CloudFormation 템플릿이 YAML 형식으로 출력됩니다. AWS CDK CLI JSON형식이 지정된 템플릿도 `cdk.out` 디렉토리에 저장됩니다.

다음은 AWS CloudFormation 템플릿의 출력 예제입니다.

AWS CloudFormation 템플릿

```
Resources:
  HelloWorldFunctionServiceRoleunique-identifier:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
```

```
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
        Version: "2012-10-17"
      ManagedPolicyArns:
        - Fn::Join:
            - ""
            - - "arn:"
              - Ref: AWS::Partition
              - :iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldFunction/ServiceRole/Resource
  HelloWorldFunctionunique-identifier:
    Type: AWS::Lambda::Function
    Properties:
      Code:
        S3Bucket:
          Fn::Sub: cdk-unique-identifier-assets-${AWS::AccountId}-${AWS::Region}
        S3Key: unique-identifier.zip
      Handler: hello.handler
      Role:
        Fn::GetAtt:
          - HelloWorldFunctionServiceRoleunique-identifier
          - Arn
      Runtime: nodejs20.x
    DependsOn:
      - HelloWorldFunctionServiceRoleunique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldFunction/Resource
      aws:asset:path: asset.unique-identifier
      aws:asset:is-bundled: false
      aws:asset:property: Code
  HelloWorldApiunique-identifier:
    Type: AWS::ApiGateway::RestApi
    Properties:
      Name: HelloWorldApi
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Resource
  HelloWorldApiDeploymentunique-identifier:
    Type: AWS::ApiGateway::Deployment
    Properties:
      Description: Automatically created by the RestApi construct
```

```
      RestApiId:
        Ref: HelloWorldApiunique-identifier
    DependsOn:
      - HelloWorldApihelloGETunique-identifier
      - HelloWorldApihellounique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Deployment/Resource
  HelloWorldApiDeploymentStageprod012345ABC:
    Type: AWS::ApiGateway::Stage
    Properties:
      DeploymentId:
        Ref: HelloWorldApiDeploymentunique-identifier
      RestApiId:
        Ref: HelloWorldApiunique-identifier
      StageName: prod
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/DeploymentStage.prod/Resource
  HelloWorldApihellounique-identifier:
    Type: AWS::ApiGateway::Resource
    Properties:
      ParentId:
        Fn::GetAtt:
          - HelloWorldApiunique-identifier
          - RootResourceId
      PathPart: hello
      RestApiId:
        Ref: HelloWorldApiunique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/Resource
  HelloWorldApihelloGETApiPermissionCdkHelloWorldStackHelloWorldApiunique-identifier:
    Type: AWS::Lambda::Permission
    Properties:
      Action: lambda:InvokeFunction
      FunctionName:
        Fn::GetAtt:
          - HelloWorldFunctionunique-identifier
          - Arn
      Principal: apigateway.amazonaws.com
      SourceArn:
        Fn::Join:
          - ""
          - - "arn:"
            - Ref: AWS::Partition
            - ":execute-api:"
```

```
            - Ref: AWS::Region
            - ":"
            - Ref: AWS::AccountId
            - ":"
            - Ref: HelloWorldApi9E278160
            - /
            - Ref: HelloWorldApiDeploymentStageprodunique-identifier
            - /GET/hello
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/GET/
ApiPermission.CdkHelloWorldStackHelloWorldApiunique-identifier.GET..hello
  HelloWorldApihelloGETApiPermissionTestCdkHelloWorldStackHelloWorldApiunique-
identifier:
    Type: AWS::Lambda::Permission
    Properties:
      Action: lambda:InvokeFunction
      FunctionName:
        Fn::GetAtt:
          - HelloWorldFunctionunique-identifier
          - Arn
      Principal: apigateway.amazonaws.com
      SourceArn:
        Fn::Join:
          - ""
          - - "arn:"
            - Ref: AWS::Partition
            - ":execute-api:"
            - Ref: AWS::Region
            - ":"
            - Ref: AWS::AccountId
            - ":"
            - Ref: HelloWorldApiunique-identifier
            - /test-invoke-stage/GET/hello
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/GET/
ApiPermission.Test.CdkHelloWorldStackHelloWorldApiunique-identifier.GET..hello
  HelloWorldApihelloGETunique-identifier:
    Type: AWS::ApiGateway::Method
    Properties:
      AuthorizationType: NONE
      HttpMethod: GET
      Integration:
        IntegrationHttpMethod: POST
        Type: AWS_PROXY
```

```
        Uri:
          Fn::Join:
            - ""
            - - "arn:"
              - Ref: AWS::Partition
              - ":apigateway:"
              - Ref: AWS::Region
              - :lambda:path/2015-03-31/functions/
              - Fn::GetAtt:
                  - HelloWorldFunctionunique-identifier
                  - Arn
              - /invocations
      ResourceId:
        Ref: HelloWorldApihellounique-identifier
      RestApiId:
        Ref: HelloWorldApiunique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/HelloWorldApi/Default/hello/GET/Resource
  CDKMetadata:
    Type: AWS::CDK::Metadata
    Properties:
      Analytics: v2:deflate64:unique-identifier
    Metadata:
      aws:cdk:path: CdkHelloWorldStack/CDKMetadata/Default
    Condition: CDKMetadataAvailable
Outputs:
  HelloWorldApiEndpointunique-identifier:
    Value:
      Fn::Join:
        - ""
        - - https://
          - Ref: HelloWorldApiunique-identifier
          - .execute-api.
          - Ref: AWS::Region
          - "."
          - Ref: AWS::URLSuffix
          - /
          - Ref: HelloWorldApiDeploymentStageprodunique-identifier
          - /
Conditions:
  CDKMetadataAvailable:
    Fn::Or:
      - Fn::Or:
          - Fn::Equals:
```

```
            - Ref: AWS::Region
            - af-south-1
        - Fn::Equals:
            - Ref: AWS::Region
            - ap-east-1
        - Fn::Equals:
            - Ref: AWS::Region
            - ap-northeast-1
        - Fn::Equals:
            - Ref: AWS::Region
            - ap-northeast-2
        - Fn::Equals:
            - Ref: AWS::Region
            - ap-south-1
        - Fn::Equals:
            - Ref: AWS::Region
            - ap-southeast-1
        - Fn::Equals:
            - Ref: AWS::Region
            - ap-southeast-2
        - Fn::Equals:
            - Ref: AWS::Region
            - ca-central-1
        - Fn::Equals:
            - Ref: AWS::Region
            - cn-north-1
        - Fn::Equals:
            - Ref: AWS::Region
            - cn-northwest-1
    - Fn::Or:
        - Fn::Equals:
            - Ref: AWS::Region
            - eu-central-1
        - Fn::Equals:
            - Ref: AWS::Region
            - eu-north-1
        - Fn::Equals:
            - Ref: AWS::Region
            - eu-south-1
        - Fn::Equals:
            - Ref: AWS::Region
            - eu-west-1
        - Fn::Equals:
            - Ref: AWS::Region
```

```
              - eu-west-2
          - Fn::Equals:
              - Ref: AWS::Region
              - eu-west-3
          - Fn::Equals:
              - Ref: AWS::Region
              - il-central-1
          - Fn::Equals:
              - Ref: AWS::Region
              - me-central-1
          - Fn::Equals:
              - Ref: AWS::Region
              - me-south-1
          - Fn::Equals:
              - Ref: AWS::Region
              - sa-east-1
      - Fn::Or:
          - Fn::Equals:
              - Ref: AWS::Region
              - us-east-1
          - Fn::Equals:
              - Ref: AWS::Region
              - us-east-2
          - Fn::Equals:
              - Ref: AWS::Region
              - us-west-1
          - Fn::Equals:
              - Ref: AWS::Region
              - us-west-2
Parameters:
  BootstrapVersion:
    Type: AWS::SSM::Parameter::Value<String>
    Default: /cdk-bootstrap/hnb659fds/version
    Description: Version of the CDK Bootstrap resources in this environment,
 automatically retrieved from SSM Parameter Store. [cdk:skip]
Rules:
  CheckBootstrapVersion:
    Assertions:
      - Assert:
          Fn::Not:
            - Fn::Contains:
                - - "1"
                  - "2"
                  - "3"
```

```
                  - "4"
                  - "5"
                - Ref: BootstrapVersion
        AssertDescription: CDK bootstrap stack version 6 required. Please run 'cdk
 bootstrap' with a recent version of the CDK CLI.
```

L2 구문을 사용하면 몇 가지 속성을 정의하여 리소스를 구성하고 도우미 메서드를 사용하여 리소스를 통합할 수 있습니다. 는 AWS CDK 애플리케이션을 프로비저닝하는 데 필요한 대부분의 AWS CloudFormation 리소스와 속성을 구성합니다.

5단계: 애플리케이션 배포

이 단계에서는 AWS CDK CLI `cdk deploy` 명령을 사용하여 애플리케이션을 배포합니다. 는 AWS CloudFormation 서비스와 함께 AWS CDK 작동하여 리소스를 프로비저닝합니다.

> **Important**
>
> 배포 전에 AWS 환경을 한 번 부트스트래핑해야 합니다. 지침은 환경 부트스트랩을 참조하십시오.

프로젝트 루트에서 다음을 실행하세요. 메시지가 표시되면 변경 내용을 확인합니다.

```
$ cdk deploy

#  Synthesis time: 2.44s

...

Do you wish to deploy these changes (y/n)? y
```

배포가 완료되면 엔드포인트 AWS CDK CLI URL이 출력됩니다. 다음 단계를 위해 이 URL을 복사하세요. 다음은 그 예제입니다.

```
...
#  HelloWorldStack

#  Deployment time: 45.37s

Outputs:
```

```
HelloWorldStack.HelloWorldApiEndpointunique-identifier = https://<api-id>.execute-
api.<region>.amazonaws.com/prod/
Stack ARN:
arn:aws:cloudformation:region:account-id:stack/HelloWorldStack/unique-identifier
...
```

6단계: 애플리케이션과 상호작용

이 단계에서는 API 엔드포인트에 대한 GET 요청을 시작하고 Lambda 함수 응답을 받습니다.

이전 단계에서 엔드포인트 URL을 찾아 경로를 추가합니다. /hello 그런 다음 브라우저 또는 명령 프롬프트를 사용하여 엔드포인트에 GET 요청을 보냅니다. 다음은 그 예제입니다.

```
$ curl https://<api-id>.execute-api.<region>.amazonaws.com/prod/hello
{"message":"Hello World!"}%
```

축하합니다. 를 사용하여 애플리케이션을 성공적으로 만들고 배포하고 상호 작용했습니다. AWS CDK

7단계: 애플리케이션 삭제

이 단계에서는 를 AWS CDK CLI 사용하여 에서 애플리케이션을 삭제합니다 AWS 클라우드.

애플리케이션을 삭제하려면 를 실행하십시오cdk destroy. 메시지가 표시되면 애플리케이션 삭제 요청을 확인합니다.

```
$ cdk destroy
Are you sure you want to delete: CdkHelloWorldStack (y/n)? y
CdkHelloWorldStack: destroying... [1/1]
...
 #  CdkHelloWorldStack: destroyed
```

문제 해결

오류: {“message”: “내부 서버 오류”}%

배포된 Lambda 함수를 호출할 때 이 오류가 발생합니다. 이 오류는 여러 가지 이유로 발생할 수 있습니다.

추가 문제 해결 방법

를 AWS CLI 사용하여 Lambda 함수를 호출할 수 있습니다.

1. 스택 파일을 수정하여 배포된 Lambda 함수 이름의 출력 값을 캡처합니다. 다음은 그 예제입니다.

```
...

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. 애플리케이션을 다시 배포하십시오. 배포된 Lambda 함수 이름의 값이 AWS CDK CLI 출력됩니다.

```
$ cdk deploy

#  Synthesis time: 0.29s
...
 #  CdkHelloWorldStack

#  Deployment time: 20.36s

Outputs:
...
CdkHelloWorldStack.HelloWorldFunctionName = CdkHelloWorldStack-
HelloWorldFunctionunique-identifier
...
```

3. AWS CLI 를 사용하여 에서 AWS 클라우드 Lambda 함수를 호출하고 텍스트 파일에 대한 응답을 출력합니다.

```
$ aws lambda invoke --function-name CdkHelloWorldStack-HelloWorldFunctionunique-
identifier output.txt
```

4. `output.txt`결과를 확인해 보십시오.

가능한 원인: API Gateway 리소스가 스택 파일에 잘못 정의되었습니다.

성공적인 Lambda 함수 응답이 `output.txt` 표시되는 경우 API Gateway REST API를 정의한 방식에 문제가 있을 수 있습니다. 는 AWS CLI 엔드포인트를 통하지 않고 Lambda를 직접 호출합니다. 코드가 이 자습서와 일치하는지 확인하십시오. 그런 다음 다시 배포하세요.

가능한 원인: Lambda 리소스가 스택 파일에 잘못 정의되었습니다.

오류가 `output.txt` 반환되는 경우 Lambda 함수를 정의한 방법이 문제일 수 있습니다. 코드가 이 자습서와 일치하는지 확인하십시오. 그런 다음 다시 배포하세요.

여러 스택이 있는 앱 만들기

여러 스택이 포함된 AWS Cloud Development Kit (AWS CDK) 애플리케이션을 만들 수 있습니다. AWS CDK 앱을 배포하면 각 스택이 자체 AWS CloudFormation 템플릿이 됩니다. AWS CDK CLI`cdk deploy`명령을 사용하여 각 스택을 개별적으로 합성하고 배포할 수도 있습니다.

이 자습서에서는 다음 내용을 다룹니다.

- 새 속성이나 인수를 허용하도록 `Stack` 클래스를 확장하는 방법
- 속성을 사용하여 스택에 포함된 리소스와 해당 구성을 결정하는 방법
- 이 클래스에서 여러 스택을 인스턴스화하는 방법

이 항목의 예제에서는 `encryptBucket` (Python:`encrypt_bucket`) 이라는 Boolean 속성을 사용합니다. Amazon S3 버킷을 암호화해야 하는지 여부를 나타냅니다. 그렇다면 스택은 AWS Key Management Service (AWS KMS) 에서 관리하는 키를 사용하여 암호화를 활성화합니다. 앱은 이 스택의 인스턴스 두 개를 생성합니다. 하나는 암호화가 적용되고 다른 하나는 암호화되지 않습니다.

주제

시작하기 전 준비 사항

먼저 Node.js 및 AWS CDK 명령줄 도구를 설치하세요 (아직 설치하지 않았다면). 세부 정보는 시작하기 AWS CDK를 참조하세요.

그런 다음 명령줄에 다음 명령을 입력하여 AWS CDK 프로젝트를 생성합니다.

TypeScript

```
mkdir multistack
cd multistack
cdk init --language=typescript
```

JavaScript

```
mkdir multistack
cd multistack
cdk init --language=javascript
```

Python

```
mkdir multistack
cd multistack
cdk init --language=python
source .venv/bin/activate
pip install -r requirements.txt
```

Java

```
mkdir multistack
cd multistack
cdk init --language=java
```

결과 Maven 프로젝트를 Java IDE로 가져올 수 있습니다.

C#

```
mkdir multistack
cd multistack
cdk init --language=csharp
```

Visual `src/Pipeline.sln` Studio에서 파일을 열 수 있습니다.

선택적 매개 변수 추가

`Stack`생성자의 `props` 인수가 인터페이스를 충족합니다. `StackProps` 이 예시에서는 Amazon S3 버킷을 암호화할지 여부를 알려주는 추가 속성을 스택에 허용하려고 합니다. 속성을 포함하는 인터페이스나 클래스를 만들어야 합니다. 이렇게 하면 컴파일러가 속성에 Boolean 값이 있는지 확인하고 IDE에서 해당 속성에 대한 자동 완성을 활성화할 수 있습니다.

따라서 표시된 소스 파일을 IDE 또는 편집기에서 열고 새 인터페이스, 클래스 또는 인수를 추가하십시오. 변경 후 코드는 다음과 같아야 합니다. 추가한 줄은 굵게 표시됩니다.

TypeScript

파일: `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

파일: `lib/multistack-stack.js`

JavaScript 인터페이스 기능이 없으므로 코드를 추가할 필요가 없습니다.

```
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

파일: multistack/multistack_stack.py

Python에는 인터페이스 기능이 없으므로 키워드 인수를 추가하여 새 속성을 받아들이도록 스택을 확장합니다.

```
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

파일: src/main/java/com/myorg/MultistackStack.java

Java에서 props 유형을 확장하는 것은 생각보다 더 복잡합니다. 대신 선택적 Boolean 파라미터를 받아들이도록 스택의 생성자를 작성하세요. props는 선택적 인수이므로 건너뛸 수 있는 추가 생성자를 작성해 보겠습니다. 기본값은 입니다. 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#

파일: 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
        }
```

```
    }
}
```

새 속성은 선택 사항입니다. `encryptBucket`(Python:`encrypt_bucket`) 가 없는 경우 해당 값은 또는 로컬에 해당하는 값입니다`undefined`. 버킷은 기본적으로 암호화되지 않습니다.

스택 클래스를 정의합니다.

이제 새 속성을 사용하여 스택 클래스를 정의해 보겠습니다. 코드를 다음과 같이 만들어 보세요. 추가 또는 변경해야 하는 코드는 굵게 표시됩니다.

TypeScript

파일: `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

파일: 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

파일: 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

파일: 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#

파일: 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
                });
            }
        }
    }
}
```

스택 인스턴스 2개 생성

이제 두 개의 개별 스택을 인스턴스화하는 코드를 추가해 보겠습니다. 이전과 마찬가지로 굵게 표시된 코드 줄을 추가해야 합니다. 기존 `MultistackStack` 정의를 삭제합니다.

TypeScript

파일: `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

파일: `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

파일: `./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

파일: `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#

파일: 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();
        }
    }
}
```

이 코드는 `MultistackStack` 클래스의 new `encryptBucket` (Python:`encrypt_bucket`) 속성을 사용하여 다음을 인스턴스화합니다.

- 리전에 암호화된 Amazon S3 버킷이 `us-east-1` AWS 있는 스택 1개
- 리전에 암호화되지 않은 Amazon S3 버킷이 `us-west-1` AWS 있는 스택 1개

스택을 합성하고 배포합니다.

이제 앱에서 스택을 배포할 수 있습니다. 먼저 `MyEastCdkStack` —스택에 대한 AWS CloudFormation 템플릿을 합성합니다. `us-east-1` 암호화된 S3 버킷이 있는 스택입니다.

```
$ cdk synth MyEastCdkStack
```

이 스택을 AWS 계정에 배포하려면 다음 명령 중 하나를 실행합니다. 첫 번째 명령은 기본 AWS 프로필을 사용하여 스택을 배포하는 데 필요한 자격 증명을 얻습니다. 두 번째 방법은 지정한 프로필을 사용합니다. *`PROFILE_NAME#`* 경우 지역에 배포하기 위한 적절한 자격 증명이 포함된 AWS CLI 프로필 이름을 대체하십시오. `us-east-1` AWS

```
cdk deploy MyEastCdkStack
```

```
cdk deploy MyEastCdkStack --profile=PROFILE_NAME
```

정리

배포한 리소스에 대한 요금이 청구되지 않도록 하려면 다음 명령을 사용하여 스택을 삭제하세요.

```
cdk destroy MyEastCdkStack
```

스택 버킷에 저장된 항목이 있는 경우 삭제 작업이 실패합니다. 이 주제의 지침만 따랐다면 그럴 수 없습니다. 하지만 버킷에 무언가를 넣었다면 스택을 파괴하기 전에 버킷 콘텐츠를 삭제해야 합니다. (버킷 자체는 삭제하지 마세요.) AWS Management Console 또는 를 AWS CLI 사용하여 버킷 콘텐츠를 삭제합니다.

예제

이 항목에는 다음 예제가 포함되어 있습니다.

- 서버리스 헬로 월드 애플리케이션 만들기Lambda, API Gateway 및 Amazon S3를 사용하여 서버리스 애플리케이션을 생성합니다.
- 를 사용하여 AWS Fargate 서비스 생성 AWS CDK이미지를 기반으로 Amazon ECS Fargate 서비스를 생성합니다. DockerHub

를 사용하여 AWS Fargate 서비스 생성 AWS CDK

이 예제는 Amazon ECR의 이미지에서 인터넷 연결 애플리케이션 로드 밸런서가 앞에 있는 Amazon Elastic Container Service (Amazon ECS) 클러스터에서 실행되는 AWS Fargate 서비스를 생성하는 방법을 안내합니다.

Amazon ECS는 클러스터에서 Docker 컨테이너를 손쉽게 실행, 중지 및 관리할 수 있게 해주는 컨테이너 관리 서비스로서 확장성과 속도가 뛰어납니다. Fargate 시작 유형을 사용하여 서비스 또는 작업을 시작하여 Amazon ECS에서 관리하는 서버리스 인프라에서 클러스터를 호스팅할 수 있습니다. 제어를 강화하려면 Amazon EC2 시작 유형을 사용하여 관리하는 Amazon Elastic Compute Cloud (Amazon EC2) 인스턴스 클러스터에서 작업을 호스팅할 수 있습니다.

이 자습서에서는 Fargate 시작 유형을 사용하여 일부 서비스를 시작하는 방법을 보여줍니다. 를 사용하여 Fargate 서비스를 만들어 본 적이 있다면 해당 작업을 수행하기 위해 따라야 할 단계가 많다는 것을 알고 계실 것입니다. AWS Management Console AWS 에는 Fargate 서비스를 만드는 과정을 안내하는 다음과 같은 몇 가지 자습서 및 설명서 항목이 있습니다.

- Docker 컨테이너를 배포하는 방법 - AWS
- Amazon ECS를 사용한 설정
- Fargate를 사용하여 아마존 ECS 시작하기

이 예제는 유사한 Fargate 서비스를 코드로 생성합니다. AWS CDK

이 자습서에서 사용하는 Amazon ECS 구조는 다음과 같은 이점을 제공하여 AWS 서비스를 사용하는 데 도움이 됩니다.

- 로드 밸런서를 자동으로 구성합니다.

- 로드 밸런서를 위한 보안 그룹을 자동으로 엽니다. 이렇게 하면 보안 그룹을 명시적으로 생성하지 않고도 로드 밸런서가 인스턴스와 통신할 수 있습니다.
- 대상 그룹에 연결된 서비스와 로드 밸런서 간의 종속성을 자동으로 정렬합니다. 대상 그룹에서는 인스턴스가 생성되기 전에 리스너를 생성하는 올바른 순서를 AWS CDK 적용합니다.
- 자동 스케일링 그룹의 사용자 데이터를 자동으로 구성합니다. 이렇게 하면 클러스터를 AMI에 연결할 수 있는 올바른 구성이 생성됩니다.
- 파라미터 조합을 조기에 검증합니다. 이렇게 하면 AWS CloudFormation 문제가 조기에 노출되므로 배포 시간이 절약됩니다. 예를 들어 작업에 따라 메모리 설정을 잘못 구성하기 쉽습니다. 이전에는 앱을 배포하기 전까지는 오류가 발생하지 않았습니다. 하지만 이제는 잘못된 구성을 감지하여 앱을 AWS CDK 합성할 때 오류를 발생시킬 수 있습니다.
- Amazon ECR의 이미지를 사용하는 경우 Amazon Elastic Container 레지스트리 (Amazon ECR) 에 대한 권한을 자동으로 추가합니다.
- 자동으로 크기를 조정합니다. Amazon EC2 클러스터를 사용할 때 인스턴스를 자동 확장할 수 있는 방법을 AWS CDK 제공합니다. 이는 Fargate 클러스터의 인스턴스를 사용할 때 자동으로 발생합니다.

  또한 는 자동 크기 조정이 인스턴스를 종료하려고 하는데 해당 인스턴스에서 작업이 실행 중이거나 예약 중인 경우 인스턴스가 삭제되지 AWS CDK 않도록 합니다.

  이전에는 이 기능을 사용하려면 Lambda 함수를 생성해야 했습니다.
- 자산 지원을 제공하므로 머신의 소스를 Amazon ECS로 한 번에 배포할 수 있습니다. 이전에는 애플리케이션 소스를 사용하려면 Amazon ECR에 업로드하고 Docker 이미지를 생성하는 등 몇 가지 수동 단계를 수행해야 했습니다.

자세한 내용은 ECS를 참조하십시오.

> **Important**
>
> 앞으로 사용할 `ApplicationLoadBalancedFargateService` 구성에는 수많은 AWS 구성 요소가 포함되며, 그중 일부는 사용하지 않더라도 AWS 계정에 프로비저닝된 상태로 두면 비용이 많이 듭니다. 이 예제를 완료한 후에는 반드시 cleanup () cdk destroy 하세요.

디렉터리 생성 및 초기화 AWS CDK

먼저 AWS CDK 코드를 저장할 디렉터리를 만든 다음 해당 디렉터리에 AWS CDK 앱을 만들어 보겠습니다.

TypeScript

```
mkdir MyEcsConstruct
cd MyEcsConstruct
cdk init --language typescript
```

JavaScript

```
mkdir MyEcsConstruct
cd MyEcsConstruct
cdk init --language javascript
```

Python

```
mkdir MyEcsConstruct
cd MyEcsConstruct
cdk init --language python
source .venv/bin/activate
pip install -r requirements.txt
```

Java

```
mkdir MyEcsConstruct
cd MyEcsConstruct
cdk init --language java
```

이제 Maven 프로젝트를 IDE로 가져올 수 있습니다.

C#

```
mkdir MyEcsConstruct
cd MyEcsConstruct
cdk init --language csharp
```

이제 Visual `src/MyEcsConstruct.sln` Studio에서 열 수 있습니다.

앱을 실행하고 빈 스택이 생성되는지 확인합니다.

```
cdk synth
```

Fargate 서비스 만들기

Amazon ECS로 컨테이너 작업을 실행하는 두 가지 방법이 있습니다.

- Amazon ECS가 컨테이너가 실행되는 물리적 머신을 대신 관리하는 `Fargate` 시작 유형을 사용하십시오.
- 자동 크기 조정 지정과 같은 관리를 수행하는 `EC2` 시작 유형을 사용하십시오.

이 예제에서는 인터넷에 연결된 Application Load Balancer가 앞에 있는 ECS 클러스터에서 실행되는 Fargate 서비스를 생성해 보겠습니다.

다음 AWS Construct Library 모듈 가져오기를 표시된 파일에 추가합니다.

TypeScript

파일: `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

파일: `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

파일: `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

파일: 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#

파일: src/MyEcsConstruct/MyEcsConstructStack.cs

```
using Amazon.CDK.AWS.EC2;
using Amazon.CDK.AWS.ECS;
using Amazon.CDK.AWS.ECS.Patterns;
```

생성자 끝에 있는 주석을 다음 코드로 바꿉니다.

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
```

```
                }
            );
```

저장하고 스택이 실행되고 생성되는지 확인하세요.

```
cdk synth
```

스택은 수백 줄에 달하므로 여기서는 보여주지 않겠습니다. 스택에는 기본 인스턴스 1개, 3개의 가용 영역에 대한 프라이빗 서브넷 및 퍼블릭 서브넷, 보안 그룹이 포함되어야 합니다.

스택을 배포합니다.

```
cdk deploy
```

AWS CloudFormation 앱을 배포할 때 거치는 수십 단계에 대한 정보를 표시합니다.

따라서 Docker 이미지를 실행하기 위한 Fargate 기반 Amazon ECS 서비스를 만드는 것이 얼마나 쉬운지 알 수 있습니다.

정리

예상치 못한 AWS 요금이 부과되지 않도록 하려면 이 연습을 완료한 후 AWS CDK 스택을 폐기하십시오.

```
cdk destroy
```

AWS CDK 예시

선호하는 지원 프로그래밍 언어의 AWS CDK 스택 및 앱 예제를 더 보려면 AWS CDK 예제 저장소를 참조하십시오. GitHub

AWS CDK 도구

이 섹션에는 아래 나열된 AWS CDK 도구에 대한 정보가 포함되어 있습니다.

주제

AWS CDK 툴킷 (**cdk**명령)

CLI cdk 명령어인 AWS CDK 툴킷은 앱과 상호작용하기 위한 기본 도구입니다. AWS CDK 앱을 실행하고, 정의한 애플리케이션 모델을 조사하고, 에서 생성된 템플릿을 생성 및 배포합니다. AWS CloudFormation AWS CDK또한 프로젝트를 만들고 작업하는 데 유용한 다른 기능도 제공합니다. AWS CDK 이 항목에는 CDK 툴킷의 일반적인 사용 사례에 대한 정보가 포함되어 있습니다.

AWS CDK 툴킷은 Node Package Manager와 함께 설치됩니다. 대부분의 경우 전역으로 설치하는 것이 좋습니다.

```
npm install -g aws-cdk              # install latest version
npm install -g aws-cdk@X.YY.Z       # install specific version
```

> **Tip**
>
> 정기적으로 여러 버전의 툴킷을 사용하는 경우 개별 CDK 프로젝트에 일치하는 버전의 AWS CDK 툴킷을 설치하는 것을 고려해 보십시오. AWS CDK이 작업을 수행하려면 `-g` 명령에서 생략하십시오. `npm install` 그런 다음 `npx aws-cdk` 를 사용하여 호출하십시오. 이렇게 하면 로컬 버전이 있으면 해당 버전이 실행되고, 없으면 글로벌 버전으로 대체됩니다.

툴킷 명령

모든 CDK 툴킷 명령은 로 cdk 시작하고 그 뒤에 하위 명령 (`list`, `synthesize`deploy, 등) 이 옵니다. 일부 하위 명령에는 동일한 더 짧은 버전 (`ls`synth, 등) 이 있습니다. 옵션과 인수는 어떤 순서로든 하위 명령 뒤에 옵니다. 사용 가능한 명령이 여기에 요약되어 있습니다.

Command	함수
`cdk list` (`ls`)	앱의 스택을 나열합니다.
`cdk synthesize` (`synth`)	하나 이상의 지정된 스택에 대한 CloudFormation 템플릿을 합성하고 인쇄합니다.
`cdk bootstrap`	CDK 툴킷 스테이징 스택을 배포합니다. 참조 the section called “부트스트래핑”
`cdk deploy`	하나 이상의 지정된 스택을 배포합니다.
`cdk destroy`	하나 이상의 지정된 스택을 파괴합니다.
`cdk diff`	지정된 스택과 해당 종속성을 배포된 스택 또는 로컬 템플릿과 비교합니다. CloudFormation
`cdk import`	CloudFormation 리소스 가져오기를 사용하여 기존 리소스를 CDK로 관리되는 스택으로 가져옵니다.
`cdk metadata`	지정된 스택에 대한 메타데이터를 표시합니다.
`cdk init`	지정된 템플릿을 사용하여 현재 디렉터리에 새 CDK 프로젝트를 만듭니다.
`cdk context`	캐시된 컨텍스트 값을 관리합니다.
`cdk docs` (`doc`)	브라우저에서 CDK API 레퍼런스를 엽니다.
`cdk doctor`	CDK 프로젝트에 잠재적 문제가 있는지 확인합니다.

각 명령에 사용할 수 있는 옵션은 을 참조하십시오the section called “내장된 도움말”.

옵션 및 해당 값 지정하기

명령줄 옵션은 두 개의 하이픈 () -- 으로 시작합니다. 자주 사용되는 일부 옵션에는 단일 하이픈으로 시작하는 단일 문자 동의어가 있습니다 (예: 동의어가 있음). --app -a 툴킷 명령의 옵션 순서는 중요하지 않습니다. AWS CDK

모든 옵션은 값을 허용하며, 이 값은 옵션 이름 뒤에 와야 합니다. 값은 공백이나 등호로 이름과 구분할 수 있습니다. = 다음 두 옵션은 동일합니다.

```
--toolkit-stack-name MyBootstrapStack
--toolkit-stack-name=MyBootstrapStack
```

일부 옵션은 플래그 (불리언) 입니다. true또는 값을 지정할 수 있습니다false. 값을 제공하지 않으면 값이 로 간주됩니다true. no-false암시하기 위해 옵션 이름 앞에 접두사를 붙일 수도 있습니다.

```
# sets staging flag to true
--staging
--staging=true
--staging true

# sets staging flag to false
--no-staging
--staging=false
--staging false
```

, --context --parameters --plugin --tags--trust, 및 같은 몇 가지 옵션을 여러 번 지정하여 여러 값을 지정할 수 있습니다. 이러한 항목은 CDK 툴킷 [array] 도움말에 입력이 있는 것으로 기록되어 있습니다. 예:

```
cdk bootstrap --tags costCenter=0123 --tags responsibleParty=jdoe
```

내장된 도움말

AWS CDK 툴킷에는 통합 도움말이 있습니다. 다음을 실행하여 유틸리티에 대한 일반 도움말과 제공된 하위 명령 목록을 볼 수 있습니다.

```
cdk --help
```

예를 들어 deploy 특정 하위 명령에 대한 도움말을 보려면 플래그 앞에 해당 --help 하위 명령을 지정하십시오.

```
cdk deploy --help
```

AWS CDK 툴킷 버전을 표시하는 `cdk version` 데 문제가 있습니다. 지원을 요청할 때 이 정보를 제공하십시오.

버전 보고

AWS CDK 가 사용되는 방식을 파악하기 위해 AWS CDK 애플리케이션에서 사용하는 구조를 다음과 같이 `AWS::CDK::Metadata` 식별된 리소스를 사용하여 수집하고 보고합니다. 이 리소스는 AWS CloudFormation 템플릿에 추가되며 쉽게 검토할 수 있습니다. 알려진 보안 또는 안정성 문제가 있는 구조를 사용하여 스택을 식별하는 데에도 이 정보를 사용할 수 있습니다. AWS 또한 사용자에게 중요한 정보를 문의하는 데에도 사용할 수 있습니다.

> **Note**
>
> 버전 1.93.0 이전에는 스택에서 사용된 구문 대신 합성 중에 로드된 모듈의 이름과 버전을 AWS CDK 보고했습니다.

기본적으로 는 스택에서 사용되는 다음 NPM 모듈의 구문 사용을 AWS CDK 보고합니다.

- AWS CDK 코어 모듈
- AWS 라이브러리 모듈 생성
- AWS 솔루션 구성 모듈
- AWS 렌더 팜 배포 키트 모듈

`AWS::CDK::Metadata`리소스는 다음과 같습니다.

```
CDKMetadata:
  Type: "AWS::CDK::Metadata"
  Properties:
    Analytics:
 "v2:deflate64:H4sIAND9SGAAAzXKSw5AMBAA0L1b2PdzBYnEAdio3RglglY60zQi7u6TWL/
XKmNUlxeQSOKwaPTBqrNhwEWU3hGHiCzK0dWWfAxoL/Fd8mvk+QkS/0X6BdjnCdgmOOQKWz
+AqqLDt2Y3YMnLYWwAAAA="
```

`Analytics`속성은 gzip으로 압축되고 base64로 인코딩되고 접두사로 인코딩된 스택의 구문 목록입니다.

버전 보고를 거부하려면 다음 방법 중 하나를 사용하십시오.

- cdk명령을 --no-version-reporting 인수와 함께 사용하면 단일 명령을 옵트아웃할 수 있습니다.

```
cdk --no-version-reporting synth
```

  AWS CDK 툴킷은 배포하기 전에 새 템플릿을 합성하므로 명령에도 `--no-version-reporting` 추가해야 한다는 점을 기억하십시오. `cdk deploy`

- 또는 versionReporting 에서 false로 설정하십시오. `./cdk.json` `~/.cdk.json` 개별 명령을 `--version-reporting` 지정하여 옵트인하지 않으면 옵트아웃됩니다.

```
{
  "app": "...",
  "versionReporting": false
}
```

다음을 통한 인증 AWS

AWS 리소스에 대한 프로그래밍 방식 액세스를 구성할 수 있는 방법은 환경 및 사용 가능한 AWS 액세스에 따라 다릅니다.

인증 방법을 선택하고 CDK 툴킷에 맞게 구성하려면 AWS SDK 및 도구 참조 안내서의 인증 및 액세스를 참조하십시오.

고용주로부터 인증 방법을 제공하지 않고 현지에서 개발하는 신규 사용자에게 권장되는 접근 방식은 설정하는 것입니다. AWS IAM Identity Center이 방법에는 간편한 구성을 AWS CLI 위한 설치와 AWS 액세스 포털에 정기적으로 로그인하기 위한 설치가 포함됩니다. 이 방법을 선택하는 경우AWS SDK 및 도구 참조 안내서의 IAM Identity Center authentication 절차를 완료한 후 환경에 다음 요소가 포함되어야 합니다.

- 는 AWS CLI애플리케이션을 실행하기 전에 AWS 액세스 포털 세션을 시작하는 데 사용합니다.
- 에서 참조할 수 있는 구성 값 집합이 포함된 `[default]` 프로필이 있는 공유 AWSconfig 파일입니다. AWS CDK이 파일의 위치를 찾으려면 AWS SDK 및 도구 참조 가이드에서 공유 파일의 위치를 참조하세요.
- 공유 `config` 파일은 `region` 설정을 지정합니다. 이렇게 하면 툴킷이 AWS 리전 요청에 사용하는 기본 AWS CDK 및 CDK가 설정됩니다. AWS

- CDK 툴킷은 요청을 보내기 전에 프로필의 SSO 토큰 공급자 구성을 사용하여 자격 증명을 획득합니다. AWS IAM Identity Center 권한 집합에 연결된 IAM 역할인 `sso_role_name` 값은 애플리케이션에서 사용자에 대한 액세스를 허용해야 합니다. AWS 서비스

  다음 샘플 `config` 파일은 SSO 토큰 공급자 구성으로 설정된 기본 프로필을 보여줍니다. 프로필의 `sso_session` 설정은 이름이 지정된 sso-session section을 참조합니다. `sso-session`섹션에는 AWS 액세스 포털 세션을 시작하기 위한 설정이 포함되어 있습니다.

```
[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
```

AWS 액세스 포털 세션 시작

AWS 서비스액세스하기 전에 CDK 툴킷에서 IAM Identity Center 인증을 사용하여 자격 증명을 확인할 수 있는 활성 AWS 액세스 포털 세션이 필요합니다. 구성된 세션 길이에 따라 액세스는 결국 만료되며 CDK 툴킷에서 인증 오류가 발생합니다. 에서 다음 명령을 AWS CLI 실행하여 액세스 포털에 로그인합니다. AWS

```
aws sso login
```

SSO 토큰 공급자 구성에서 기본 프로필 대신 명명된 프로필을 사용하는 경우 명령은 입니다`aws sso login --profile NAME`. 또한 --profile 옵션이나 `AWS_PROFILE` 환경 변수를 사용하여 cdk 명령을 실행할 때 이 프로필을 지정하십시오.

이미 활성 세션이 있는지 테스트하려면 다음 AWS CLI 명령을 실행합니다.

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

이 명령에 대한 응답은 공유 `config` 파일에 구성된 IAM Identity Center 계정 및 권한 집합을 보고해야 합니다.

> **Note**
>
> 이미 활성 AWS 액세스 포털 세션을 실행하고 `aws sso login` 있는 경우에는 자격 증명을 제공할 필요가 없습니다.
> 로그인 과정에서 데이터에 AWS CLI 대한 접근을 허용하라는 메시지가 표시될 수 있습니다.
> AWS CLI 는 Python용 SDK를 기반으로 구축되었으므로 권한 메시지에는 다양한 `botocore` 이름이 포함될 수 있습니다.

지역 및 기타 구성 지정

CDK 툴킷은 배포하려는 AWS 지역과 인증 방법을 알아야 합니다. AWS이는 배포 작업과 합성 중에 컨텍스트 값을 검색하는 데 필요합니다. 계정과 지역이 함께 환경을 구성합니다.

지역은 환경 변수를 사용하거나 구성 파일에서 지정할 수 있습니다. 이는 및 다양한 AWS SDK와 같은 다른 AWS 도구에서 사용하는 것과 동일한 변수 AWS CLI 및 파일입니다. CDK 툴킷은 다음과 같은 순서로 이 정보를 찾습니다.

- `AWS_DEFAULT_REGION` 환경 변수.
- 표준 AWS `config` 파일에 정의되고 명령의 `--profile` 옵션을 사용하여 지정된 이름이 지정된 프로파일입니다. `cdk`
- 표준 AWS `config` 파일의 `[default]` 섹션.

`[default]`섹션에서 AWS 인증 및 지역을 지정하는 것 외에도 하나 이상의 `[profile` *`NAME`*`]` 섹션을 추가할 수도 있습니다. 여기서 *`NAME#`* 프로필 이름입니다. 명명된 프로필에 대한 자세한 내용은 AWS SDK 및 도구 참조 안내서의 공유 구성 및 자격 증명 파일을 참조하십시오.

표준 AWS `config` 파일은 `~/.aws/config` (맥OS/리눅스) 또는 `%USERPROFILE%\.aws\config` (윈도우) 에 있습니다. 자세한 내용 및 대체 위치는 AWS SDK 및 도구 참조 안내서의 공유 구성 및 자격 증명 파일 위치를 참조하십시오.

스택의 `env` 속성을 사용하여 AWS CDK 앱에 지정하는 환경은 합성 중에 사용됩니다. 환경별 AWS CloudFormation 템플릿을 생성하는 데 사용되며, 배포 중에는 위의 방법 중 하나로 지정된 계정 또는 지역을 재정의합니다. 자세한 정보는 the section called “환경”을 참조하세요.

> **Note**
>
> 는 를 비롯한 다른 AWS 도구 및 SDK와 동일한 소스 파일의 자격 증명을 AWS CDK 사용합니다. AWS Command Line Interface 하지만 이 도구들과는 약간 AWS CDK 다르게 동작할 수 있습니다. AWS SDK for JavaScript 언더-더-후드를 사용합니다. 의 자격 증명 설정에 대한 자세한 내용은 자격 증명 설정을 참조하십시오. AWS SDK for JavaScript

선택적으로 `--role-arn` (또는`-r`) 옵션을 사용하여 배포에 사용해야 하는 IAM 역할의 ARN을 지정할 수 있습니다. 사용 중인 AWS 계정이 이 역할을 맡을 수 있어야 합니다.

앱 명령 지정

CDK 툴킷의 많은 기능을 사용하려면 하나 이상의 AWS CloudFormation 템플릿을 합성해야 하며, 이를 위해서는 애플리케이션을 실행해야 합니다. 다양한 언어로 작성된 프로그램을 AWS CDK 지원합니다. 따라서 구성 옵션을 사용하여 앱을 실행하는 데 필요한 정확한 명령을 지정합니다. 이 옵션은 두 가지 방법으로 지정할 수 있습니다.

먼저, 가장 일반적으로 파일 내 `app` 키를 사용하여 지정할 수 `cdk.json` 있습니다. 이것은 AWS CDK 프로젝트의 메인 디렉터리에 있습니다. CDK 툴킷은 새 프로젝트를 만들 때 적절한 명령을 제공합니다. `cdk init` 예를 들어, 다음은 새 TypeScript 프로젝트에서 `cdk.json` 가져온 것입니다.

```
{
  "app": "npx ts-node bin/hello-cdk.ts"
}
```

CDK 툴킷은 앱을 실행하려고 할 때 현재 작업 디렉터리를 찾습니다`cdk.json`. 이 때문에 프로젝트의 메인 디렉터리에 CDK 툴킷 명령을 실행하기 위한 셸을 열어 둘 수 있습니다.

또한 CDK 툴킷은 앱 키를 찾을 `~/.cdk.json` 수 없는 경우 홈 디렉터리에서 앱 키를 찾습니다. `./cdk.json` 보통 같은 언어의 CDK 코드를 사용하는 경우 여기에 app 명령을 추가하면 유용할 수 있습니다.

다른 디렉터리에 있거나 에 있는 명령이 아닌 다른 명령을 사용하여 앱을 실행하려면 `--app` (또는`-a`) 옵션을 사용하여 지정하십시오. `cdk.json`

```
cdk --app "npx ts-node bin/hello-cdk.ts" ls
```

배포할 때 합성된 클라우드 어셈블리가 포함된 디렉터리를 값 (예:) 으로 cdk.out 지정할 수도 있습니다. --app 지정된 스택은 이 디렉터리에서 배포되며 앱은 합성되지 않습니다.

스택 지정

많은 CDK 툴킷 명령 (예:cdk deploy) 은 앱에 정의된 스택에서 작동합니다. 앱에 스택이 하나뿐인 경우 스택을 명시적으로 지정하지 않으면 CDK 툴킷은 스택을 의미하는 것으로 간주합니다.

그렇지 않으면 작업할 스택을 하나 또는 여러 개 지정해야 합니다. 명령줄에서 ID별로 원하는 스택을 개별적으로 지정하여 이 작업을 수행할 수 있습니다. 스택을 인스턴스화할 때 ID는 두 번째 인수로 지정된 값이라는 점을 기억하십시오.

```
cdk synth PipelineStack LambdaStack
```

와일드카드를 사용하여 패턴과 일치하는 ID를 지정할 수도 있습니다.

- ?모든 단일 문자와 일치합니다.
- *원하는 수의 문자와 일치합니다 (*단독으로 모든 스택과 일치함).
- **계층 구조의 모든 항목과 일치합니다.

--all옵션을 사용하여 모든 스택을 지정할 수도 있습니다.

앱에서 CDK Pipelines를 사용하는 경우 CDK 툴킷은 스택과 스테이지를 계층 구조로 이해합니다. 또한 --all 옵션과 와일드카드는 최상위 스택에만 일치합니다*. 모든 스택을 일치시키려면 를 사용하십시오. ** 또한 특정 계층 아래의 모든 스택을 나타내는 ** 데 사용합니다.

와일드카드를 사용할 때는 패턴을 따옴표로 묶거나 를 사용하여 와일드카드를 이스케이프하세요. \ 그렇지 않으면 셸에서 패턴을 현재 디렉터리의 파일 이름으로 확장하려고 할 수 있습니다. 기껏해야 이렇게 하면 예상한 대로 되지 않을 것입니다. 최악의 경우 의도하지 않은 스택을 배포할 수도 있습니다. 와일드카드를 확장하지 cmd.exe 않기 때문에 Windows에서 반드시 필요한 것은 아니지만 그래도 좋은 방법입니다.

```
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**
>
> 스택을 지정하는 순서가 반드시 스택이 처리되는 순서와 같을 필요는 없습니다. AWS CDK 툴킷은 스택 처리 순서를 결정할 때 스택 간의 종속성을 고려합니다. 예를 들어 한 스택이 다른 스택에서 생성된 값 (예: 두 번째 스택에 정의된 리소스의 ARN) 을 사용한다고 가정해 보겠습니다. 이 경우 이러한 종속성 때문에 두 번째 스택이 첫 번째 스택보다 먼저 합성됩니다. 스택의 메서드를 사용하여 스택 간에 종속성을 수동으로 추가할 수 있습니다. addDependency()

환경 부트스트래핑 AWS

CDK로 스택을 배포하려면 특별한 전용 AWS CDK 리소스를 프로비저닝해야 합니다. `cdk bootstrap`명령을 실행하면 필요한 리소스가 자동으로 생성됩니다. 이러한 전용 리소스가 필요한 스택을 배포하는 경우에만 부트스트랩하면 됩니다. 세부 정보는 the section called “부트스트래핑”를 참조하세요.

```
cdk bootstrap
```

여기에 표시된 것처럼 인수 없이 `cdk bootstrap` 실행하면 명령이 현재 앱을 합성하고 해당 스택이 배포될 환경을 부트스트랩합니다. 앱에 환경을 명시적으로 지정하지 않는 환경에 구애받지 않는 스택이 포함된 경우 기본 계정 및 지역이 부트스트랩되거나 를 사용하여 지정된 환경이 부트스트랩됩니다. `--profile`

앱 외부에서는 부트스트랩할 환경을 명시적으로 지정해야 합니다. 앱 또는 로컬 프로필에 지정되지 않은 환경을 부트스트랩하기 위해 그렇게 할 수도 있습니다. AWS 지정된 계정 및 지역에 대한 자격 증명을 구성 (예: 에서`~/.aws/credentials`) 해야 합니다. 필수 자격 증명이 포함된 프로필을 지정할 수 있습니다.

```
cdk bootstrap ACCOUNT-NUMBER/REGION # e.g.
cdk bootstrap 1111111111/us-east-1
cdk bootstrap --profile test 1111111111/us-east-1
```

> **Important**
>
> 이러한 스택을 배포하는 각 환경 (계정/지역 조합) 은 별도로 부트스트랩해야 합니다.

부트스트랩된 리소스에 AWS CDK 저장하는 AWS 항목에 대해 요금이 부과될 수 있습니다. 또한 를 사용하면 `-bootstrap-customer-key` AWS KMS 키가 생성되며, 이 경우에도 환경별로 요금이 부과됩니다.

> **Note**
>
> 이전 버전의 부트스트랩 템플릿에서는 기본적으로 KMS 키를 생성했습니다. 요금이 부과되지 않도록 하려면 를 사용하여 다시 부트스트랩하세요. `--no-bootstrap-customer-key`

> **Note**
>
> CDK Toolkit v2는 CDK v1에서 기본적으로 사용되는 레거시 템플릿이라고 하는 원본 부트스트랩 템플릿을 지원하지 않습니다.

> **Important**
>
> 최신 부트스트랩 템플릿은 에서 암시하는 권한을 목록에 있는 모든 계정에 효과적으로 부여합니다. `--cloudformation-execution-policies` AWS `--trust` 기본적으로 이렇게 하면 부트스트랩된 계정의 모든 리소스에 대한 읽기 및 쓰기 권한이 확장됩니다. 마음에 드는 정책 및 신뢰할 수 있는 계정으로 부트스트래핑 스택을 구성해야 합니다.

새 앱 만들기

새 앱을 만들려면 해당 앱을 위한 디렉터리를 만든 다음 디렉터리 내에서 명령을 `cdk init` 실행하십시오.

```
mkdir my-cdk-app
cd my-cdk-app
cdk init TEMPLATE --language LANGUAGE
```

지원되는 *## (LANGUAGE*) 는 다음과 같습니다.

코드	언어
`typescript`	TypeScript

코드	언어
`javascript`	JavaScript
`python`	Python
`java`	Java
`csharp`	C#

선택적 템플릿입니다. 원하는 템플릿이 기본값인 app인 경우 생략할 수 있습니다. 사용 가능한 템플릿은 다음과 같습니다.

템플릿	설명
app(기본값)	빈 AWS CDK 앱을 만듭니다.
`sample-app`	Amazon SQS 대기열과 Amazon SNS 주제를 포함하는 스택으로 AWS CDK 앱을 생성합니다.

템플릿은 프로젝트 폴더 이름을 사용하여 새 앱 내의 파일 및 클래스 이름을 생성합니다.

리스팅 스택

AWS CDK 애플리케이션에 있는 스택의 ID 목록을 보려면 다음과 같은 해당 명령 중 하나를 입력하십시오.

```
cdk list
cdk ls
```

애플리케이션에 CDK Pipelines 스택이 포함된 경우 CDK 툴킷은 파이프라인 계층 구조에서의 위치에 따라 스택 이름을 경로로 표시합니다. (예:,,`PipelineStack`) `PipelineStack/Prod` `PipelineStack/Prod/MyService`

앱에 많은 스택이 포함된 경우 나열할 스택의 전체 또는 부분 스택 ID를 지정할 수 있습니다. 자세한 정보는 the section called “스택 지정”을 참조하세요.

`--long`플래그를 추가하면 스택 이름과 해당 환경 (AWS 계정 및 지역) 을 비롯한 스택에 대한 자세한 정보를 볼 수 있습니다.

스택 합성

`cdk synthesize`명령 (거의 항상 축약됨`synth`) 은 앱에 정의된 스택을 템플릿으로 합성합니다. CloudFormation

```
cdk synth         # if app contains only one stack
cdk synth MyStack
cdk synth Stack1 Stack2
cdk synth "*"     # all stacks in app
```

> **Note**
>
> CDK 툴킷은 실제로 앱을 실행하고 대부분의 작업 (예: 스택 배포 또는 비교) 에 앞서 새 템플릿을 합성합니다. 이러한 템플릿은 기본적으로 디렉터리에 저장됩니다. `cdk.out` 이 `cdk synth` 명령은 하나 이상의 지정된 스택에 대해 생성된 템플릿을 간단히 인쇄합니다.

사용 가능한 모든 옵션은 `cdk synth --help` 을 참조하십시오. 가장 자주 사용되는 몇 가지 옵션은 다음 섹션에서 다룹니다.

컨텍스트 값 지정

`--contextor -c` 옵션을 사용하여 런타임 컨텍스트 값을 CDK 앱에 전달할 수 있습니다.

```
# 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
```

여러 스택을 배포할 때는 일반적으로 지정된 컨텍스트 값이 모든 스택에 전달됩니다. 원하는 경우 컨텍스트 값 앞에 스택 이름을 접두어로 붙여 각 스택에 다른 값을 지정할 수 있습니다.

```
# different context values for each stack
cdk synth --context Stack1:key=value Stack2:key=value Stack1 Stack2
```

표시 형식 지정

기본적으로 합성된 템플릿은 YAML 형식으로 표시됩니다. 대신 `--json` 플래그를 추가하여 JSON 형식으로 표시하세요.

```
cdk synth --json MyStack
```

출력 디렉터리 지정

합성된 템플릿을 이외의 `cdk.out` 디렉토리에 기록하려면 `--output` (`-o`) 옵션을 추가합니다.

```
cdk synth --output=~/templates
```

스택 배포

`cdk deploy`하위 명령은 하나 이상의 지정된 스택을 계정에 배포합니다. AWS

```
cdk deploy        # if app contains only one stack
cdk deploy MyStack
cdk deploy Stack1 Stack2
cdk deploy "*"    # all stacks in app
```

> **Note**
>
> CDK 툴킷은 앱을 실행하고 배포하기 전에 새로운 템플릿을 합성합니다. AWS CloudFormation 따라서 함께 사용할 수 있는 대부분의 명령줄 옵션 `cdk synth` (예:`--context`) 은 에서도 사용할 수 있습니다. `cdk deploy`

사용 가능한 모든 옵션은 `cdk deploy --help` 을 참조하십시오. 가장 유용한 몇 가지 옵션은 다음 섹션에서 다룹니다.

합성 건너뛰기

일반적으로 이 cdk deploy 명령어는 배포에 최신 버전의 앱이 반영되도록 배포하기 전에 앱 스택을 합성합니다. 마지막 코드 이후로 코드를 변경하지 않은 경우 배포 시 중복 합성 cdk synth 단계를 생략할 수 있습니다. 이렇게 하려면 옵션에 프로젝트 `cdk.out` 디렉터리를 지정하세요. --app

```
cdk deploy --app cdk.out StackOne StackTwo
```

롤백 비활성화

AWS CloudFormation 배포가 원자적으로 이루어지도록 변경 사항을 롤백하는 기능이 있습니다. 즉, 전체적으로 성공하거나 실패할 수 있습니다. 는 AWS CDK 템플릿을 합성하고 배포하기 때문에 이 기능을 상속합니다. AWS CloudFormation

롤백은 리소스가 항상 일관된 상태를 유지하도록 하며, 이는 프로덕션 스택에 매우 중요합니다. 하지만 인프라를 개발하는 동안에는 일부 장애가 불가피하며, 실패한 배포를 롤백하면 속도가 느려질 수 있습니다.

이러한 이유로 CDK 툴킷을 사용하면 명령에 명령을 추가하여 롤백을 비활성화할 수 있습니다. `--no-rollback` `cdk deploy` 이 플래그를 사용하면 실패한 배포는 롤백되지 않습니다. 대신 장애가 발생한 리소스가 발생하기 전에 배포된 리소스는 그대로 유지되고 다음 배포는 실패한 리소스로 시작됩니다. 배포를 기다리는 시간이 훨씬 줄어들고 인프라 개발에 더 많은 시간을 할애할 수 있습니다.

핫 스와핑

AWS CloudFormation 변경 세트를 `cdk deploy` 생성하여 배포하는 대신 `--hotswap` 플래그와 함께 사용하면 AWS 리소스를 직접 업데이트할 수 있습니다. 핫 스왑이 불가능한 경우 AWS CloudFormation 배포는 배포로 대체됩니다.

현재 핫 스와핑은 Lambda 함수, Step Functions 상태 머신, Amazon ECS 컨테이너 이미지를 지원합니다. `--hotswap`플래그는 또한 롤백을 비활성화합니다 (예: 암시). `--no-rollback`

> **Important**
>
> 프로덕션 배포에는 핫 스왑을 사용하지 않는 것이 좋습니다.

감시 모드

CDK 툴킷의 감시 모드 (cdk deploy --watch또는 줄여서) 는 CDK 앱의 소스 파일 및 cdk watch 자산에 변경 사항이 있는지 지속적으로 모니터링합니다. 변경이 감지되면 지정된 스택을 즉시 배포합니다.

기본적으로 이러한 배포에는 Lambda 함수에 대한 변경 사항 배포를 신속하게 추적하는 `--hotswap` 플래그가 사용됩니다. 또한 인프라 구성을 변경한 AWS CloudFormation 경우 배포로 대체됩니다. `cdk`

watch항상 전체 AWS CloudFormation 배포를 수행하도록 하려면 플래그를 에 추가하십시오. --no-hotswap cdk watch

이미 배포를 수행하는 동안 cdk watch 변경한 모든 내용은 단일 배포로 통합되며, 진행 중인 배포가 완료되는 즉시 배포가 시작됩니다.

감시 모드에서는 프로젝트의 "watch" 키를 사용하여 cdk.json 모니터링할 파일을 결정합니다. 기본적으로 이러한 파일은 애플리케이션 파일 및 에셋이지만 "watch" 키의 "include" 및 "exclude" 항목을 수정하여 변경할 수 있습니다. 다음 cdk.json 파일은 이러한 항목의 예를 보여줍니다.

```
{
  "app": "mvn -e -q compile exec:java",
  "watch": {
    "include": "src/main/**",
    "exclude": "target/*"
  }
}
```

cdk watch합성 전에 의 "build" 명령을 cdk.json 실행하여 앱을 빌드합니다. 배포에 Lambda 코드 (또는 CDK 앱에 없는 다른 코드) 를 빌드하거나 패키징하기 위한 명령이 필요한 경우 여기에 추가하십시오.

및 키에는 Git 스타일 와일드카드 (*및**) 를 모두 사용할 수 있습니다. "watch" "build" 각 경로는 의 상위 디렉토리를 기준으로 해석됩니다. cdk.json 기본값은 include 이며**/*, 프로젝트 루트 디렉터리의 모든 파일 및 디렉터리를 의미합니다. exclude선택사항입니다.

> **Important**
>
> 프로덕션 배포에는 감시 모드를 사용하지 않는 것이 좋습니다.

파라미터 지정 AWS CloudFormation

AWS CDK 툴킷은 배포 시 AWS CloudFormation 매개변수 지정을 지원합니다. --parameters플래그 다음에 명령줄에서 이러한 정보를 제공할 수 있습니다.

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

매개변수를 여러 개 정의하려면 여러 개의 --parameters 플래그를 사용하십시오.

```
cdk deploy MyStack --parameters uploadBucketName=UpBucket --parameters
 downloadBucketName=DownBucket
```

여러 스택을 배포하는 경우 각 스택에 대해 각 매개 변수의 값을 다르게 지정할 수 있습니다. 이렇게 하려면 매개변수 이름 앞에 스택 이름과 콜론을 붙입니다. 그렇지 않으면 모든 스택에 동일한 값이 전달됩니다.

```
cdk deploy MyStack YourStack --parameters MyStack:uploadBucketName=UploadBucket --
parameters YourStack:uploadBucketName=UpBucket
```

기본적으로 는 이전 배포의 매개 변수 값을 AWS CDK 보존하고 명시적으로 지정하지 않은 경우 이후 배포에서 사용합니다. `--no-previous-parameters`플래그를 사용하여 모든 파라미터를 지정하도록 요구할 수 있습니다.

출력 파일 지정

스택에서 AWS CloudFormation 출력을 선언하는 경우 일반적으로 배포 종료 시 화면에 표시됩니다. JSON 형식으로 파일에 기록하려면 플래그를 사용하십시오. `--outputs-file`

```
cdk deploy --outputs-file outputs.json MyStack
```

보안 관련 변경사항

보안 상태에 영향을 미치는 의도하지 않은 변경을 방지하기 위해 AWS CDK 툴킷은 보안 관련 변경을 배포하기 전에 승인하라는 메시지를 표시합니다. 승인이 필요한 변경 수준을 지정할 수 있습니다.

```
cdk deploy --require-approval LEVEL
```

LEVEL# 다음 중 하나일 수 있습니다.

용어	의미
`never`	승인은 절대 필요하지 않습니다.
`any-change`	모든 IAM 또는 security-group-related 변경 사항에 대한 승인이 필요합니다.
`broadening` (기본값)	IAM 명령문 또는 트래픽 규칙 추가 시 승인이 필요합니다. 제거에는 승인이 필요하지 않습니다.

파일에서 설정을 구성할 수도 있습니다. cdk.json

```
{
  "app": "...",
  "requireApproval": "never"
}
```

스택 비교

이 cdk diff 명령어는 앱에 정의된 스택의 현재 버전 (및 종속성) 을 이미 배포된 버전 또는 저장된 AWS CloudFormation 템플릿과 비교하고 변경 사항 목록을 표시합니다.

```
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
```

앱 스택을 기존 배포와 비교하려면:

```
cdk diff MyStack
```

앱 스택을 저장된 CloudFormation 템플릿과 비교하려면:

```
cdk diff --template ~/stacks/MyStack.old MyStack
```

기존 리소스를 스택으로 가져오기

`cdk import`명령어를 사용하여 특정 AWS CDK 스택의 CloudFormation 리소스를 관리할 수 있습니다. 이는 스택으로 AWS CDK마이그레이션하거나 스택 간에 리소스를 이동하거나 해당 논리적 ID를 변경할 때 유용합니다. `cdk import` 는 CloudFormation 리소스 가져오기를 사용합니다. 여기에서 가져올 수 있는 리소스 목록을 참조하십시오.

기존 리소스를 AWS CDK 스택으로 가져오려면 다음 단계를 따르세요.

- 현재 다른 CloudFormation 스택에서 리소스를 관리하고 있지 않은지 확인하세요. 그렇다면 먼저 리소스가 현재 속해 있는 `RemovalPolicy.RETAIN` 스택에 제거 정책을 설정하고 배포를 수행하십시오. 그런 다음 스택에서 리소스를 제거하고 다른 배포를 수행합니다. 이 프로세스를 통해 리소스가 더 이상 관리되지 않고 CloudFormation 삭제되지는 않습니다.
- `cdk diffa`를 실행하여 리소스를 가져오려는 AWS CDK 스택에 보류 중인 변경 사항이 없는지 확인합니다. “가져오기” 작업에서 허용되는 유일한 변경 사항은 가져오려는 새 리소스를 추가하는 것입니다.
- 스택으로 가져오려는 리소스의 구문을 추가합니다. 예를 들어 Amazon S3 버킷을 가져오려면 다음과 같은 것을 추가하십시오`new s3.Bucket(this, 'ImportedS3Bucket', {});`. 다른 리소스는 수정하지 마십시오.

  또한 리소스의 현재 상태를 정의에 정확히 모델링해야 합니다. 버킷의 예로는 AWS KMS 키, 라이프사이클 정책 및 기타 버킷과 관련된 모든 것을 포함해야 합니다. 그렇지 않으면 후속 업데이트 작업이 예상대로 작동하지 않을 수 있습니다.

  물리적 버킷 이름을 포함할지 여부를 선택할 수 있습니다. 일반적으로 리소스를 여러 번 배포하기 쉽도록 AWS CDK 리소스 정의에 리소스 이름을 포함하지 않는 것이 좋습니다.
- `cdk import` *`STACKNAME`*를 실행합니다.
- 리소스 이름이 모델에 없는 경우 CLI는 가져오려는 리소스의 실제 이름을 전달하라는 메시지를 표시합니다. 그런 다음 가져오기가 시작됩니다.
- 성공 `cdk import` 보고가 되면 이제 리소스는 AWS CDK 및 에 의해 관리됩니다 CloudFormation. 이후에 AWS CDK 앱의 리소스 속성에 대한 모든 변경 사항 (구성 구성) 은 다음 배포 시 적용됩니다.
- AWS CDK 앱의 리소스 정의가 리소스의 현재 상태와 일치하는지 확인하기 위해 CloudFormation 드리프트 감지 작업을 시작할 수 있습니다.

이 기능은 현재 리소스를 중첩된 스택으로 가져오는 것을 지원하지 않습니다.

구성 () cdk.json

많은 CDK Toolkit 명령줄 플래그의 기본값은 프로젝트 cdk.json 파일이나 사용자 디렉토리의 파일에 저장할 수 있습니다. .cdk.json 다음은 지원되는 구성 설정을 알파벳순으로 참조한 것입니다.

키	참고	CDK 툴킷 옵션
app	CDK 애플리케이션을 실행하는 명령입니다.	--app
assetMetadata	false경우 CDK는 자산을 사용하는 리소스에 메타데이터를 추가하지 않습니다.	--no-asset-metadata
bootstrapKmsKeyId	Amazon S3 배포 버킷을 암호화하는 데 사용된 AWS KMS 키의 ID를 재정의합니다.	--bootstrap-kms-key-id
build	합성 전에 CDK 애플리케이션을 컴파일하거나 빌드하는 명령입니다. 사용할 수 없습니다. ~/.cdk.json	--build
browser	cdk docs하위 명령의 웹 브라우저를 시작하기 위한 명령입니다.	--browser
context	the section called “컨텍스트” 섹션을 참조하십시오. 구성 파일의 컨텍스트 값은 지워지지 않습니다. cdk context --clear (CDK 툴킷은 캐시된 컨텍스트 값을 안에 배치합니다.) cdk.context.json	--context

키	참고	CDK 툴킷 옵션
`debug`	그렇다면 `true` CDK 툴킷은 디버깅에 유용한 더 자세한 정보를 내보냅니다.	--debug
`language`	새 프로젝트를 초기화하는 데 사용되는 언어입니다.	--language
`lookups`	`false`인 경우 컨텍스트 조회가 허용되지 않습니다. 컨텍스트 조회를 수행해야 하는 경우 합성이 실패합니다.	--no-lookups
`notices`	`false`If, 는 보안 취약성, 회귀 및 지원되지 않는 버전에 대한 메시지 표시를 억제합니다.	--no-notices
`output`	합성된 클라우드 어셈블리를 내보낼 디렉터리의 이름 (기본값). `"cdk.out"`	--output
`outputsFile`	배포된 스택의 AWS CloudFormation 출력이 기록되는 파일 (JSON 형식).	--outputs-file
`pathMetadata`	CDK 경로 메타데이터가 합성된 템플릿에 추가되지 않는 경우`false`.	--no-path-metadata
`plugin`	CDK를 확장하는 패키지의 패키지 이름 또는 로컬 경로를 지정하는 JSON 배열	--plugin
`profile`	지역 및 계정 자격 증명을 지정하는 데 사용되는 기본 AWS 프로필의 이름.	--profile

키	참고	CDK 툴킷 옵션
`progress`	로 `"events"` 설정하면 CDK 툴킷은 배포 중에 진행률 표시줄 대신 모든 AWS CloudFormation 이벤트를 표시합니다.	--progress
`requireApproval`	보안 변경에 대한 기본 승인 수준. the section called “보안 관련 변경사항” 섹션 참조	--require-approval
`rollback`	이 `false` 경우 배포가 실패하면 롤백되지 않습니다.	--no-rollback
`staging`	에셋이 출력 디렉터리에 복사되지 않는 경우 `false` (소스 파일의 로컬 디버깅에 사용). AWS SAM	--no-staging
`tags`	스택용 태그 (키-값 쌍) 를 포함하는 JSON 객체입니다.	--tags
`toolkitBucketName`	Lambda 함수 및 컨테이너 이미지와 같은 자산을 배포하는 데 사용되는 Amazon S3 버킷의 이름 (참조). the section called “환경 부트스트래핑 AWS”	--toolkit-bucket-name
`toolkitStackName`	부트스트랩 스택의 이름 (참조). the section called “환경 부트스트래핑 AWS”	--toolkit-stack-name
`versionReporting`	If `false` 는 버전 보고를 옵트아웃합니다.	--no-version-reporting

키	참고	CDK 툴킷 옵션
`watch`	변경 시 프로젝트 재빌드를 트리거해야 하는 파일과 트리거하지 말아야 하는 파일을 나타내는 `"exclude"` 키가 포함된 `"include"` JSON 객체입니다. the section called “감시 모드” 섹션을 참조하세요.	--watch

cdk migrate 명령 참조

AWS Cloud Development Kit (AWS CDK) 명령줄 인터페이스 (CLI) `cdk migrate` 명령에 대한 참조입니다. 사용에 대한 자세한 내용은 `cdk migrate` 을 참조하십시오기존 리소스 및 AWS CloudFormation 템플릿을 다음으로 마이그레이션하십시오. AWS CDK.

이 `cdk migrate` 명령은 배포된 AWS 리소스, AWS CloudFormation 스택 및 로컬 AWS CloudFormation 템플릿을 로 마이그레이션합니다. AWS CDK

주제

사용량

```
$ cdk migrate <options>
```

옵션

필수 옵션

`--stack-name` *STRING*

마이그레이션 후 CDK 앱 내에 생성될 AWS CloudFormation 스택의 이름.

필수 항목 여부: 예

조건부 옵션

`--from-path` *PATH*

마이그레이션할 AWS CloudFormation 템플릿의 경로. 로컬 템플릿을 지정하려면 이 옵션을 제공합니다.

필수 항목 여부: 조건부. 로컬 AWS CloudFormation 템플릿에서 마이그레이션하는 경우 필수입니다.

`--from-scan` *STRING*

AWS 환경에서 배포된 리소스를 마이그레이션할 때 이 옵션을 사용하여 새 검사를 시작할지 또는 마지막으로 성공한 스캔을 AWS CDK CLI 사용할지 여부를 지정합니다.

필수 항목 여부: 조건부. 배포된 AWS 리소스에서 마이그레이션할 때 필요합니다.

허용되는 값:, `most-recent` `new`

`--from-stack`

배포된 AWS CloudFormation 스택에서 마이그레이션하려면 이 옵션을 제공하십시오. 배포된 AWS CloudFormation 스택의 이름을 지정하는 `--stack-name` 데 사용합니다.

필수 항목 여부: 조건부. 배포된 AWS CloudFormation 스택에서 마이그레이션하는 경우 필요합니다.

옵션 옵션

`--account` *STRING*

AWS CloudFormation 스택 템플릿을 가져올 계정입니다.

필수 항목 여부: 아니요

기본값: 는 기본 소스에서 계정 정보를 AWS CDK CLI 가져옵니다.

`--compress`

생성된 CDK 프로젝트를 파일로 압축하려면 이 옵션을 제공하십시오. ZIP

필수 항목 여부: 아니요

`--filter` *ARRAY*

계정 및 계정에서 배포된 리소스를 마이그레이션할 때 사용합니다. AWS AWS 리전이 옵션은 마이그레이션할 배포된 리소스를 결정하는 필터를 지정합니다.

이 옵션은 키-값 쌍의 배열을 받아들입니다. 여기서 키는 필터 유형을 나타내고 값은 필터링할 값을 나타냅니다.

허용되는 키는 다음과 같습니다.

- `resource-identifier`— 리소스의 식별자입니다. 값은 리소스 논리적 ID 또는 물리적 ID일 수 있습니다. 예를 들어 `resource-identifier="ClusterName"`입니다.
- `resource-type-prefix`— AWS CloudFormation 리소스 유형 접두사. 예를 들어 모든 Amazon DynamoDB 리소스를 `resource-type-prefix="AWS::DynamoDB::"` 필터링하도록 지정합니다.
- `tag-key`— 리소스 태그의 키. 예를 들어 `tag-key="myTagKey"`입니다.
- `tag-value`— 리소스 태그의 값. 예를 들어 `tag-value="myTagValue"`입니다.

AND조건부 로직에 여러 개의 키-값 쌍을 제공하십시오. 다음은 태그 `myTagKey` 키로 태그가 지정된 모든 DynamoDB 리소스를 필터링하는 예제입니다. `--filter resource-type-prefix="AWS::DynamoDB::", tag-key="myTagKey"`

조건부 로직을 위해 OR 단일 명령으로 `--filter` 옵션을 여러 번 제공하십시오. 다음 예제는 DynamoDB 리소스이거나 태그 `myTagKey` 키로 태그가 지정된 모든 리소스를 필터링합니다. `--filter resource-type-prefix="AWS::DynamoDB::" --filter tag-key="myTagKey"`

필수 항목 여부: 아니요

`--language` *STRING*

마이그레이션 중에 생성된 CDK 프로젝트에 사용할 프로그래밍 언어입니다.

필수 항목 여부: 아니요

허용되는 값:`typescript`,`python`, `javacsharp`,`go`.

기본값: `typescript`

`--output-path` *PATH*

마이그레이션된 CDK 프로젝트의 출력 경로.

필수 항목 여부: 아니요

기본값: 기본적으로 AWS CDK CLI 는 현재 작업 디렉토리를 사용합니다.

`--region` *STRING*

에서 AWS CloudFormation 스택 템플릿을 검색할 수 있습니다. AWS 리전

필수 항목 여부: 아니요

기본값: 는 기본 소스에서 AWS 리전 정보를 AWS CDK CLI 가져옵니다.

AWS Visual Studio 코드용 툴킷

Visual Studio Code용AWS 툴킷은 Visual Studio Code용 오픈 소스 플러그인으로, 응용 프로그램을 더 쉽게 만들고, 디버깅하고, 배포할 수 있습니다. AWS이 툴킷은 응용 프로그램 개발을 위한 통합 환경을 제공합니다. AWS CDK 여기에는 AWS CDK 프로젝트를 나열하고 CDK 애플리케이션의 다양한 구성 요소를 탐색할 수 있는 AWS CDK 탐색기 기능이 포함되어 있습니다. AWS 툴킷을 설치하고 탐색기 사용에 대해 자세히 알아보십시오. AWS CDK

AWS SAM 통합

AWS CDK 와 AWS Serverless Application Model (AWS SAM) 를 함께 사용하면 CDK에 정의된 서버리스 애플리케이션을 로컬에서 빌드하고 테스트할 수 있습니다. 자세한 내용은 AWS SAM 개발자 AWS Cloud Development Kit (AWS CDK)안내서를 참조하십시오. SAM CLI를 설치하려면 CLI 설치를 참조하십시오. AWS SAM

테스트 구성

를 AWS CDK사용하면 작성하는 다른 코드처럼 인프라를 테스트할 수 있습니다. AWS CDK 앱을 테스트하는 표준 접근 방식은 AWS CDK의 어설션 모듈과 Jest for 및/또는 Pytest for TypeScript JavaScript Python과 같은 인기 있는 테스트 프레임워크를 사용합니다.

앱용으로 작성할 수 있는 테스트에는 두 가지 범주가 있습니다. AWS CDK

- 세분화된 어설션은 생성된 AWS CloudFormation 템플릿의 특정 측면 (예: “이 리소스는 이 값을 가진 이 속성을 가짐”) 을 테스트합니다. 이러한 테스트를 통해 회귀를 감지할 수 있습니다. 테스트 기반 개발을 사용하여 새 기능을 개발할 때도 유용합니다. (먼저 테스트를 작성한 다음 올바른 구현을 작성하여 통과하도록 할 수 있습니다.) 세밀한 어설션이 가장 자주 사용되는 테스트입니다.
- 스냅샷 테스트는 이전에 저장된 베이스라인 템플릿과 비교하여 합성된 AWS CloudFormation 템플릿을 테스트합니다. 스냅샷 테스트를 사용하면 리팩토링된 코드가 원본과 정확히 동일한 방식으로 작동하는지 확인할 수 있으므로 자유롭게 리팩토링할 수 있습니다. 의도적으로 변경한 경우 향후 테스트에 사용할 새 기준선을 적용할 수 있습니다. 하지만 CDK 업그레이드로 인해 합성된 템플릿도 변경될 수 있으므로 구현이 올바른지 확인하기 위해 스냅샷에만 의존할 수는 없습니다.

> **Note**
>
> 이 항목에서 예제로 사용된 TypeScript, Python 및 Java 앱의 전체 버전은 GitHub에서 사용할 수 있습니다.

시작하기

이러한 테스트를 작성하는 방법을 설명하기 위해 AWS Step Functions 상태 머신과 AWS Lambda 함수가 포함된 스택을 만들어 보겠습니다. Lambda 함수는 Amazon SNS 주제를 구독하고 메시지를 상태 시스템에 전달하기만 하면 됩니다.

먼저 CDK 툴킷을 사용하여 빈 CDK 애플리케이션 프로젝트를 생성하고 필요한 라이브러리를 설치합니다. 우리가 사용할 구문은 모두 CDK 툴킷으로 만든 프로젝트의 기본 종속성인 기본 CDK 패키지에 있습니다. 하지만 테스트 프레임워크는 설치해야 합니다.

TypeScript

```
mkdir state-machine && cd state-machine
```

```
cdk init --language=typescript
npm install --save-dev jest @types/jest
```

테스트용 디렉터리를 만드세요.

```
mkdir test
```

프로젝트를 package.json 편집하여 NPM에게 Jest 실행 방법을 알려주고 Jest에게 수집할 파일 종류를 알려주세요. 필요한 변경은 다음과 같습니다.

- scripts섹션에 새 test 키를 추가합니다.
- Jest와 해당 유형을 섹션에 추가합니다. devDependencies
- 선언과 함께 새 jest 최상위 키를 추가합니다. moduleFileExtensions

이러한 변경 사항은 다음 개요에 나와 있습니다. 에 표시된 곳에 새 텍스트를 package.json 배치하십시오. "..." 자리 표시자는 파일에서 변경해서는 안 되는 기존 부분을 나타냅니다.

```
{
  ...
  "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
```

테스트용 디렉터리를 만드세요.

```
mkdir test
```

프로젝트를 package.json 편집하여 NPM에게 Jest 실행 방법을 알려주고 Jest에게 수집할 파일 종류를 알려주세요. 필요한 변경은 다음과 같습니다.

- scripts섹션에 새 test 키를 추가합니다.
- 섹션에 Jest 추가 devDependencies
- 선언과 함께 새 jest 최상위 키를 추가합니다. moduleFileExtensions

이러한 변경 사항은 다음 개요에 나와 있습니다. 에 표시된 곳에 새 텍스트를 package.json 배치하십시오. "..." 자리 표시자는 파일에서 변경해서는 안 되는 기존 부분을 나타냅니다.

```
{
  ...
  "scripts": {
    ...
    "test": "jest"
  },
  "devDependencies": {
    ...
    "jest": "^24.9.0"
  },
  "jest": {
    "moduleFileExtensions": ["js"]
  }
}
```

Python

```
mkdir state-machine && cd state-machine
cdk init --language=python
source .venv/bin/activate
python -m pip install -r requirements.txt
python -m pip install -r requirements-dev.txt
```

Java

```
mkdir state-machine && cd-state-machine
cdk init --language=java
```

원하는 Java IDE에서 프로젝트를 엽니다. (Eclipse에서는 [파일] > [가져오기] > [기존 Maven 프로젝트] 를 사용합니다.)

C#

```
mkdir state-machine && cd-state-machine
cdk init --language=csharp
```

Visual Studio에서 `src\StateMachine.sln`을 엽니다.

솔루션 탐색기에서 솔루션을 마우스 오른쪽 단추로 클릭하고 추가 > 새 프로젝트를 선택합니다. MSTest C #을 검색하고 C #용 MSTest 테스트 프로젝트를 추가합니다. (기본 이름은 괜찮습니다.) `TestProject1`

마우스 오른쪽 버튼을 `TestProject1` 클릭하고 추가 > 프로젝트 참조를 선택한 다음 `StateMachine` 프로젝트를 참조로 추가합니다.

예제 스택

이 주제에서 테스트할 스택은 다음과 같습니다. 앞서 설명했듯이, 여기에는 Lambda 함수와 Step Functions 상태 머신이 포함되어 있으며 하나 이상의 Amazon SNS 주제를 수락합니다. Lambda 함수는 Amazon SNS 주제를 구독하여 상태 시스템에 전달합니다.

앱을 테스트할 수 있게 만들기 위해 특별한 조치를 취하지 않아도 됩니다. 실제로 이 CDK 스택은 이 가이드에 있는 다른 예제 스택과 중요한 측면에서 전혀 다르지 않습니다.

TypeScript

다음 코드를 삽입하세요. `lib/state-machine-stack.ts`

```
import * as cdk from "aws-cdk-lib";
import * as sns from "aws-cdk-lib/aws-sns";
import * as sns_subscriptions from "aws-cdk-lib/aws-sns-subscriptions";
import * as lambda from "aws-cdk-lib/aws-lambda";
import * as sfn from "aws-cdk-lib/aws-stepfunctions";
import { Construct } from "constructs";

export interface StateMachineStackProps extends cdk.StackProps {
  readonly topics: sns.Topic[];
}
```

```
export class StateMachineStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props: StateMachineStackProps) {
    super(scope, id, props);

    // In the future this state machine will do some work...
    const stateMachine = new sfn.StateMachine(this, "StateMachine", {
      definition: new sfn.Pass(this, "StartState"),
    });

    // This Lambda function starts the state machine.
    const func = new lambda.Function(this, "LambdaFunction", {
      runtime: lambda.Runtime.NODEJS_18_X,
      handler: "handler",
      code: lambda.Code.fromAsset("./start-state-machine"),
      environment: {
        STATE_MACHINE_ARN: stateMachine.stateMachineArn,
      },
    });
    stateMachine.grantStartExecution(func);

    const subscription = new sns_subscriptions.LambdaSubscription(func);
    for (const topic of props.topics) {
      topic.addSubscription(subscription);
    }
  }
}
```

JavaScript

다음 코드를 삽입하세요`lib/state-machine-stack.js`.

```
const cdk = require("aws-cdk-lib");
const sns = require("aws-cdk-lib/aws-sns");
const sns_subscriptions = require("aws-cdk-lib/aws-sns-subscriptions");
const lambda = require("aws-cdk-lib/aws-lambda");
const sfn = require("aws-cdk-lib/aws-stepfunctions");

class StateMachineStack extends cdk.Stack {
  constructor(scope, id, props) {
    super(scope, id, props);

    // In the future this state machine will do some work...
    const stateMachine = new sfn.StateMachine(this, "StateMachine", {
      definition: new sfn.Pass(this, "StartState"),
```

```
    });

    // This Lambda function starts the state machine.
    const func = new lambda.Function(this, "LambdaFunction", {
      runtime: lambda.Runtime.NODEJS_18_X,
      handler: "handler",
      code: lambda.Code.fromAsset("./start-state-machine"),
      environment: {
        STATE_MACHINE_ARN: stateMachine.stateMachineArn,
      },
    });
    stateMachine.grantStartExecution(func);

    const subscription = new sns_subscriptions.LambdaSubscription(func);
    for (const topic of props.topics) {
      topic.addSubscription(subscription);
    }
  }
}

module.exports = { StateMachineStack }
```

Python

다음 코드를 삽입하세요`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);
            }

        }
    }
}
```

스택을 실제로 인스턴스화하지 않도록 앱의 기본 진입점을 수정하겠습니다. 실수로 배포하고 싶지 않아요. 테스트를 통해 테스트용 앱과 스택 인스턴스를 생성합니다. 이는 테스트 기반 개발과 결합할 때 유용한 전략입니다. 배포를 활성화하기 전에 스택이 모든 테스트를 통과했는지 확인하세요.

TypeScript

bin/state-machine.ts에서:

```
#!/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

bin/state-machine.js에서:

```
#!/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

app.py에서:

```
#!/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();
        }
    }
}
```

람다 함수

예제 스택에는 상태 머신을 시작하는 Lambda 함수가 포함되어 있습니다. Lambda 함수 리소스 생성의 일환으로 CDK가 이 함수를 번들로 묶어 배포할 수 있도록 이 함수의 소스 코드를 제공해야 합니다.

- 앱의 기본 `start-state-machine` 디렉터리에 폴더를 생성합니다.
- 이 폴더에 파일을 하나 이상 생성하십시오. 예를 들어, 다음 코드를 에 저장할 수 `start-state-machines/index.js` 있습니다.

```
exports.handler = async function (event, context) {
 return 'hello world';
};
```

  하지만 스택을 실제로 배포하지는 않을 것이므로 어떤 파일이든 사용할 수 있습니다.

테스트 실행

참고로 AWS CDK 앱에서 테스트를 실행할 때 사용하는 명령은 다음과 같습니다. 이는 동일한 테스트 프레임워크를 사용하는 모든 프로젝트에서 테스트를 실행할 때 사용하는 명령과 동일합니다. 빌드 단계가 필요한 언어의 경우 해당 단계를 포함하여 테스트가 컴파일되었는지 확인하세요.

TypeScript

```
tsc && npm test
```

JavaScript

```
npm test
```

Python

```
python -m pytest
```

Java

```
mvn compile && mvn test
```

C#

솔루션을 빌드 (F6) 하여 테스트를 검색한 다음 테스트를 실행합니다 ([테스트] > [모든 테스트 실행]). 실행할 테스트를 선택하려면 테스트 탐색기를 엽니다 ([테스트] > [테스트 탐색기]).

또는 다음과 같습니다.

```
dotnet test src
```

세분화된 어설션

세분화된 어설션을 사용하여 스택을 테스트하는 첫 번째 단계는 스택을 합성하는 것입니다. 생성된 템플릿에 대해 어설션을 작성하기 때문입니다. AWS CloudFormation

상태 머신에 `StateMachineStackStack` 전달하려면 Amazon SNS 주제를 전달해야 합니다. 따라서 테스트에서는 주제를 포함할 별도의 스택을 생성해 보겠습니다.

일반적으로 CDK 앱을 작성할 때 스택의 생성자에서 Amazon SNS 주제를 `Stack` 서브클래스화하고 인스턴스화할 수 있습니다. 테스트에서는 `Stack` 직접 인스턴스화한 다음 이 스택을 스코프로 전달하여 스택에 연결합니다`Topic`. 이는 기능적으로 동일하며 장황하지 않습니다. 또한 테스트에서만 사용되는 스택을 배포하려는 스택과 “다르게” 보이게 만드는 데도 도움이 됩니다.

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
        }
    }
}
```

이제 Lambda 함수와 Amazon SNS 구독이 생성되었다고 주장할 수 있습니다.

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);
```

Lambda 함수 테스트는 함수 리소스의 두 가지 특정 속성에 특정 값이 있는지 확인합니다. 기본적으로 hasResourceProperties 메서드는 합성된 템플릿에 지정된 대로 리소스의 속성을 부분적으로 일치시킵니다. CloudFormation 이 테스트를 수행하려면 제공된 속성이 존재하고 지정된 값을 가져야 하지만 리소스에 테스트되지 않은 다른 속성이 있을 수도 있습니다.

Amazon SNS 주장은 합성된 템플릿에 구독이 포함되어 있다고 주장하지만 구독 자체에 대한 내용은 없습니다. 이 어설션은 주로 리소스 수를 기준으로 어설션을 설정하는 방법을 설명하기 위해 포함시켰습니다. 이 Template 클래스는 템플릿의, 및 섹션에 대한 어설션을 작성할 수 있는 보다 구체적인 메서드를 제공합니다. Resources Outputs Mapping CloudFormation

매처

의 기본 부분 일치 동작은 클래스의 매처를 사용하여 변경할 hasResourceProperties 수 있습니다. Match

매처의 범위는 너리언트 (Match.anyValue) 에서 strict () 까지 다양합니다. Match.objectEquals 이를 중첩하여 리소스 속성의 각 부분에 서로 다른 매칭 메서드를 적용할 수 있습니다. 예를 들어 Match.objectEquals 및 를 Match.anyValue 함께 사용하면 변경될 수 있는 속성에 대해 특정 값을 요구하지 않으면서 상태 머신의 IAM 역할을 더 완벽하게 테스트할 수 있습니다.

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" }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
```

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

많은 CloudFormation 리소스에는 문자열로 표시되는 직렬화된 JSON 객체가 포함되어 있습니다. `Match.serializedJson()`매처를 사용하여 이 JSON 내의 속성을 일치시킬 수 있습니다.

예를 들어 Step Functions 상태 머신은 JSON 기반 Amazon States 언어의 문자열을 사용하여 정의됩니다. 초기 상태가 유일한 단계인지 확인하는 `Match.serializedJson()` 데 사용합니다. 다시 말하지만, 중첩 매처를 사용하여 객체의 여러 부분에 다양한 종류의 매칭을 적용해 보겠습니다.

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

캡처하기

정확한 값을 미리 알 필요 없이 속성이 특정 형식을 따르는지 또는 다른 속성과 동일한 값을 갖는지 테스트하는 것이 유용한 경우가 많습니다. 이 assertions 모듈은 Capture클래스에서 이 기능을 제공합니다.

의 값 대신 Capture 인스턴스를 hasResourceProperties 지정하면 해당 값이 Capture 객체에 유지됩니다. 실제 캡처된 값은 asNumber() asString()asObject, 및 등의 객체 as 메서드를 사용하여 검색한 후 테스트할 수 있습니다. 매처와 Capture 함께 사용하면 직렬화된 JSON 속성을 비롯한 리소스 속성 내에서 캡처할 값의 정확한 위치를 지정할 수 있습니다.

다음 예시에서는 상태 머신의 시작 상태 이름이 로 시작되는지 테스트합니다. Start 또한 머신의 상태 목록에 이 상태가 존재하는지도 테스트합니다.

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()));
```

스냅샷 테스트

스냅숏 테스트에서는 합성된 전체 CloudFormation 템플릿을 이전에 저장된 베이스라인 (종종 “마스터”라고도 함) 템플릿과 비교합니다. 세분화된 어설션과 달리 스냅샷 테스트는 회귀를 파악하는 데 유용하지 않습니다. 스냅샷 테스트는 전체 템플릿에 적용되고 코드 변경 이외의 다른 사항으로 인해 합성 결과에 작은 (또는) 차이가 발생할 수 있기 때문입니다. not-so-small 이러한 변경은 배포에 영향을 주지 않을 수도 있지만 여전히 스냅샷 테스트에 실패하게 됩니다.

예를 들어 새로운 모범 사례를 통합하도록 CDK 구조를 업데이트할 수 있으며, 이로 인해 합성된 리소스나 구성 방식이 변경될 수 있습니다. 또는 CDK 툴킷을 추가 메타데이터를 보고하는 버전으로 업데이트할 수도 있습니다. 컨텍스트 값을 변경하면 합성된 템플릿에도 영향을 미칠 수 있습니다.

하지만 스냅샷 테스트는 합성된 템플릿에 영향을 줄 수 있는 다른 모든 요소를 일정하게 유지하는 한 리팩토링에 큰 도움이 될 수 있습니다. 변경한 내용으로 인해 템플릿이 의도치 않게 변경된 경우 즉시 알 수 있습니다. 의도적으로 변경한 경우 새 템플릿을 기준으로 적용하기만 하면 됩니다.

예를 들어 다음과 같은 구조를 사용하는 경우`DeadLetterQueue`:

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()
            });
        }
    }
}
```

다음과 같이 테스트할 수 있습니다.

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());
        }
    }
}
```

테스트 팁

테스트는 테스트하는 코드만큼이나 오래 지속되며 자주 읽고 수정된다는 점을 기억하세요. 따라서 잠시 시간을 내어 테스트 코드를 작성하는 최선의 방법을 생각해 보는 것이 좋습니다.

설정 줄이나 일반적인 어설션을 복사해서 붙여넣지 마세요. 대신 이 로직을 픽스쳐 또는 헬퍼 함수로 리팩토링하세요. 각 테스트에서 실제로 테스트한 내용을 반영하는 좋은 이름을 사용하세요.

한 번의 테스트로 너무 많은 작업을 시도하지 마세요. 가급적이면 테스트에서는 한 가지 행동만 테스트하는 것이 좋습니다. 실수로 해당 동작을 중단한 경우 정확히 하나의 테스트가 실패하고 테스트 이름을 통해 어떤 테스트가 실패했는지 알 수 있습니다. 하지만 이 방법을 시도해 보는 것이 더 이상적입니다. 때로는 부득이하게 (또는 실수로) 하나 이상의 동작을 테스트하는 테스트를 작성하게 될 수도 있습니다. 스냅숏 테스트는 앞서 설명한 이유로 특히 이 문제가 발생하기 쉬우므로 아껴서 사용하세요.

에 대한 보안 AWS Cloud Development Kit (AWS CDK)

Amazon Web Services(AWS)에서 가장 우선순위가 높은 것이 클라우드 보안입니다. AWS 고객은 가장 보안에 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처를 활용할 수 있습니다. 보안은 기업과 기업 간의 AWS 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

클라우드 보안 — AWS 클라우드에서 제공되는 모든 서비스를 실행하는 인프라를 보호하고 안전하게 사용할 수 있는 서비스를 제공하는 역할을 합니다. AWS 당사의 보안 책임은 AWS최우선 과제이며 AWS 규정 준수 프로그램의 일환으로 타사 감사자가 보안 효과를 정기적으로 테스트하고 검증합니다.

클라우드에서의 보안 — 사용자의 책임은 사용 중인 AWS 서비스와 데이터의 민감도, 조직의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요인에 따라 결정됩니다.

는 지원하는 특정 Amazon Web Services (AWS) 서비스를 통한 공동 책임 모델을 AWS CDK 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 AWS 규정 준수 프로그램의 규정 준수 노력 범위에 속하는AWS 서비스를 참조하십시오.

주제

다음을 위한 ID 및 액세스 관리 AWS Cloud Development Kit (AWS CDK)

AWS Identity and Access Management (IAM) 은 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 AWS 서비스 있도록 도와줍니다. IAM 관리자는 리소스를 사용할 수 있는 인증 (로그인) 및 권한 부여 (권한 보유) 를 받을 수 있는 사용자를 제어합니다. AWS IAM은 추가 AWS 서비스 비용 없이 사용할 수 있습니다.

고객

사용하는 방식 AWS Identity and Access Management (IAM) 은 수행하는 작업에 따라 다릅니다. AWS

서비스 사용자 - 작업을 수행하는 AWS 서비스 데 사용하는 경우 관리자가 필요한 자격 증명과 권한을 제공합니다. 더 많은 AWS 기능을 사용하여 작업을 수행함에 따라 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다.

서비스 관리자 — 회사에서 AWS 리소스를 담당하는 경우 AWS 리소스에 대한 전체 액세스 권한이 있을 수 있습니다. 서비스 사용자가 액세스해야 하는 AWS 서비스 리소스와 리소스를 결정하는 것은 여러분의 몫입니다. 그런 다음, IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해합니다.

IAM 관리자 - IAM 관리자라면 AWS 서비스에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다.

자격 증명을 통한 인증

인증은 ID 자격 증명을 AWS 사용하여 로그인하는 방법입니다. IAM 사용자로 인증 (로그인 AWS) 하거나 IAM 역할을 맡아 인증 (로그인) 해야 합니다. AWS 계정 루트 사용자

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 로그인할 수 있습니다. AWS AWS IAM Identity Center (IAM ID 센터) 사용자, 회사의 싱글 사인온 인증, Google 또는 Facebook 자격 증명이 페더레이션 ID의 예입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 액세스하는 경우 AWS 간접적으로 역할을 맡게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의 내 로그인 방법을 참조하십시오. AWS AWS 계정

AWS 프로그래밍 방식으로 액세스할 수 있도록 SDK (소프트웨어 개발 키트) 및 자격 증명을 사용하여 요청에 암호로 서명하는 명령줄 인터페이스 (CLI) 를 AWS 제공합니다. AWS CDK AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 방법에 대한 자세한 내용은 AWS 일반 참조의 Signature Version 4 서명 프로세스를 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS 계정의 보안을 강화하기 위해 다단계 인증 (MFA) 을 사용할 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 다중 인증 및 IAM 사용 설명서의 AWS에서 다중 인증(MFA) 사용을 참조하세요.

AWS 계정 루트 사용자

계정을 AWS 계정만들 때는 먼저 계정의 모든 AWS 서비스 리소스에 대한 완전한 액세스 권한을 가진 하나의 로그인 ID로 시작합니다. 이 ID를 AWS 계정 루트 사용자라고 하며, 계정을 만들 때 사용한 이메일 주소와 비밀번호로 로그인하여 액세스할 수 있습니다. 일상적인 작업에 루트 사용자를 사용하지

않을 것을 강력히 권장합니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는 작업을수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 작업의 전체 목록은 IAM 사용자 안내서의 루트 사용자 보안 인증이 필요한 작업을 참조하세요.

페더레이션 ID

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 비롯한 수동 AWS 서비스 사용자가 ID 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 액세스하도록 하는 것입니다.

페더레이션 ID는 기업 사용자 디렉토리, 웹 ID 공급자, Identity Center 디렉터리의 사용자 또는 ID 소스를 통해 제공된 자격 증명을 사용하여 액세스하는 AWS 서비스 모든 사용자를 말합니다. AWS Directory Service페더레이션 ID에 AWS 계정액세스하면 이들이 역할을 맡고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center을 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 자체 ID 소스의 사용자 및 그룹 집합에 연결하고 동기화하여 모든 사용자 및 애플리케이션에서 사용할 수 있습니다. AWS 계정 IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 IAM Identity Center란 무엇입니까?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한을 AWS 계정 가진 사용자 내 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용자 설명서의 장기 보안 인증이 필요한 사용 사례의 경우 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어 IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 보안 인증 정보를 가지고 있지만, 역할은 임시 보안 인증 정보만 제공합니다. 자세한 정보는 IAM 사용 설명서의 IAM 사용자(역할을 대신하여)를 만들어야 하는 경우를 참조하세요.

IAM 역할

IAM 역할은 특정 권한을 가진 사용자 AWS 계정 내의 자격 증명입니다. 이 역할은 IAM 사용자와 유사하지만 특정 개인과 연결되지 않습니다. 역할을 AWS Management Console 전환하여 에서 일시적으

로 IAM 역할을 맡을 수 있습니다. AWS CLI 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 정보는 IAM 사용자 설명서의 IAM 역할 사용을 참조하세요.

임시 보안 인증 정보가 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 연동 사용자 액세스 - 연동 자격 증명에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 연동 자격 증명이 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용 설명서의 서드 파티 자격 증명 공급자의 역할 만들기 부분을 참조하세요. IAM Identity Center를 사용하는 경우 권한 세트를 구성합니다. 인증 후 자격 증명이 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 IAM의 역할과 연관짓습니다. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 세트를 참조하세요.
- 임시 IAM 사용자 권한: IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 태스크에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 크로스 계정 액세스: IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 AWS 서비스경우에는 역할을 프록시로 사용하는 대신 정책을 리소스에 직접 연결할 수 있습니다. 크로스 계정 액세스를 위한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하세요.
- 서비스 간 액세스 — 일부는 다른 AWS 서비스서비스의 기능을 AWS 서비스 사용합니다. 예컨대, 어떤 서비스에서 호출을 수행하면 일반적으로 해당 서비스는 EC2에서 애플리케이션을 실행하거나 S3에 객체를 저장합니다. 서비스는 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 태스크를 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하세요.
  - 서비스 연결 역할 - 서비스 연결 역할은 에 연결된 서비스 역할의 한 유형입니다. AWS 서비스서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 사용자에게 AWS 계정 표시되며 해당 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.
- Amazon EC2에서 실행되는 애플리케이션 — IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 API 요청을 AWS CLI 하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. AWS 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있게 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인

증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 IAM 사용자 설명서의 IAM 역할(사용자 대신)을 생성하는 경우를 참조하세요.

규정 준수 검증 AWS Cloud Development Kit (AWS CDK)

는 지원하는 특정 Amazon Web Services (AWS) 서비스를 통한 공동 책임 모델을 AWS CDK 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 AWS 규정 준수 프로그램의 규정 준수 노력 범위에 속하는AWS 서비스를 참조하십시오.

AWS 서비스의 보안 및 규정 준수는 여러 AWS 규정 준수 프로그램의 일환으로 타사 감사자가 평가합니다. 여기에는 SOC, PCI, FedRAMP, HIPAA 등이 포함됩니다. AWS 규정 준수 프로그램별 범위 내 AWS 서비스에서 특정 규정 준수 프로그램 범위 내에서 자주 업데이트되는 AWS 서비스 목록을 제공합니다.

AWS Artifact를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 에서 AWS Artifact보고서 다운로드를 참조하십시오.

AWS 규정 준수 프로그램에 대한 자세한 내용은 규정 AWS 준수 프로그램을 참조하십시오.

를 사용하여 AWS 서비스에 액세스할 때의 규정 준수 책임은 데이터의 민감도, 조직의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다. AWS CDK AWS 서비스 사용이 HIPAA, PCI 또는 FedRAMP와 같은 표준을 준수해야 하는 경우 다음을 지원하는 리소스를 제공합니다. AWS

- 보안 및 규정 준수 퀵 스타트 가이드 — 아키텍처 고려 사항을 설명하고 보안 및 규정 준수에 중점을 둔 기본 환경을 배포하기 위한 단계를 제공하는 배포 안내서입니다. AWS
- AWS 규정 준수 리소스 — 업계 및 지역에 적용할 수 있는 통합 문서 및 가이드 모음입니다.
- AWS Config – 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하고 있는지 평가하는 서비스입니다.
- AWS Security Hub— AWS 해당 보안 상태를 종합적으로 파악하여 보안 업계 표준 및 모범 사례를 준수하는지 확인할 수 있습니다.

를 위한 레질리언스 AWS Cloud Development Kit (AWS CDK)

Amazon Web Services (AWS) 글로벌 인프라는 AWS 지역 및 가용 영역을 중심으로 구축됩니다.

AWS 지역은 물리적으로 분리되고 격리된 여러 가용 영역을 제공하며, 이러한 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹으로 연결됩니다.

가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 지역 및 가용 영역에 대한 자세한 내용은 글로벌 인프라를 참조하십시오AWS .

는 지원하는 특정 Amazon Web Services (AWS) 서비스를 통한 공동 책임 모델을 AWS CDK 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 AWS 규정 준수 프로그램의 규정 준수 노력 범위에 속하는AWS 서비스를 참조하십시오.

를 위한 인프라 보안 AWS Cloud Development Kit (AWS CDK)

는 지원하는 특정 Amazon Web Services (AWS) 서비스를 통한 공동 책임 모델을 AWS CDK 따릅니다. AWS 서비스 보안 정보는 AWS 서비스 보안 설명서 페이지 및 AWS 규정 준수 프로그램의 규정 준수 노력 범위에 속하는AWS 서비스를 참조하십시오.

일반적인 AWS CDK 문제 해결

이 항목에서는 에서 발생하는 다음 문제를 해결하는 방법에 대해 설명합니다. AWS CDK

- 를 AWS CDK업데이트한 후 AWS CDK 툴킷 (CLI) 이 구성 라이브러리와의 불일치를 보고합니다. AWS
- AWS CDK 스택을 배포할 때 오류가 발생합니다. `NoSuchBucket`
- AWS CDK 스택을 배포할 때 메시지를 받습니다. `forbidden: null`
- AWS CDK 스택을 합성할 때 다음과 같은 메시지가 나타납니다. `--app is required either in command-line, in cdk.json or in ~/.cdk.json`
- AWS CDK 스택을 합성할 때 AWS CloudFormation 템플릿에 너무 많은 리소스가 포함되어 있어서 오류가 발생합니다.
- Auto Scaling 그룹 또는 VPC에 대해 세 개 (또는 그 이상) 의 가용 영역을 지정했지만 두 개로만 배포되었습니다.
- 발급 시 S3 버킷, DynamoDB 테이블 또는 기타 리소스가 삭제되지 않음 `cdk destroy`

를 AWS CDK업데이트한 후 AWS CDK 툴킷 (CLI) 이 구성 라이브러리와의 불일치를 보고합니다. AWS

`cdk`명령을 제공하는 AWS CDK 툴킷의 버전은 최소한 기본 AWS 구성 라이브러리 모듈의 버전과 같아야 합니다. `aws-cdk-lib` 이 툴킷은 이전 버전과 호환되도록 만들어졌습니다. 최신 2.x 버전의 툴킷은 라이브러리의 모든 1.x 또는 2.x 릴리스와 함께 사용할 수 있습니다. 따라서 이 구성 요소를 전역적으로 설치하고 최신 상태로 유지하는 것이 좋습니다.

```
npm update -g aws-cdk
```

여러 버전의 AWS CDK 툴킷을 사용해야 하는 경우 프로젝트 폴더에 로컬로 특정 버전의 툴킷을 설치하십시오.

TypeScript 또는 를 사용하는 경우 프로젝트 디렉토리에 JavaScript CDK Toolkit의 버전이 지정된 로컬 사본이 이미 포함되어 있습니다.

다른 언어를 사용하는 경우 플래그를 생략하고 원하는 버전을 `npm` 지정하여 AWS CDK 툴킷을 설치할 때 사용하십시오. `-g` 예:

```
npm install aws-cdk@2.0
```

로컬에 설치된 AWS CDK 툴킷을 실행하려면 only 대신 명령을 npx aws-cdk 사용하십시오. cdk 예:

```
npx aws-cdk deploy MyStack
```

npx aws-cdk AWS CDK 툴킷의 로컬 버전이 있는 경우 해당 버전을 실행합니다. 프로젝트에 로컬 설치가 없는 경우 글로벌 버전으로 폴백됩니다. 가 항상 이런 방식으로 호출되도록 셸 별칭을 설정하는 cdk 것이 편리할 수도 있습니다.

macOS/Linux

```
alias cdk="npx aws-cdk"
```

Windows

```
doskey cdk=npx aws-cdk $*
```

(목록으로 돌아가기)

AWS CDK 스택을 배포할 때 오류가 발생합니다. **NoSuchBucket**

AWS 환경이 부트스트랩되지 않았으므로 배포 중에 리소스를 보관할 Amazon S3 버킷이 없습니다. 다음 명령을 사용하여 스테이징 버킷 및 기타 필수 리소스를 생성할 수 있습니다.

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

예상치 못한 AWS 요금이 발생하지 않도록 에서는 환경을 자동으로 부트스트랩하지 AWS CDK 않습니다. 배포할 각 환경을 명시적으로 부트스트랩해야 합니다.

기본적으로 부트스트랩 리소스는 현재 애플리케이션의 스택에서 사용되는 지역 또는 지역에 생성됩니다. AWS CDK 또는 로컬 AWS 프로필 (설정 기준aws configure) 에 지정된 지역에서 해당 프로필의 계정을 사용하여 만들 수도 있습니다. 다음과 같이 명령줄에서 다른 계정 및 지역을 지정할 수 있습니다. (앱 디렉터리에 없는 경우 계정 및 지역을 지정해야 합니다.)

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

자세한 설명은 the section called “부트스트래핑” 섹션을 참조하세요.

(목록으로 돌아가기)

AWS CDK 스택을 배포할 때 메시지를 받습니다. **`forbidden: null`**

부트스트랩 리소스가 필요하지만 쓰기 권한이 없는 IAM 역할 또는 계정을 사용하는 스택을 배포하고 있습니다. (스테이징 버킷은 자산을 포함하거나 5만 개 이상의 템플릿을 합성하는 스택을 배포할 때 사용됩니다.) AWS CloudFormation 오류 메시지에 언급된 버킷에 `s3:*` 대해 작업을 수행할 권한이 있는 계정 또는 역할을 사용하십시오.

(목록으로 돌아가기)

AWS CDK 스택을 합성할 때 다음과 같은 메시지가 나타납니다. **`--app is required either in command-line, in cdk.json or in ~/.cdk.json`**

이 메시지는 일반적으로 문제가 `cdk synth` 발생할 때 AWS CDK 프로젝트의 기본 디렉터리에 있지 않다는 의미입니다. `cdk init`명령으로 생성된 이 `cdk.json` 디렉터리의 파일에는 AWS CDK 앱을 실행 (및 합성) 하는 데 필요한 명령줄이 들어 있습니다. 예를 들어 TypeScript 앱의 경우 기본값은 `cdk.json` 다음과 같습니다.

```
{
  "app": "npx ts-node bin/my-cdk-app.ts"
}
```

AWS CDK 툴킷이 해당 디렉토리를 찾아 `cdk.json` 앱을 성공적으로 실행할 수 있도록 프로젝트의 기본 디렉터리에서만 `cdk` 명령을 실행하는 것이 좋습니다.

어떤 이유로든 이것이 실용적이지 않은 경우 AWS CDK 툴킷은 다른 두 위치에서 앱의 명령줄을 찾습니다.

- 홈 `cdk.json` 디렉터리에서
- `cdk synth`명령 자체에서 `-a` 옵션을 사용하여

예를 들어 다음과 같이 TypeScript 앱에서 스택을 합성할 수 있습니다.

```
cdk synth --app "npx ts-node my-cdk-app.ts" MyStack
```

(목록으로 돌아가기)

AWS CDK 스택을 합성할 때 AWS CloudFormation 템플릿에 너무 많은 리소스가 포함되어 있어서 오류가 발생합니다.

는 AWS CDK 템플릿을 생성하고 배포합니다. AWS CloudFormation AWS CloudFormation 스택에 포함할 수 있는 리소스 수에는 엄격한 제한이 있습니다. 를 AWS CDK사용하면 예상보다 더 빨리 이 한도에 도달할 수 있습니다.

> **Note**
>
> 이 글을 쓰는 시점에서 AWS CloudFormation 리소스 한도는 500입니다. 현재 리소스 AWS CloudFormation 한도는 할당량을 참조하십시오.

AWS Construct Library의 상위 수준 인텐트 기반 구조는 로깅, 키 관리, 권한 부여 및 기타 목적에 필요한 모든 보조 리소스를 자동으로 제공합니다. 예를 들어 한 리소스에 다른 리소스에 대한 액세스 권한을 부여하면 관련 서비스가 통신하는 데 필요한 IAM 객체가 생성됩니다.

경험에 따르면 실제 환경에서 인텐트 기반 구문을 사용하면 구성당 1~5개의 AWS CloudFormation 리소스가 발생하지만 이는 다를 수 있습니다. 서버리스 애플리케이션의 경우 일반적으로 API 엔드포인트당 AWS 5~8개의 리소스가 사용됩니다.

더 높은 수준의 추상화를 나타내는 패턴을 사용하면 더 적은 코드로 더 많은 AWS 리소스를 정의할 수 있습니다. 예를 the section called “ECS” 들어 의 AWS CDK 코드는 구문을 세 개만 정의하면서 50개 이상의 AWS CloudFormation 리소스를 생성합니다!

AWS CloudFormation 리소스 제한을 초과하면 AWS CloudFormation 합성 중에 오류가 발생합니다. 스택이 한도의 80% 를 초과하면 경고가 AWS CDK 발생합니다. 스택에서 `maxResources` 속성을 설정하여 다른 제한을 사용하거나 0으로 설정하여 검증을 `maxResources` 비활성화할 수 있습니다.

> **Tip**
>
> 다음 유틸리티 스크립트를 사용하여 합성된 출력의 정확한 리소스 수를 가져올 수 있습니다. (모든 AWS CDK 개발자에게 Node.js 가 필요하므로 스크립트가 JavaScript 작성됩니다.)

```
// 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");
```

스택의 리소스 수가 한도에 가까워지면 일부 Lambda 함수를 결합하거나 스택을 여러 스택으로 나누는 등 스택에 포함된 리소스 수를 줄이기 위해 아키텍처를 다시 설계해 보십시오. CDK는 스택 간 참조를 지원하므로 가장 적합한 방식으로 앱 기능을 여러 스택으로 분리할 수 있습니다.

> **Note**
>
> AWS CloudFormation 전문가들은 종종 리소스 제한에 대한 해결책으로 중첩 스택을 사용할 것을 제안합니다. 는 구문을 통해 이 접근 방식을 AWS CDK 지원합니다. `NestedStack`

(목록으로 돌아가기)

Auto Scaling 그룹 또는 VPC에 대해 세 개 (또는 그 이상) 의 가용 영역을 지정했지만 두 개로만 배포되었습니다.

요청한 가용 영역 수를 가져오려면 스택의 env 속성에 계정과 지역을 지정하십시오. 둘 다 지정하지 않으면 기본적으로 스택이 AWS CDK환경에 구애받지 않는 것으로 합성됩니다. 그런 다음 를 사용하여 스택을 특정 지역에 배포할 수 있습니다. AWS CloudFormation일부 지역에는 가용 영역이 두 개뿐이므로 환경에 구애받지 않는 템플릿은 가용 영역을 두 개 이상 사용하지 않습니다.

> **Note**
>
> 과거에는 가용 영역이 하나뿐인 지역이 출시되는 경우가 있었습니다. 환경에 구애받지 않는 AWS CDK 스택은 이러한 지역에 배포할 수 없습니다. 하지만 이 글을 쓰는 시점에서 모든 AWS 지역에는 최소 두 개의 AZ가 있습니다.

스택의 `availablilityZones`(Python:`availability_zones`) 속성을 재정의하여 사용하려는 영역을 명시적으로 지정하여 이 동작을 변경할 수 있습니다.

모든 지역에 배포할 수 있는 유연성을 유지하면서 합성 시 스택의 계정 및 지역을 지정하는 방법에 대한 자세한 내용은 을 참조하십시오. the section called “환경”

(목록으로 돌아가기)

발급 시 S3 버킷, DynamoDB 테이블 또는 기타 리소스가 삭제되지 않음 **cdk destroy**

기본적으로 사용자 데이터를 포함할 수 있는 리소스의 RETAIN 속성은 `removalPolicy` (Python:`removal_policy`) 이며 스택이 파괴되어도 리소스는 삭제되지 않습니다. 대신 리소스는 스택에서 분리됩니다. 그런 다음 스택이 파괴된 후 리소스를 수동으로 삭제해야 합니다. 그렇게 하기 전까지는 스택 재배포가 실패합니다. 이는 배포 중에 생성되는 새 리소스의 이름이 분리된 리소스의 이름과 충돌하기 때문입니다.

리소스의 제거 정책을 로 DESTROY 설정하면 스택이 제거될 때 해당 리소스가 삭제됩니다.

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 비어 있지 않은 Amazon S3 버킷은 삭제할 수 없습니다. Amazon S3 버킷의 제거 정책을 로 `DESTROY` 설정하고 여기에 데이터가 포함되어 있는 경우, 버킷을 삭제할 수 없으므로 스택 제거 시도가 실패합니다. 버킷의 `autoDeleteObjects` prop을 로 설정하여 제거를 시도하기 전에 버킷의 객체를 AWS CDK 삭제하도록 할 수 있습니다. `true`

(목록으로 돌아가기)

와 jsii를 위한 OpenPGP 키 AWS CDK

이 항목에는 및 jsii의 현재 및 과거 OpenPGP 키가 포함되어 있습니다. AWS CDK

현재 키

이 키는 AWS CDK 및 jsii의 현재 릴리스를 검증하는 데 사용해야 합니다.

AWS CDK OpenPGP 키

키 ID:	0x42B9CF2286CD987A
유형:	RSA
사이즈:	4096/4096
만든 사람:	2022-07-05
만료:	2026-07-04
사용자 ID:	AWS 클라우드 개발 키트 < aws-cdk@amazon.com >
키 핑거프린트:	69B5 2D5B A295 1D11 FA65 413B 42B9 CF22 86CD 987A

“복사” 아이콘을 선택하여 다음 OpenPGP 키를 복사하십시오.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGLEgOsBEADCoAMwvnszMLybJ+AD9cHhVyX6+rYIUEXYSgVnfkl6Z7qawIwv
wgd/a5fEs9Kiz2XJmfwS9Rxb4d+0+Y1ls1A+gnpw9FMLcZlqkC9KLnS2MqvuXWLB
t3z4kjZaL9fQ+58PoD4gy/M2hDg6gZrYqR3gtJuw8FcFpb/1KlkzRQUM8eAMFxf2
TyfjP0V0tSHwcB+84oushX7fUXVMyc3+OHsCPOe/WBFMIlWgKA+n33JKIQlUUC8f
kCWBAsAFupil0lCveT6mZu5slNRlc1I3iBLjUZ3/MtLygfqAMKwUVXeawtDvRIZe
PrAFc2NyODEhly2JG6K0FW7eIcvBqR3rg8U49t9Y74ELTM0kKnfd+flvq35xWqQC
0zghnk3kDppRTN4zWBgTKiCMxBcsHXGOoGn57t4B9VY9Zy3vkeySigeiwl/Tw9nJ
PE0SRnwEc/HnjTTfX+GTG1aQVE0xSVyZ4m5ymRNCu6+rNH8lKwo5FujlXJ+GXPkp
```

```
qT+Lx6Ix/Ny7PaoweWxwtZUkLRS4pWUsg0yotZrGyIbS+X3yMEG8WBTFI9hf6HTq
0ryfi5/TsBrdrGKqWB99EC9xYEGgtHp4fKO5X0ynOagVOhf0jSe8t1uyuJPGb2Gc
MQagSys5xMhdG/ZnEY4Cb+JDtH/4jc3tca0+4Z5RQ7kF9IhCncFtrbjJbwARAQAB
tC5BV1MgQ2xvdWQgRGV2ZWxvcG1lbnQgS2l0IDxhd3MtY2RrQGFtYXpvbi5jb20+
iQI/BBMBAgApBQJixIDrAhsvBQkHhM4ABwsJCAcDAgEGFQgCCQoLBBYCAwECHgEC
F4AACgkQQrnPIobNmHo2qg//Zt9p/kN1DevflzxWKouUX0AS7UmUtRYXu5k/EEbu
wkYNHpUr7+lZ+Me5YyjcIpt6UwuG9cW4SvwuxIfXucyKAWiwEbydCQauvnrYDxDa
J6Yr/ntk7Sii6An9re99qic3IsvX+xlUXh+qJ/34ooP/1PHziCMqykvW/DwAIyhx
2qvTXy+9+OlOWSUbhkCnNz5XKb4XQGq73DqalZX1nH4dG6fckZmYRX+dpw2njfTw
ZLdZ7bkrfiL84FI4A21RfSbEU4s4ngiV17lZ9ivilBKTbDv3da7+yc919M7C5N4J
yrlxvtyYNDoqKAD2WYZAnpEbG/shu3f56RyOJd56tXGwl9nKPh+F9y+379XthSwA
xZTURFtjWf7wWHaDZadU0DKi+Oeeszjg2f/VJaGmmS8PIg7q6GiSHHpqHqNvACHm
ZXMw12QFd3qt3xu0JMmE11ZC5VBgblwpkQTrO04Sq1rOpJwXI9ODMS/ZEhAIoYmT
OR7ouknlAx6mj9fwpavWDAAJHLdVUMYBZTXiQYFzDvx51ivvTRWkB1zTJcFdqShY
B37+Jz2jLDNdMrcHk2yfVp/VvfbxKcexg8wEwrrtQUslTUenl5jBZJouoz/wW81s
Y4U1nCPCdTK5/C7JCKzR2gVnCpe6uaxAWkkM2feQhjqJZkTC4cFVgBT+4M6WcT1r
yq4=
=ahbs
-----END PGP PUBLIC KEY BLOCK-----
```

jsii OpenPGP 키

키 ID:	0x056C4E15DAE3D8D9
유형:	RSA
사이즈:	4096/4096
만든 사람:	2022-07-05
만료:	2026-07-04
사용자 ID:	AWS JSII 팀 < aws-jsii@amazon.com >
키 핑거프린트:	1E07 31D4 57E5 FE87 87E5 530A 056C 4E15 DAE3 D8D9

“복사” 아이콘을 선택하여 다음 OpenPGP 키를 복사합니다.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
```

```
mQINBGLEgOkBEAD27EPVG9g2mHQ3+M6tF6le+tfhARJ2EV7m7NKIrTdSlCZATLWn
AVLlxG1unW34NlkKZbcbR86gAxRnnAhuEhPuloU/S5wAqPGbRiFl58YjYZDNJw6U
1SSMpE4O1sfjxv9yAbiRihLYtvksyHHZmaDhYner2aK1PdeWu+BKq/tjfm3Yzsd2
uuVEduJ72YoQk/29dEiGOHfT+2kUKxUX+0tJSJ9MGlEf4NtQE4WLzrT6Xqb2SG4+
alIiIVxIEi0XKDn7n8ZLjFwfJwOYxVYLtEUkqFWM8e8vgoc9/nYc+vDXZVED2g3Z
FWrwSnDSXbQpnMa2cLhD4xLpDHUS3i2p7r3dkJQGLo/5JGOopLibrOAbYZ72izhu
H/TuPFogSz0mNFPglrWdnLF04UIjIq420+06V4WQZC9n55Zjcbki/OhnC3B9pAdU
tiy8zg070bWq45dPGf5STkPPn7G8A2zmKefyO51iLIi26ZzW78siB+FvcGRhdg25
39sHJ1cmrTeC+B+k4KeV5sQ/m3UucimrZnk1xdaiVp8mWzRqWb8bB6Rs8K9RMrMV
tFBOKOBAT2QxOQtRGAantVgm193E1T1cmNpD0FKAKkDdPs64rKBEwFiHxccXHbah
eMd1weVwn3AKFD6uAm8ZRMV+dyssfcQxqpo/kfT1XpA6cQeOmGDOcKBfdwARAQAB
tCNBV1MgSlNJSSBUZWFtIDxhd3MtanNpaUBhbWF6b24uY29tPokCPwQTAQIAKQUC
YsSA6QIbLwUJB4TOAAcLCQgHAwIBBhUIAgkKCwQWAgMBAh4BAheAAAoJEAVsThXa
49jZjU4QANoyqOJUT4gRrXshE3N0mW5Ad4i8Ke09GA62HyvTtfbsA+2nkNVGJpXm
sFMzdaFO95Q65RkLS9vW4nhhjXBEc2XYNCt2AnARudA/41ykjDPwU112z9ZTB9he
y4ItIeNGpHvMWr51fihl0y2nkpODOBeiv44jscLbHyOmZfki1f5fuIu2U2IbUGK3
5FtYyeHcgRHnpYkzLuzK4PfayOywqQPJ7M9DWrHf+v5Cu4ZCZDOIKfzF+ew7MWwc
6KaoWHCYbFpX8jxFppbGsSFOQ8Sl2quoP0TLz9Wsq70Khi6C2P8JI6lm0HRLO+1M
jFbQxNOwAcN3k4HSwunAjXBlmT/6oc1RsdBdpXBaZ2AWseIXwSYZqNXp+5L179uZ
vSiD3DSSUqLJbdQRVOsJi3/87V5QU59byq2dToHveRjtSbVnK0TkTx9ZlgkcpjvM
BwHNqWhratV6af2Upjq2YQ0fdSB42f3pgopInxNJPMvlAb+cCfr0Pfwu7ge7UooQ
WHTxbpCvwtn/HNctMGpWscO02WsWgoYVjnVFay/XphE77pQ9rRUkhMe6VKXfxj/n
OCZJKrydluIIwR8vvONNqO+QwZ1xDEhO7MaSZlOm1AuUZIXFPgaWQkPZHKiiwFA/
QWnL/+shuRtMH2geTjkev198Jgb5HyXFm4SyYtZferQROyliEhik
=BuGv
-----END PGP PUBLIC KEY BLOCK-----
```

히스토리컬 키

이러한 키는 2022-07-05 이전에 AWS CDK 및 jsii의 릴리스를 검증하는 데 사용될 수 있습니다.

> **Important**
>
> 새 키는 이전 키가 만료되기 전에 생성됩니다. 따라서 언제든지 둘 이상의 키가 유효할 수 있습니다. 키는 생성된 날부터 아티팩트에 서명하는 데 사용되므로 키의 유효성이 겹치는 경우에는 최근에 발급된 키를 사용하십시오.

AWS CDK OpenPGP 키 (2022-04-07)

> **Note**
>
> 이 키는 2022-07-05 이후에는 아티팩트에 서명하는 데 사용되지 않았습니다. AWS CDK

키 ID:	0x015584281F44A3C3
유형:	RSA
사이즈:	4096/4096
만든 사람:	2022-04-07
만료:	2026-04-06
사용자 ID:	AWS 클라우드 개발 키트 < aws-cdk@amazon.com >
키 핑거프린트:	EAE1 1A24 82B0 AA86 456E 6C67 0155 8428 1F44 A3C3

“복사” 아이콘을 선택하여 다음 OpenPGP 키를 복사하십시오.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGJPLgUBEADtlR5jQtxtBmROQvmWlPOViqqnJNhk0dULc3tXnq8NS/l6X81r
wHk+/CHG5kBunwvM0qaqLFRC6z9NnnNDxEHcTi47n+OAjWyDM6unxxWOPz8Dfaps
Uq/ZWa4by292ZeqRC9Ir2wdrizb69JbRjeshBwlJDAS/qtqCAqBRH/f7Zw7QSD6/
XTxyIy+KOVjZwFPFNHMRQ/NmgUc/Rfxsa0pUjk1YAj/AkvQlwwD8DEnASoBhO0DP
QonZxouLqIpgp4LsGo8TZdQv30ocIj0C9DuYUiUXWlCPlYPgDj6IWf3rgpMQ6nB9
wC9lx4t/L3Zg1HUD52y8aymndmbdHVn90mzlNg4XWyc58rioYrEk57YwbDnea/Kk
Hv4kVHZRfJ4/OFPyqs5ex1X3X6rb07VvA1tfLgPywO9XF2Xws8YWOWcEobaWTcnb
AzyVC6wKya8rEQzxkYJ6UkJlhDB6g6bZwIpsI2zlimG+kSBsyFvE2oRYMS0cXPqU
o+tX0+4TvxEyW3RrUQzQHIpqXrb0X1Q8Z2idPn5dwsipDEa4gsFXtrSXmbB/0Cee
eJVvKWQAsxol3+NE9L/yozq3cz5PWh0SSbmCLRcs78lMJ23MmzbMWV7BWC9DXdY+
TywY5IkDUPjGCKlD8VlrI3TgC222bH6qaua6LYCiTtRtvpDYuJNAlUjhawARAQAB
tC5BV1MgQ2xvdWQgRGV2ZWxvcG1lbnQgS2l0IDxhd3MtY2RrQGFtYXpvbi5jb20+
iQI/BBMBAgApBQJiTy4FAhsvBQkHhM4ABwsJCAcDAgEGFQgCCQoLBBYCAwECHgEC
```

```
F4AACgkQAVWEKB9Eo8NpbxAAiBF0kR/lVw3vuam60mk4l0iGMVsP8Xq6g/buzbEO
2MEB4Ftk04qOnoa+93S0ZiLR9PqxrwsGSp4ADDX3Vtc4uxwzUlKUi1ywEhQ1cwyL
YHQI3Hd75K1J81ozMEu6qJH+yF0TtTDZMeZHtH/XvuIYJW3Lx4o5ZFlsEegFPAgX
YCCpUS+k9qC6M8g2VjcltQJpyjGswsKm6FWaKHW+B9dfjdOHlImB9E2jaknJ8eoY
zb9zHgFANluMzpZ6rYVSiCuXiEgYmazQWCvlPcMOP7nX+1hq1z11LMqeSnfE09gX
H+OYho9cMEJkb1dzx1H9MRpylFIn9tL+2iCp4UPJjnqi6uawWyLZ2tp4G11haqQq
1yAh69u233I8GZKFUySzjHwH5qWGRgBTjrZ6FdcjSS2w/wMkVKuCPkWtdvo/TJrm
msCd1Reye8SEKYqrs0ujTwmlvWmUZmO06AdUjo1kWiBKeslTJrWEuG7Yk4pFOoA4
dsaq83gxpOJNVCh6M3y4DLNrvl7dhF95NwTWMROPj2otw7NIjF4/cdzve2+P7YNN
pVAtyCtTJdD3eZbQPVaL3T8cf1VGqt6++pnLGnWJ0+X3TyvfmTohdJvN3TE+tq7A
7cprDX/q9c56HaXdJzVpxEzuf/YC+JuYKeHwsX3QouDhyRg3PsigdZES/02Wr8so
l6U=
=MQI4
-----END PGP PUBLIC KEY BLOCK-----
```

jsii OpenPG 키 (2022-04-07)

> **Note**
>
> 이 키는 2022-07-05 이후에는 jsii 아티팩트에 서명하는 데 사용되지 않았습니다.

키 ID:	0x985F5BC974B79356
유형:	RSA
사이즈:	4096/4096
만든 사람:	2022-04-07
만료:	2026-04-06
사용자 ID:	AWS JSII 팀 < aws-jsii@amazon.com >
키 핑거프린트:	35A7 1785 8FA6 282D C5AC CD95 985F 5BC9 74B7 9356

“복사” 아이콘을 선택하여 다음 OpenPGP 키를 복사하십시오.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
```

```
mQINBGJPLewBEADHH4TXup/gOlHrKDZRbj8MvsMTdM6eDteA6/c32UYV/YsK9rDA
jN8Jv/xlfosOebcHrfnFpHF9VTkmjuOpN695XdwMrW/NvlEPISTGEJf21x6ZTQ2r
1xWFYzC3sl3FZmvj9XAXTmygdv+XM3TqsFgZeCaBkZVdiLbQf+FhYrovUlgotb5D
YiCQI3ofV5QTE+141jhO5Pkd3ZIoBG+P826LaT8NXhwS0o1XqVk39DCZNoFshNmR
WFZpkVCTHyv5ZhVey1NWXnD8opO375htGNV4AeSmSIH9YkURD1g5F+2t7RiosKFo
kJrfPmUjhHn8IFpReGc8qmMMZX0WaV3t+VAWfOHGGyrXDfQ4xz1VCot75C2+qypM
+qhwOAOOP0zA7CfI96ULZzSH/j8HuQk3O0DsUCybpMuKEazEMxP3tgGtRerwDaFG
jQvAlK8Rbq3v8buBI6YJuXTwSzJE8KLjleUiTFumE6WP4rsAvlP/5rBvubeMfa3n
NIMm5Rkl36Z+jt3e2Z2ZqWDPpBRta8m7QHccrZhkvqu3YC3Gl6kdnm4Vio3Xfpg2
qtWhIQutQ6DmItewV+weQHas3hl88RPJtSrfWWIIMkpbF7Y4vbX9xcnsYCLlp2Mz
tWbbnU+EWATNSsufml/Kdnu9iEEuLmeovE11I69nwjNOq9P+GJ3r/FUb2wARAQAB
tCNBV1MgSlNJSSBUZWFtIDxhd3MtanNpaUBhbWF6b24uY29tPokCPwQTAQIAKQUC
Yk8t7AIbLwUJB4TOAAcLCQgHAwIBBhUIAgkKCwQWAgMBAh4BAheAAAoJEJhfW8l0
t5NWo64P/2y7gcMRylLLW/wbrCjton2O4+YRocwQxKm1cBml9FVDUR5967YczNuu
EwEOfH/Pu3UAlrBfKAfxPNhKchLwYiOBNh2Wk5UUxRcldNHTLb5jn5gxCeWNAsl/
Tc46qY+ObdBMdOf2Vu33UCOg83WLbg1bfBoA8Bm1cd0XObtLGucu606EBt1dBrKq
9UTcbJfuGivY2Xjy5r4kEiMHBoLKcFrSo2Mm7VtYlE4Mabjyj9+orqUio7qxOl60
aa7Psa6rMvs1Ip9IOrAdG7o5Y29tQpeINH0R1/u47BrlTEAgG63Dfy49w2h/1g0G
c9KPXVuN55OWRIu0hsiySDMk/2ERsF348TU3NURZltnCOxp6pHlbPJIxRVTNa9Cn
f8tbLB3y3HfA80516g+qwNYIYiqksDdV2bz+VbvmCWcO+FellDZli831gyMGa5JJ
rq7d0lEr6nqjcnKiVwItTQXyFYmKTAXweQtVC72g1sd3oZIyqa7T8pvhWpKXxoJV
WP+OPBhGg/JEVC9sguhuv53tzVwayrNwb54JxJsD2nemfhQm1Wyvb2bPTEaJ3mrv
mhPUvXZj/I9rgsEq3L/sm2Xjy09nra4o3oe3bhEL8nOj11wkIodi17VaGP0y+H3s
I5zB5UztS6dy+cH+J7DoRaxzVzq7qtH/ZY2quClt30wwqDHUX1ef
=+iYX
-----END PGP PUBLIC KEY BLOCK-----
```

AWS CDK OpenPGP 키 (2018-06-19)

키 ID:	0x0566A784E17F3870
유형:	RSA
사이즈:	4096/4096
만든 사람:	2018-06-19
만료:	2022-06-18
사용자 ID:	AWS CDK 팀 < aws-cdk@amazon.com >

키 핑거프린트:	E88B E3B6 F0B1 E350 9E36 4F96 0566 A784 E17F 3870

“복사” 아이콘을 선택하여 다음 OpenPGP 키를 복사합니다.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBFsovE8BEADEFVCHeAVPvoQgsjVu9FPUczxy9P+2zGIT/MLI3/vPLiULQwRy
IN2oxyBNDtcDToNa/fTkW3Ev0NTP4V1h+uBoKDZD/p+dTmSDRfByECMI0sGZ3UsG
Ohhyl2Of44s0sL8gdLtDnqSRLf+ZrfT3gpgUnplW7VitkwLxr78jDpW4QD8p8dZ9
WNm3JgB55jyPgaJKqA1Ln4Vduni/1XkrG42nxrrU71uUdZPvPZ2ELLJa6n0/raG8
jq3le+xQh45gAIs6PGaAgy7jAsfbwkGTBHjjujITAY1DwvQH5iS31OaCM9n4JNpc
xGZeJAVYTLilznf2QtS/a50t+ZOmpq67Ssp2j6qYpiumm0Lo9q3K/R4/yF0FZ8SL
1TuNX0ecXEptiMVUfTiqrLsANg18EPtLZZOYW+ZkbcVytkDpiqj7bMwA7mI7zGCJ
1gjaTbcEmOmVdQYS1G6ZptwbTtvrgA6AfnZxX1HUxLRQ7tT/wvRtABfbQKAh85Ff
a3U9W4oC3c1MP5IyhNV1Wo8Zm0flZiZc0iZnojTtSG6UbcxNNL4Q8e08FWjhungj
yxSsIBnQO1Aeo1N4BbzlI+n9iaXVDUN7Kz1QEyS4PNpjvUyrUiQ+a9C5sRA7WP+x
IEOaBBGpoAXB3oLsdTNO6AcwcDd9+r2NlXlhWC4/uH2YHQUIegPqHmPWxwARAQAB
tCFBV1MgQ0RLIFRlYW0gPGF3cy1jZGtAYW1hem9uLmNvbT6JAj8EEwEIACkFAlso
vE8CGy8FCQeEzgAHCwkIBwMCAQYVCAIJCgsEFgIDAQIeAQIXgAAKCRAFZqeE4X84
cLGxD/0XHnhoR2xvz38GM8HQlwlZy9W1wVhQKmNDQUavw8Zx7+iRR3m7nq3xM7Qq
BDbcbKSg1lVLSBQ6H2V6vRpysOhkPSH1nN2dO8DtvSKIPcxK48+1x7lmO+ksSs/+
oo1UvOmTDaRzOitYh3kOGXHHXk/l11GtF2FGQzYssX5iM4PHcjBsK1unThs56IMh
OJeZezEYzBaskTu/ytRJ236bPP2kZIEXfzAvhmTytuXWUXEftxOxc6fIAcYiKTha
aofG7WyR+Fvb1j5gNLcbY552QMxa23NZd5cSZH7468WEW1SGJ3AdLA7k5xvsPPOC
2YvQFD+vUOZ1JJuu6B5rHkiEMhRTLklkvqXEShTxuXiCp7iTOo6TBCmrWAT4eQr7
htLmqlXrgKi8qPkWmRdXXG+MQBzI/UyZq2q8KC6cx2md1PhANmeefhiM7FZZfeNM
WLonWfh8gVCsNH5h8WJ9fxsQCADd3Xxx3NelS2zDYBPRoaqZEEBbgUP6LnWFprA2
EkSlc/RoDqZCpBGgcoy1FFWvV/ZLgNU6OTQlYH6oYOWiylSJnaTDyurrktsxJI6d
4gdsFb6tqwTGecuUPvvZaEuvhWExLxAebhu780FdAPXgVTX+YCLI2zf+dWQvkFQf
80RE7ayn7BsiaLzFBVux/zz/WgvudsZX18r8tDiVQBL51ORmqw==
=0wuQ
-----END PGP PUBLIC KEY BLOCK-----
```

jsii OpenPG 키 (2018-08-06)

키 ID:	0x1C7ACE4CB2A1B93A
유형:	RSA

사이즈:	4096/4096
만든 사람:	2018-08-06
만료:	2022-08-05
사용자 ID:	AWS JSII 팀 < aws-jsii@amazon.com >
키 핑거프린트:	85EF 6522 4CE2 1E8C 72DB 28EC 1C7A CE4C B2A1 B93A

“복사” 아이콘을 선택하여 다음 OpenPGP 키를 복사하십시오.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBFtoSs0BEAD6WweLD0B26h0F7Jo9iR6tVQ4PgQBK1Va5H/eP+A2Iqw79UyxZ
WNzHYhzQ5MjYYI1SgcPavXy5/LV1N8HJ7QzyKszybnLYpNTLPYArWE8ZM9ZmjvIR
p1GzwnVBGQfoOlxyeutE9T5ZkAn45dTS5jlno4unji4gHjnwXKf2nP1APU2CZfdK
8vDpLOgj9LeeGlerYNbx+7xtY/I+csFIQvK09FPLSNMJQLlkBhY0r6Rt9ZQG+653
tJn+AUjyM237w0UIX1IqyYc5IONXu8HklPGu0NYuX9AY/63Ak2Cyfj0w/PZlvueQ
noQNM3j0nkOEsTOEXCyaLQw9iBKpxvLnm5RjMSODDCkj8c9uu0LHr7J4EOtgt2S1
pem7Y/c/N+/Z+Ksg9fP8fVTfYwRPvdI1x2sCiRDfLoQSG9tdrN5VwPFi4sGV04sI
x7Al8Vf/OBjAGZrDaJgM/gVvb9SKAQUA6t3ofeP14gDrS0eYodEXZ+lamnxFglxF
Sn8NRC4JFNmkXSUaTNGUdFf//F0D69PRNT8CnFfmniGj0CphN5037PCA2LC/Buq2
3+K6mTPkCcCHYPC/SwItp/xIDAQsGuDc1i1SfDYXrjsK7uOuwC5jLA9X6wZ/jgXQ
4umRRJBAV1aW8b1+yfaYYCO2AfXXO6caObv8IvH7Pc4leC2DoqylD3KklQARAQAB
tCNBV1MgSlNJSSBUZWFtIDxhd3MtanNpaUBhbWF6b24uY29tPokCPwQTAQgAKQUC
W2hKzQIbLwUJB4TOAAcLCQgHAwIBBhUIAgkKCwQWAgMBAh4BAheAAAoJEBx6zkyy
obk6B34P/iNb5QjKyhT0glZiq1wK7tuDDRpR6fC/sp6Jd/GhaNjO4BzlDbUPSjW5
950VT+qwaHXbIma/QVP7EIRztfwWy7m8eOodjpiu7JyJprhwG9nocXiNsLADcMoH
BvabkDRWXWIWSurq2wbcFMlTVwxjHPIQs6kt2oojpzP985CDS/KTzyjow6/gfMim
DLdhSSbDUM34STEgew79L2sQzL7cvM/N59k+AGyEMHZDXHkEw/Bge5Ovz50YOnsp
lisH4BzPRIw7uWqPlkVPzJKwMuo2WvMjDfgbYLbyjfvs5mqDxT2GTwAx/rd2taU6
iSqP0QmLM54BtTVVdoVXZSmJyTmXAAGlITq8ECZ/coUW9K2pUSgVuWyu63lktFP6
MyCQYRmXPh9aSd4+ielteXM9Y39snlyLgEJBhMxioZXVO2oszwluPuhPoAp4ekwj
/umVsBf6As6PoAchg7Qzr+lRZGmV9YTJOgDn2Z7jf/7tOes0g/mdiXTQMSGtp/Fp
ggnifTBx3iXkrQhqHlwtam8XTHGHy3MvX17ZslNuB8Pjh+07hhCxv0VUVZPUHJqJ
ZsLa398LMteQ8UMxwJ3t06jwDWAd7mbr2tatIilLHtWWBFoCwBh1XLe/03ENCpDp
njZ7OsBsBK2nVVcN0H2v5ey0T1yE93o6r7xOwCwBiVp5skTCRUob
=2Tag
-----END PGP PUBLIC KEY BLOCK-----
```

AWS CDK 개발자 가이드 기록

릴리스에 대한 자세한 내용은 릴리스를 참조하십시오. AWS CDK 대략 일주일에 한 번 AWS CDK 업데이트됩니다. 중요한 문제를 해결하기 위해 주간 릴리스 사이에 유지 관리 버전이 릴리스될 수 있습니다. 각 릴리스에는 일치하는 AWS CDK 툴킷 (CDK CLI), AWS 구성 라이브러리 및 API 참조가 포함됩니다. 이 가이드의 업데이트는 일반적으로 릴리스와 동기화되지 않습니다. AWS CDK

> **Note**
>
> 아래 표는 중요한 문서 관련 마일스톤을 나타냅니다. 우리는 지속적으로 오류를 수정하고 콘텐츠를 개선합니다.

변경 사항	설명	날짜
CDK 마이그레이션 기능에 대한 설명서 추가	AWS CDK CLIcdk migrate명령을 사용하여 배포된 AWS 리소스, 배포된 AWS CloudFormation 스택 및 로컬 AWS CloudFormation 템플릿을 로 마이그레이션합니다. AWS CDK자세한 내용은 마이그레이션 대상을 AWS CDK 참조하십시오.	2024년 2월 2일
IAM 모범 사례 업데이트	IAM 모범 사례에 따라 가이드가 업데이트되었습니다. 자세한 내용은 IAM의 보안 모범 사례를 참조하십시오.	2023년 3월 23일
문서 cdk.json	cdk.json구성 값에 대한 설명서를 추가합니다.	2022년 4월 20일
종속성 관리	를 사용하여 종속성을 관리하는 방법에 대한 주제를 추가합니다. AWS CDK	2022년 4월 7일

Java 예제에서 이중 중괄호 제거	전체적으로 이 안티패턴을 Java 9로 바꾸십시오. `Map.of`	2022년 3월 9일
AWS CDK v2 릴리스	AWS CDK 개발자 안내서 버전 2가 출시되었습니다. CDK v1의 문서 기록	2021년 12월 04일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.