aws

Praktik terbaik untuk menggunakan AWS CDK in TypeScript untuk membuat proyek IAc

AWS Panduan Preskriptif

AWS Panduan Preskriptif: Praktik terbaik untuk menggunakan AWS CDK in TypeScript untuk membuat proyek IAc

Table of Contents

Praktik terbaik untuk menggunakan AWS CDK in TypeScript untuk membuat proyek IAc

Sandeep Gawande, Mason Cahill, Sandip Gangapadhyay, Siamak Heshmati, dan Rajneesh Tyagi, Amazon Web Services ()AWS

Oktober 2025 (sejarah dokumen)

Panduan ini memberikan rekomendasi dan praktik terbaik untuk menggunakan proyek AWS Cloud Development Kit (AWS CDK)in TypeScript untuk membangun dan menyebarkan infrastruktur skala besar sebagai kode (IAc). AWS CDK Ini adalah kerangka kerja untuk mendefinisikan infrastruktur cloud dalam kode dan penyediaan infrastruktur tersebut. AWS CloudFormation Jika Anda tidak memiliki struktur proyek yang terdefinisi dengan baik, membangun dan mengelola AWS CDK basis kode untuk proyek skala besar dapat menjadi tantangan. Untuk mengatasi tantangan ini, beberapa organisasi menggunakan anti-pola untuk proyek skala besar, tetapi pola ini dapat memperlambat proyek Anda dan menciptakan masalah lain yang berdampak negatif pada organisasi Anda. Misalnya, anti-pola dapat mempersulit dan memperlambat orientasi pengembang, perbaikan bug, dan adopsi fitur baru.

Panduan ini memberikan alternatif untuk menggunakan anti-pola dan menunjukkan kepada Anda cara mengatur kode Anda untuk skalabilitas, pengujian, dan penyelarasan dengan praktik terbaik keamanan. Anda dapat menggunakan panduan ini untuk meningkatkan kualitas kode untuk proyek IAC Anda dan memaksimalkan kelincahan bisnis Anda. Panduan ini ditujukan untuk arsitek, pemimpin teknis, insinyur infrastruktur, dan peran lain yang berusaha membangun proyek yang dirancang dengan baik untuk AWS CDK proyek skala besar.

Tujuan

- Mengurangi biaya — Anda dapat menggunakan AWS CDK untuk merancang komponen Anda sendiri yang dapat digunakan kembali yang memenuhi persyaratan keamanan, kepatuhan, dan tata kelola organisasi Anda. Anda juga dapat dengan mudah berbagi komponen di sekitar organisasi Anda, sehingga Anda dapat dengan cepat mem-bootstrap proyek baru yang selaras dengan praktik terbaik secara default.
- Waktu yang lebih cepat ke pasar — Manfaatkan fitur yang sudah dikenal AWS CDK untuk mempercepat proses pengembangan Anda. Hal ini meningkatkan penggunaan kembali untuk penyebaran dan mengurangi upaya pengembangan.

- Peningkatan produktivitas pengembang — Pengembang dapat menggunakan bahasa pemrograman yang sudah dikenal untuk mendefinisikan infrastruktur. Ini membantu pengembang mengekspresikan dan memelihara AWS sumber daya. Hal ini dapat menyebabkan peningkatan efisiensi dan kolaborasi pengembang.

Praktik terbaik

Bagian ini memberikan ikhtisar praktik terbaik berikut:

- Atur kode untuk proyek skala besar
- Kembangkan pola yang dapat digunakan kembali
- Buat atau perluas konstruksi
- Ikuti praktik TypeScript terbaik
- Memindai kerentanan keamanan dan kesalahan pemformatan
- Mengembangkan dan menyempurnakan dokumentasi
- Adopsi pendekatan pengembangan berbasis pengujian
- Gunakan rilis dan kontrol versi untuk konstruksi
- Menegakkan manajemen versi perpustakaan

Atur kode untuk proyek skala besar

Mengapa organisasi kode itu penting

Sangat penting bagi AWS CDK proyek skala besar untuk memiliki struktur berkualitas tinggi dan terdefinisi dengan baik. Ketika sebuah proyek semakin besar dan jumlah fitur dan konstruksi yang didukung bertambah, navigasi kode menjadi lebih sulit. Kesulitan ini dapat memengaruhi produktivitas dan memperlambat orientasi pengembang.

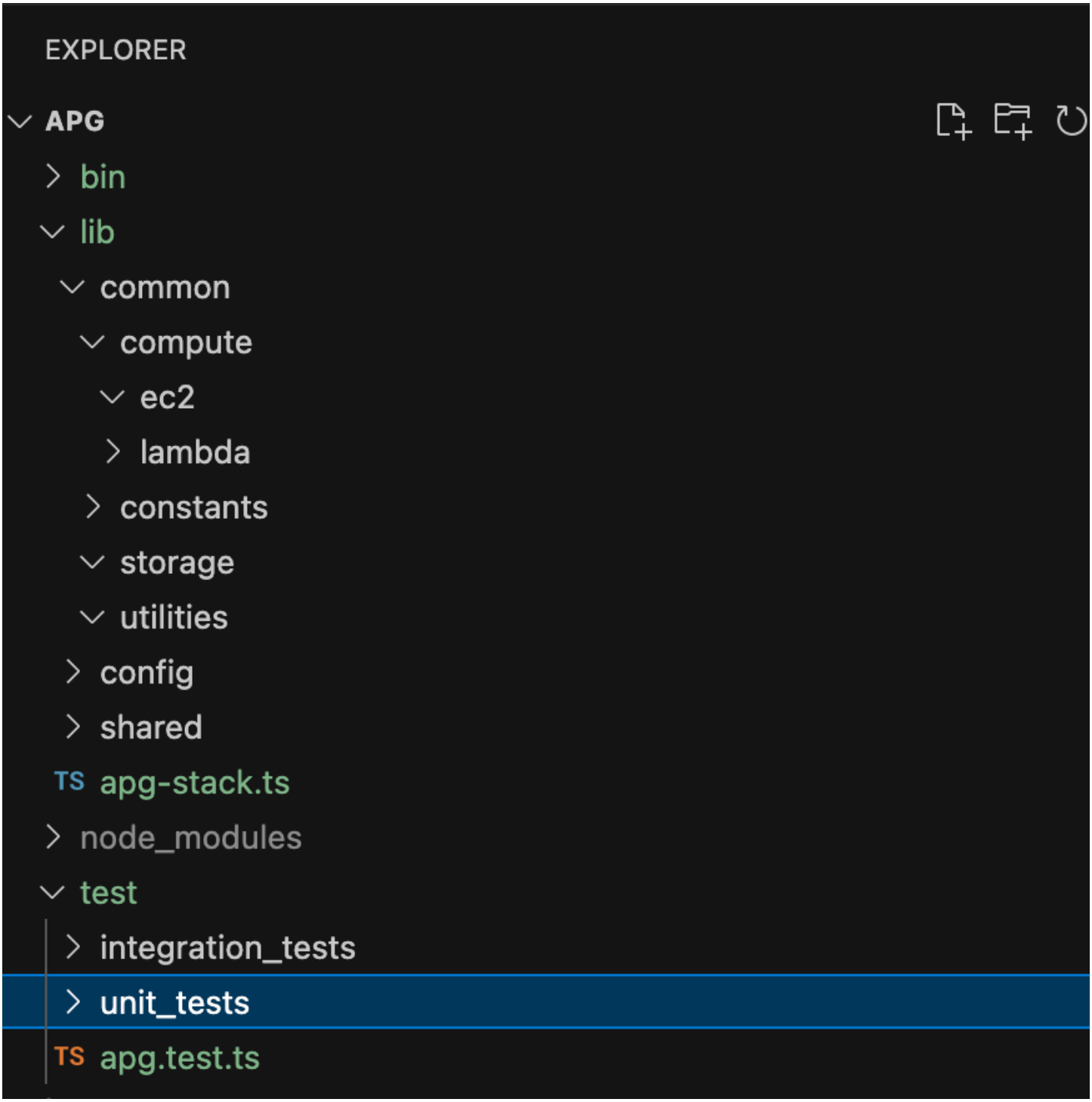
Cara mengatur kode Anda untuk skala

Untuk mencapai tingkat fleksibilitas dan keterbacaan kode yang tinggi, kami sarankan Anda membagi kode Anda menjadi potongan-potongan logis berdasarkan fungsionalitas. Divisi ini mencerminkan fakta bahwa sebagian besar konstruksi Anda digunakan dalam domain bisnis yang berbeda. Misalnya, aplikasi frontend dan backend Anda mungkin memerlukan AWS Lambda fungsi dan menggunakan kode sumber yang sama. Pabrik dapat membuat objek tanpa mengekspos logika penciptaan ke klien dan menggunakan antarmuka umum untuk merujuk ke objek yang baru dibuat. Anda dapat menggunakan pabrik sebagai pola yang efektif untuk membuat perilaku yang konsisten dalam basis kode Anda. Selain itu, pabrik dapat berfungsi sebagai sumber kebenaran tunggal untuk membantu Anda menghindari kode berulang dan mempermudah pemecahan masalah.

Untuk lebih memahami cara kerja pabrik, perhatikan contoh pabrikan mobil. Pabrikan mobil tidak perlu memiliki pengetahuan dan infrastruktur yang diperlukan untuk pembuatan ban. Sebaliknya, pabrikan mobil mengalihdayakan keahlian itu ke produsen ban khusus, dan kemudian hanya memesan ban dari pabrikan itu sesuai kebutuhan. Prinsip yang sama berlaku untuk kode. Misalnya, Anda dapat membuat pabrik Lambda yang mampu membangun fungsi Lambda berkualitas tinggi, dan kemudian memanggil pabrik Lambda dalam kode Anda kapan pun Anda perlu membuat fungsi Lambda. Demikian pula, Anda dapat menggunakan proses outsourcing yang sama ini untuk memisahkan aplikasi Anda dan membangun komponen modular.

Organisasi kode sampel

TypeScript Contoh proyek berikut, seperti yang ditunjukkan pada gambar berikut, menyertakan folder umum tempat Anda dapat menyimpan semua konstruksi atau fungsi umum Anda.



Misalnya, folder komputasi (berada di folder umum) menyimpan semua logika untuk konstruksi komputasi yang berbeda. Pengembang baru dapat dengan mudah menambahkan konstruksi komputasi baru tanpa memengaruhi sumber daya lainnya. Semua konstruksi lain tidak perlu membuat sumber daya baru secara internal. Sebaliknya, konstruksi ini hanya menyebut pabrik

konstruksi umum. Anda dapat mengatur konstruksi lain, seperti penyimpanan, dengan cara yang sama.

Konfigurasi berisi data berbasis lingkungan yang harus Anda pisahkan dari folder umum tempat Anda menyimpan logika. Kami menyarankan Anda menempatkan data konfigurasi umum Anda di folder bersama. Kami juga menyarankan Anda menggunakan folder utilitas untuk melayani semua fungsi pembantu dan membersihkan skrip.

Kembangkan pola yang dapat digunakan kembali

Pola desain perangkat lunak adalah solusi yang dapat digunakan kembali untuk masalah umum dalam pengembangan perangkat lunak. Mereka bertindak sebagai panduan atau paradigma untuk membantu insinyur perangkat lunak menciptakan produk yang mengikuti praktik terbaik. Bagian ini memberikan gambaran umum tentang dua pola yang dapat digunakan kembali yang dapat Anda gunakan dalam AWS CDK basis kode Anda: pola Pabrik Abstrak dan pola Rantai Tanggung Jawab. Anda dapat menggunakan setiap pola sebagai cetak biru dan menyesuaikannya untuk masalah desain tertentu dalam kode Anda. Untuk informasi selengkapnya tentang pola desain, lihat Pola Desain dalam Refactoring.Guru dokumentasi.

Pabrik Abstrak

Pola Abstrak Pabrik menyediakan antarmuka untuk membuat keluarga objek terkait atau dependen tanpa menentukan kelas konkret mereka. Pola ini berlaku untuk kasus penggunaan berikut:

- Ketika klien independen dari bagaimana Anda membuat dan menyusun objek dalam sistem
- Ketika sistem terdiri dari beberapa keluarga objek, dan keluarga ini dirancang untuk digunakan bersama
- Ketika Anda harus memiliki nilai runtime untuk membangun ketergantungan tertentu

Untuk informasi lebih lanjut tentang pola Pabrik Abstrak, lihat Pabrik Abstrak TypeScript di dalam Refactoring.Guru dokumentasi.

Contoh kode berikut menunjukkan bagaimana pola Abstrak Pabrik dapat digunakan untuk membangun pabrik penyimpanan Amazon Elastic Block Store (Amazon EBS).

```
abstract class EBSStorage {
    abstract initialize(): void;
}
```

```
class ProductEbs extends EBSStorage{
    constructor(value: String) {
        super();
        console.log(value);
    }
    initialize(): void {}
}

abstract class AbstractFactory {
    abstract createEbs(): EBSStorage
}

class EbsFactory extends AbstractFactory {
    createEbs(): ProductEbs{
        return new ProductEbs('EBS Created.')
    }
}

const ebs = new EbsFactory();
ebs.createEbs();
```

Rantai Tanggung Jawab

Chain of Responsibility adalah pola desain perilaku yang memungkinkan Anda untuk meneruskan permintaan di sepanjang rantai penangan potensial sampai salah satu dari mereka menangani permintaan. Pola Chain of Responsibility berlaku untuk kasus penggunaan berikut:

- Ketika beberapa objek, ditentukan saat runtime, adalah kandidat untuk menangani permintaan
- Bila Anda tidak ingin menentukan penangan secara eksplisit dalam kode Anda
- Saat Anda ingin mengeluarkan permintaan ke salah satu dari beberapa objek tanpa menentukan penerima secara eksplisit

Untuk informasi lebih lanjut tentang pola Rantai Tanggung Jawab, lihat Rantai Tanggung Jawab TypeScript di dalam Refactoring.Guru dokumentasi.

Kode berikut menunjukkan contoh bagaimana pola Rantai Tanggung Jawab digunakan untuk membangun serangkaian tindakan yang diperlukan untuk menyelesaikan tugas.

```
interface Handler {
```

```
    setNext(handler: Handler): Handler;
    handle(request: string): string;
}
abstract class AbstractHandler implements Handler
{
    private nextHandler: Handler;
    public setNext(handler: Handler): Handler {
        this.nextHandler = handler;
        return handler;
    }

    public handle(request: string): string {
        if (this.nextHandler) {
            return this.nextHandler.handle(request);
        }
        return '';
    }
}

class KMSHandler extends AbstractHandler {
    public handle(request: string): string {
        return super.handle(request);
    }
}
```

Buat atau perluas konstruksi

Apa itu konstruksi

Konstruk adalah blok bangunan dasar dari suatu AWS CDK aplikasi. Konstruksi dapat mewakili satu AWS sumber daya, seperti bucket Amazon Simple Storage Service (Amazon S3), atau dapat berupa abstraksi tingkat tinggi yang terdiri dari beberapa sumber daya terkait. AWS Komponen konstruksi dapat mencakup antrian pekerja dengan kapasitas komputasi yang terkait, atau pekerjaan terjadwal dengan sumber daya pemantauan dan dasbor. AWS CDK Termasuk koleksi konstruksi yang disebut AWS Construct Library. Perpustakaan berisi konstruksi untuk setiap Layanan AWS. Anda dapat menggunakan Construct Hub untuk menemukan konstruksi tambahan dari AWS, pihak ketiga, dan komunitas sumber terbuka AWS CDK .

Apa jenis konstruksi yang berbeda

Ada tiga jenis konstruksi untuk: AWS CDK

- Konstruksi L1 - Layer 1, atau L1, konstruksi persis sumber daya yang ditentukan oleh CloudFormation —tidak lebih, tidak kurang. Anda harus menyediakan sumber daya yang diperlukan untuk konfigurasi sendiri. Konstruksi L1 ini sangat mendasar dan harus dikonfigurasi secara manual. Konstruksi L1 memiliki `Cfn` awalan dan sesuai langsung dengan spesifikasi. CloudFormation Baru Layanan AWS didukung AWS CDK segera setelah CloudFormation memiliki dukungan untuk layanan ini. CfnBucketadalah contoh yang baik dari konstruksi L1. Kelas ini mewakili bucket S3 di mana Anda harus secara eksplisit mengkonfigurasi semua properti. Kami menyarankan Anda hanya menggunakan konstruksi L1 jika Anda tidak dapat menemukan konstruksi L2 atau L3 untuknya.
- Konstruksi L2 — Konstruksi Layer 2, atau L2, memiliki kode boilerplate dan logika lem yang umum. Konstruksi ini datang dengan default yang nyaman dan mengurangi jumlah pengetahuan yang perlu Anda ketahui tentang mereka. Konstruksi L2 menggunakan API berbasis niat untuk membangun sumber daya Anda dan biasanya merangkum modul L1 yang sesuai. Contoh yang baik dari konstruksi L2 adalah Bucket. Kelas ini membuat bucket S3 dengan properti dan metode default seperti bucket.add LifeCycleRule (), yang menambahkan aturan siklus hidup ke bucket.
- Konstruksi L3 — Konstruksi lapisan 3, atau L3, disebut pola. Konstruksi L3 dirancang untuk membantu Anda menyelesaikan tugas-tugas umum AWS, seringkali melibatkan berbagai jenis sumber daya. Ini bahkan lebih spesifik dan berpendirian daripada konstruksi L2 dan melayani kasus penggunaan tertentu. Misalnya, pola aws-ecs-. ApplicationLoadBalancedFargateServiceconstruct merupakan arsitektur yang mencakup cluster AWS Fargate kontainer yang menggunakan Application Load Balancer. Contoh lain adalah aws-apigateway. LambdaRestApimembangun. Konstruksi ini mewakili API Amazon API Gateway yang didukung oleh fungsi Lambda.

Ketika tingkat konstruksi semakin tinggi, lebih banyak asumsi dibuat tentang bagaimana konstruksi ini akan digunakan. Ini memungkinkan Anda untuk menyediakan antarmuka dengan default yang lebih efektif untuk kasus penggunaan yang sangat spesifik.

Cara membuat konstruksi Anda sendiri

Untuk menentukan konstruksi Anda sendiri, Anda harus mengikuti pendekatan tertentu. Ini karena semua konstruksi memperluas `Construct` kelas. `Construct`Kelas adalah blok bangunan dari pohon konstruksi. Konstruksi diimplementasikan di kelas yang memperluas kelas `Construct` dasar. Semua konstruksi mengambil tiga parameter saat diinisialisasi:

- Scope — Induk atau pemilik konstruksi, baik tumpukan atau konstruksi lain, yang menentukan tempatnya di pohon konstruksi. Anda biasanya harus lulus `this` (atau `self` dengan Python), yang mewakili objek saat ini, untuk ruang lingkup.
- id — Pengenal yang harus unik dalam lingkup ini. Identifier berfungsi sebagai namespace untuk semua yang didefinisikan dalam konstruksi saat ini dan digunakan untuk mengalokasikan identitas unik, seperti nama sumber daya dan ID logis. CloudFormation
- Props — Satu set properti yang menentukan konfigurasi awal konstruksi.

Contoh berikut menunjukkan bagaimana mendefinisikan konstruksi.

```
import { Construct } from 'constructs';

export interface CustomProps {
  // List all the properties
  Name: string;
}
export class MyConstruct extends Construct {
  constructor(scope: Construct, id: string, props: CustomProps) {
    super(scope, id);

    // TODO
  }
}
```

Buat atau perluas konstruksi L2

Konstruksi L2 mewakili “komponen cloud” dan merangkum segala sesuatu yang CloudFormation harus dimiliki untuk membuat komponen. Konstruksi L2 dapat berisi satu atau lebih AWS sumber daya, dan Anda bebas untuk menyesuaikan konstruksinya sendiri. Keuntungan membuat atau memperluas konstruksi L2 adalah Anda dapat menggunakan kembali komponen dalam CloudFormation tumpukan tanpa mendefinisikan ulang kode. Anda cukup mengimpor konstruksi sebagai kelas.

Ketika ada hubungan “is a” dengan konstruksi yang ada, Anda dapat memperluas konstruksi yang ada untuk menambahkan fitur default tambahan. Ini adalah praktik terbaik untuk menggunakan kembali properti konstruksi L2 yang ada. Anda dapat menimpa properti dengan memodifikasi properti secara langsung di konstruktor.

Contoh berikut menunjukkan bagaimana menyelaraskan dengan praktik terbaik dan memperluas konstruksi L2 yang ada yang disebut. `s3.Bucket` Ekstensi menetapkan properti default, seperti,`versioned`, `publicReadAccessblockPublicAccess`, untuk memastikan bahwa semua objek (dalam contoh ini, bucket S3) yang dibuat dari konstruksi baru ini akan selalu memiliki nilai default ini ditetapkan.

```
import * as s3 from 'aws-cdk-lib/aws-s3';
import { Construct } from 'constructs';
export class MySecureBucket extends s3.Bucket {
  constructor(scope: Construct, id: string, props?: s3.BucketProps) {

    super(scope, id, {
      ...props,
      versioned: true,
      publicReadAccess: false,
      blockPublicAccess: s3.BlockPublicAccess.BLOCK_ALL
    });
  }
}
```

Buat konstruksi L3

Komposisi adalah pilihan yang lebih baik ketika ada hubungan “memiliki” dengan komposisi konstruksi yang ada. Komposisi berarti Anda membangun konstruksi Anda sendiri di atas konstruksi lain yang ada. Anda dapat membuat pola Anda sendiri untuk merangkum semua sumber daya dan nilai defaultnya di dalam satu konstruksi L3 tingkat tinggi yang dapat dibagikan. Manfaat membuat konstruksi (pola) L3 Anda sendiri adalah Anda dapat menggunakan kembali komponen dalam tumpukan tanpa mendefinisikan ulang kode. Anda cukup mengimpor konstruksi sebagai kelas. Pola-pola ini dirancang untuk membantu konsumen menyediakan berbagai sumber daya berdasarkan pola umum dengan jumlah pengetahuan yang terbatas secara ringkas.

Contoh kode berikut menciptakan sebuah AWS CDK konstruksi yang disebut`ExampleConstruct`. Anda dapat menggunakan konstruksi ini sebagai template untuk menentukan komponen cloud Anda.

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

export interface ExampleConstructProps {
  //insert properties you wish to expose
}
```

```
export class ExampleConstruct extends Construct {
  constructor(scope: Construct, id: string, props: ExampleConstructProps) {
    super(scope, id);
    //Insert the AWS components you wish to integrate
  }
}
```

Contoh berikut menunjukkan cara mengimpor konstruksi yang baru dibuat dalam AWS CDK aplikasi atau tumpukan Anda.

```
import { ExampleConstruct } from './lib/construct-name';
```

Contoh berikut menunjukkan bagaimana Anda dapat membuat instance dari konstruksi yang Anda perpanjang dari kelas dasar.

```
import { ExampleConstruct } from './lib/construct-name';

new ExampleConstruct(this, 'newConstruct', {
  //insert props which you exposed in the interface `ExampleConstructProps`
});
```

Untuk informasi lebih lanjut, lihat AWS CDK Lokakarya di dokumentasi AWS CDK Lokakarya.

Escape hatch

Anda dapat menggunakan pintu keluar AWS CDK untuk naik tingkat abstraksi sehingga Anda dapat mengakses tingkat konstruksi yang lebih rendah. Palka pelarian digunakan untuk memperluas konstruksi untuk fitur yang tidak diekspos dengan versi saat ini AWS tetapi tersedia di. CloudFormation

Kami menyarankan Anda menggunakan pintu keluar dalam skenario berikut:

- Sebuah Layanan AWS fitur tersedia melalui CloudFormation, tetapi tidak ada `Construct` konstruksi untuk itu.
- Sebuah Layanan AWS fitur tersedia melalui CloudFormation dan ada `Construct` konstruksi untuk layanan, tetapi ini belum mengekspos fitur tersebut. Karena `Construct` konstruksi dikembangkan “dengan tangan,” mereka terkadang tertinggal dari konstruksi CloudFormation sumber daya.

Kode contoh berikut menunjukkan kasus penggunaan umum untuk menggunakan pintu keluar. Dalam contoh ini, fungsionalitas yang belum diimplementasikan dalam konstruksi tingkat yang lebih tinggi adalah untuk ditambahkan `httpPutResponseHopLimit` untuk penskalaan otomatis. `LaunchConfiguration`

```
const launchConfig = autoscaling.onDemandASG.node.findChild("LaunchConfig") as
 CfnLaunchConfiguration;
            launchConfig.metadataOptions = {
                   httpPutResponseHopLimit: autoscalingConfig.httpPutResponseHopLimit||
 2
            }
```

Contoh kode sebelumnya menunjukkan alur kerja berikut:

1. Anda mendefinisikan Anda `AutoScalingGroup` dengan menggunakan konstruksi L2. Konstruksi L2 tidak mendukung pembaruan`httpPutResponseHopLimit`, jadi Anda harus menggunakan pintu keluar.
2. Anda menggunakan `node.findChild()` metode ini untuk menemukan `LaunchConfig` anak spesifik dari `AutoScalingGroup` konstruksi L2 dan melemparkannya sebagai sumber daya. `CfnLaunchConfiguration`
3. Anda sekarang dapat mengatur `launchConfig.metadataOptions` properti pada `L1CfnLaunchConfiguration`.

Sumber daya khusus

Anda dapat menggunakan sumber daya kustom untuk menulis logika penyediaan kustom dalam template yang CloudFormation berjalan setiap kali Anda membuat, memperbarui (jika Anda mengubah sumber daya kustom), atau menghapus tumpukan. Misalnya, Anda dapat menggunakan sumber daya khusus jika ingin menyertakan sumber daya yang tidak tersedia di file AWS CDK. Dengan begitu, Anda masih dapat mengelola semua sumber daya terkait Anda dalam satu tumpukan.

Membangun sumber daya khusus melibatkan penulisan fungsi Lambda yang merespons peristiwa siklus hidup CREATE, UPDATE, dan DELETE sumber daya. Jika sumber daya kustom Anda harus membuat hanya satu panggilan API, pertimbangkan untuk menggunakan AwsCustomResourcekonstruksinya. Hal ini memungkinkan untuk melakukan panggilan SDK arbitrer

selama penerapan. CloudFormation Jika tidak, kami sarankan Anda menulis fungsi Lambda Anda sendiri untuk melakukan pekerjaan yang harus Anda selesaikan.

Untuk informasi selengkapnya tentang sumber daya kustom, lihat Sumber daya khusus dalam CloudFormation dokumentasi. Untuk contoh cara menggunakan sumber daya kustom, lihat repositori Sumber Daya Kustom aktif. GitHub

Contoh berikut menunjukkan cara membuat kelas sumber daya khusus untuk memulai fungsi Lambda dan CloudFormation mengirim sinyal sukses atau gagal.

```
import cdk = require('aws-cdk-lib');
import customResources = require('aws-cdk-lib/custom-resources');
import lambda = require('aws-cdk-lib/aws-lambda');
import { Construct } from 'constructs';

import fs = require('fs');

export interface MyCustomResourceProps {
  /**
   * Message to echo
   */
  message: string;
}

export class MyCustomResource extends Construct {
  public readonly response: string;

  constructor(scope: Construct, id: string, props: MyCustomResourceProps) {
    super(scope, id);

    const fn = new lambda.SingletonFunction(this, 'Singleton', {
      uuid: 'f7d4f730-4ee1-11e8-9c2d-fa7ae01bbebc',
      code: new lambda.InlineCode(fs.readFileSync('custom-resource-handler.py',
 { encoding: 'utf-8' })),
      handler: 'index.main',
      timeout: cdk.Duration.seconds(300),
      runtime: lambda.Runtime.PYTHON_3_6,
    });

    const provider = new customResources.Provider(this, 'Provider', {
      onEventHandler: fn,
    });
```

```
    const resource = new cdk.CustomResource(this, 'Resource', {
      serviceToken: provider.serviceToken,
      properties: props,
    });

    this.response = resource.getAtt('Response').toString();
    }
}
```

Contoh berikut menunjukkan logika utama dari sumber daya kustom.

```
def main(event, context):
    import logging as log
    import cfnresponse
    log.getLogger().setLevel(log.INFO)

    # This needs to change if there are to be multiple resources in the same stack
    physical_id = 'TheOnlyCustomResource'

    try:
        log.info('Input event: %s', event)

        # Check if this is a Create and we're failing Creates
        if event['RequestType'] == 'Create' and
 event['ResourceProperties'].get('FailCreate', False):
            raise RuntimeError('Create failure requested')

        # Do the thing
        message = event['ResourceProperties']['Message']
        attributes = {
            'Response': 'You said "%s"' % message
        }

        cfnresponse.send(event, context, cfnresponse.SUCCESS, attributes, physical_id)
    except Exception as e:
        log.exception(e)
        # cfnresponse's error message is always "see CloudWatch"
        cfnresponse.send(event, context, cfnresponse.FAILED, {}, physical_id)
```

Contoh berikut menunjukkan bagaimana AWS CDK tumpukan memanggil sumber daya kustom.

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

```
import { MyCustomResource } from './my-custom-resource';

/**
 * A stack that sets up MyCustomResource and shows how to get an attribute from it
 */
class MyStack extends cdk.Stack {
  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    const resource = new MyCustomResource(this, 'DemoResource', {
      message: 'CustomResource says hello',
    });

    // Publish the custom resource output
    new cdk.CfnOutput(this, 'ResponseMessage', {
      description: 'The message that came back from the Custom Resource',
      value: resource.response
    });
  }
}

const app = new cdk.App();
new MyStack(app, 'CustomResourceDemoStack');
app.synth();
```

Ikuti praktik TypeScript terbaik

TypeScript adalah bahasa yang memperluas kemampuan. JavaScript Ini adalah bahasa yang sangat diketik dan berorientasi objek. Anda dapat menggunakan TypeScript untuk menentukan jenis data yang diteruskan dalam kode Anda dan memiliki kemampuan untuk melaporkan kesalahan ketika jenis tidak cocok. Bagian ini memberikan ikhtisar praktik TypeScript terbaik.

Jelaskan data Anda

Anda dapat menggunakan TypeScript untuk menggambarkan bentuk objek dan fungsi dalam kode Anda. Menggunakan `any` tipe ini setara dengan memilih keluar dari jenis memeriksa variabel. Kami menyarankan Anda menghindari penggunaan `any` dalam kode Anda. Inilah contohnya.

```
type Result = "success" | "failure"
function verifyResult(result: Result) {
    if (result === "success") {
```

```
        console.log("Passed");
    } else {
        console.log("Failed")
    }
}
```

Gunakan enum

Anda dapat menggunakan enum untuk menentukan satu set konstanta bernama dan menentukan standar yang dapat digunakan kembali dalam basis kode Anda. Kami menyarankan Anda mengekspor enum Anda satu kali di tingkat global, dan kemudian membiarkan kelas lain mengimpor dan menggunakan enum. Asumsikan bahwa Anda ingin membuat serangkaian tindakan yang mungkin untuk menangkap peristiwa dalam basis kode Anda. TypeScript menyediakan enum numerik dan berbasis string. Contoh berikut menggunakan enum.

```
enum EventType {
    Create,
    Delete,
    Update
}

class InfraEvent {
    constructor(event: EventType) {
        if (event === EventType.Create) {
            // Call for other function
            console.log(`Event Captured :${event}`);
        }
    }
}

let eventSource: EventType = EventType.Create;
const eventExample = new InfraEvent(eventSource)
```

Gunakan antarmuka

Antarmuka adalah kontrak untuk kelas. Jika Anda membuat kontrak, maka pengguna Anda harus mematuhi kontrak. Dalam contoh berikut, antarmuka digunakan untuk membakukan `props` dan memastikan bahwa penelepon memberikan parameter yang diharapkan saat menggunakan kelas ini.

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

```
import { Construct } from "constructs";

interface BucketProps {
    name: string;
    region: string;
    encryption: boolean;

}

class S3Bucket extends Stack {
    constructor(scope: Construct, props: BucketProps) {
        super(scope);
        console.log(props.name);

    }
}
const app = App();
const myS3Bucket = new S3Bucket(app, {
    name: "amzn-s3-demo-bucket",
    region: "us-east-1",
    encryption: false
})
```

Beberapa properti hanya dapat dimodifikasi ketika objek pertama kali dibuat. Anda dapat menentukan ini dengan meletakkan `readonly` sebelum nama properti, seperti contoh berikut menunjukkan.

```
interface Position {
    readonly latitude: number;
    readonly longitute: number;
}
```

Perluas antarmuka

Memperluas antarmuka mengurangi duplikasi, karena Anda tidak perlu menyalin properti antar antarmuka. Selain itu, pembaca kode Anda dapat dengan mudah memahami hubungan dalam aplikasi Anda.

```
 interface BaseInterface{
    name: string;
  }
  interface EncryptedVolume extends BaseInterface{
```

```
      keyName: string;
  }
  interface UnencryptedVolume extends BaseInterface {
      tags: string[];
  }
```

Hindari antarmuka kosong

Kami menyarankan Anda menghindari antarmuka kosong karena potensi risiko yang mereka buat. Dalam contoh berikut, ada antarmuka kosong yang disebut`BucketProps`. `myS3Bucket2`Objek `myS3Bucket1` dan keduanya valid, tetapi mereka mengikuti standar yang berbeda karena antarmuka tidak memberlakukan kontrak apa pun. Kode berikut akan mengkompilasi dan mencetak properti tetapi ini memperkenalkan inkonsistensi dalam aplikasi Anda.

```
interface BucketProps {}

class S3Bucket implements BucketProps {
    constructor(props: BucketProps){
        console.log(props);
    }
}

const myS3Bucket1 = new S3Bucket({
    name: "amzn-s3-demo-bucket",
    region: "us-east-1",
    encryption: false,
});

const myS3Bucket2 = new S3Bucket({
    name: "amzn-s3-demo-bucket",
});
```

Gunakan pabrik

Dalam pola Abstrak Pabrik, antarmuka bertanggung jawab untuk membuat pabrik objek terkait tanpa secara eksplisit menentukan kelas mereka. Misalnya, Anda dapat membuat pabrik Lambda untuk membuat fungsi Lambda. Alih-alih membuat fungsi Lambda baru dalam konstruksi Anda, Anda mendelegasikan proses pembuatan ke pabrik. Untuk informasi lebih lanjut tentang pola desain ini, lihat Pabrik Abstrak TypeScript di dalam Refactoring.Guru dokumentasi.

Gunakan destrukturisasi pada properti

Destructuring, diperkenalkan di ECMAScript 6 (ES6), adalah JavaScript fitur yang memberi Anda kemampuan untuk mengekstrak beberapa bagian data dari array atau objek dan menetapkannya ke variabel mereka sendiri.

```
const object = {
    objname: "obj",
    scope: "this",
};

const oName = object.objname;
const oScop = object.scope;

const { objname, scope } = object;
```

Tentukan konvensi penamaan standar

Menegakkan konvensi penamaan membuat basis kode tetap konsisten dan mengurangi overhead saat memikirkan cara memberi nama variabel. Sebaiknya lakukan hal berikut:

- Gunakan camelCase untuk nama variabel dan fungsi.
- Gunakan UPPER_CASE untuk konstanta global untuk menunjukkan dengan jelas nilai waktu kompilasi yang tidak dapat diubah.
- Gunakan PascalCase untuk nama kelas dan nama antarmuka.
- Gunakan camelCase untuk anggota antarmuka.
- Gunakan PascalCase untuk nama tipe dan nama enum.
- Nama file dengan camelCase (misalnya, atau) `ebsVolumes.tsx` `storage.ts`

Berikut ini menunjukkan contoh konvensi penamaan yang direkomendasikan ini:

```
// Variables and functions
const userName = 'john';
function getUserData() { }

// Global constants
const MAX_RETRY_ATTEMPTS = 3;
const API_BASE_URL = 'https://api.example.com';
```

```
// Classes and interfaces
class DatabaseConnection { }
interface UserProfile { }

// Types and enums
type ResponseStatus = 'success' | 'error';
enum HttpStatusCode { }
```

Jangan gunakan kata kunci var

`let`Pernyataan ini digunakan untuk mendeklarasikan variabel lokal di. TypeScript Ini mirip dengan `var` kata kunci, tetapi memiliki beberapa batasan dalam pelingkupan dibandingkan dengan kata kunci. `var` Variabel yang dideklarasikan dalam blok dengan hanya `let` tersedia untuk digunakan di dalam blok itu. `var`Kata kunci tidak dapat dicakup blok, yang berarti dapat diakses di luar blok tertentu (diwakili oleh`{}`) tetapi tidak di luar fungsi yang ditentukan. Anda dapat mendeklarasikan ulang dan memperbarui variabel. `var` Ini adalah praktik terbaik untuk menghindari penggunaan `var` kata kunci.

Pertimbangkan untuk menggunakan ESLint dan Prettier

ESLint menganalisis kode Anda secara statis untuk menemukan masalah dengan cepat. Anda dapat menggunakan ESLint untuk membuat serangkaian pernyataan (disebut aturan lint) yang menentukan bagaimana kode Anda seharusnya terlihat atau berperilaku. ESLint juga memiliki saran pemecah otomatis untuk membantu Anda meningkatkan kode Anda. Terakhir, Anda dapat menggunakan ESLint untuk memuat aturan lint dari plugin bersama.

Prettier adalah pemformat kode terkenal yang mendukung berbagai bahasa pemrograman yang berbeda. Anda dapat menggunakan Prettier untuk mengatur gaya kode Anda sehingga Anda dapat menghindari pemformatan kode secara manual. Setelah instalasi, Anda dapat memperbarui `package.json` file Anda dan menjalankan `npm run lint` perintah `npm run format` dan.

Contoh berikut menunjukkan cara mengaktifkan ESLint dan formatter Prettier untuk proyek Anda. AWS CDK

```
"scripts": {
    "build": "tsc",
    "watch": "tsc -w",
    "test": "jest",
    "cdk": "cdk",
```

```
    "lint": "eslint --ext .js,.ts .",
    "format": "prettier --ignore-path .gitignore --write '**/*.+(js|ts|json)'"
}
```

Gunakan pengubah akses

Pengubah pribadi TypeScript membatasi visibilitas ke kelas yang sama saja. Saat Anda menambahkan pengubah pribadi ke properti atau metode, Anda dapat mengakses properti atau metode tersebut dalam kelas yang sama.

Pengubah publik memungkinkan properti dan metode kelas dapat diakses dari semua lokasi. Jika Anda tidak menentukan pengubah akses apa pun untuk properti dan metode, mereka akan mengambil pengubah publik secara default.

Pengubah yang dilindungi memungkinkan properti dan metode kelas dapat diakses dalam kelas yang sama dan dalam subkelas. Gunakan pengubah yang dilindungi saat Anda berharap untuk membuat subclass dalam aplikasi Anda AWS CDK .

Gunakan jenis utilitas

Jenis utilitas di TypeScript adalah fungsi tipe standar yang melakukan transformasi dan operasi pada tipe yang ada. Ini membantu Anda membuat tipe baru berdasarkan tipe yang ada. Misalnya, Anda dapat mengubah atau mengekstrak properti, membuat properti opsional atau wajib, atau membuat versi tipe yang tidak dapat diubah. Dengan menggunakan tipe utilitas, Anda dapat menentukan jenis yang lebih tepat dan menangkap potensi kesalahan pada waktu kompilasi.

<Tipe Parsi >

`Partial`menandai semua anggota dari jenis input `Type` sebagai opsional. Utilitas ini mengembalikan tipe yang mewakili semua himpunan bagian dari tipe tertentu. Berikut adalah contoh `Partial`.

```
interface Dog {
  name: string;
  age: number;
  breed: string;
  weight: number;
}

let partialDog: Partial<Dog> = {};
```

<Jenis yang Diperlukan >

`Required`melakukan kebalikan dari`Partial`. Itu membuat semua anggota tipe input `Type` non-opsional (dengan kata lain, diperlukan). Berikut adalah contoh `Required`.

```
interface Dog {
  name: string;
  age: number;
  breed: string;
  weight?: number;
}

let dog: Required<Dog> = {
  name: "scruffy",
  age: 5,
  breed: "labrador",
  weight: 55 // "Required" forces weight to be defined
};
```

Memindai kerentanan keamanan dan kesalahan pemformatan

Infrastruktur sebagai kode (IAc) dan otomatisasi telah menjadi penting bagi perusahaan. Dengan IAc yang begitu kuat, Anda memiliki tanggung jawab besar untuk mengelola risiko keamanan. Risiko keamanan IAc umum dapat mencakup hal-hal berikut:

- Over-permissive AWS Identity and Access Management Hak istimewa (IAM)
- Buka grup keamanan
- Sumber daya tidak terenkripsi
- Log akses tidak dihidupkan

Pendekatan dan alat keamanan

Kami menyarankan Anda menerapkan pendekatan keamanan berikut:

- Deteksi kerentanan dalam pengembangan - Memperbaiki kerentanan dalam produksi mahal dan memakan waktu karena kompleksitas pengembangan dan distribusi tambalan perangkat lunak. Selain itu, kerentanan dalam produksi membawa risiko eksploitasi. Kami menyarankan Anda

menggunakan pemindaian kode pada sumber daya IAC Anda sehingga kerentanan dapat dideteksi dan diperbaiki sebelum dirilis ke produksi.

- Kepatuhan dan remediasi otomatis — AWS Config menawarkan aturan AWS terkelola. Aturan ini membantu Anda menegakkan kepatuhan dan memungkinkan Anda mencoba remediasi otomatis dengan menggunakan otomatisasi.AWS Systems Manager Anda juga dapat membuat dan mengaitkan dokumen otomatisasi khusus dengan menggunakan AWS Config aturan.

Alat pengembangan umum

Alat yang tercakup dalam bagian ini membantu Anda memperluas fungsionalitas bawaannya dengan aturan kustom Anda sendiri. Kami menyarankan agar Anda menyelaraskan aturan kustom Anda dengan standar organisasi Anda. Berikut adalah beberapa alat pengembangan umum untuk dipertimbangkan:

- Gunakan cdk-nag untuk memvalidasi bahwa konstruksi dalam lingkup tertentu mematuhi seperangkat aturan yang ditentukan. Anda juga dapat menggunakan cdk-nag untuk penekanan aturan dan pelaporan kepatuhan. Alat cdk-nag memvalidasi konstruksi dengan memperluas aspek dalam. AWS CDK Untuk informasi selengkapnya, lihat Mengelola keamanan aplikasi dan kepatuhan terhadap AWS Cloud Development Kit (AWS CDK) dan cdk-nag di Blog. AWS DevOps
- Gunakan alat sumber terbuka Checkov untuk melakukan analisis statis pada lingkungan IAc Anda. Checkov membantu mengidentifikasi kesalahan konfigurasi cloud dengan memindai kode infrastruktur Anda di Kubernetes, Terraform, atau. CloudFormation Anda dapat menggunakan Checkov untuk mendapatkan output dalam format yang berbeda, termasuk JSON, JUnit XHTML, atau CLI. Checkov dapat menangani variabel secara efektif dengan membuat grafik yang menunjukkan ketergantungan kode dinamis. Untuk informasi lebih lanjut, lihat repositori GitHub Checkov dari Bridgecrew.
- Gunakan TFLint untuk memeriksa kesalahan dan sintaks usang dan untuk membantu Anda menerapkan praktik terbaik. Perhatikan bahwa TFLint mungkin tidak memvalidasi masalah khusus penyedia. Untuk informasi lebih lanjut tentang TFlint, lihat repositori TFlint dari GitHub Terraform Linters.
- Gunakan Pengembang Amazon Q untuk melakukan pemindaian keamanan. Saat digunakan dalam lingkungan pengembangan terintegrasi (IDE), Amazon Q Developer menyediakan bantuan pengembangan AI-powered perangkat lunak. Ini dapat mengobrol tentang kode, menyediakan penyelesaian kode sebaris, menghasilkan kode baru bersih, memindai kode Anda untuk kerentanan keamanan, dan membuat peningkatan dan peningkatan kode.

Mengembangkan dan menyempurnakan dokumentasi

Dokumentasi sangat penting untuk keberhasilan proyek Anda. Dokumentasi tidak hanya menjelaskan cara kerja kode Anda, tetapi juga membantu pengembang lebih memahami fitur dan fungsionalitas aplikasi Anda. Mengembangkan dan menyempurnakan dokumentasi berkualitas tinggi dapat memperkuat proses pengembangan perangkat lunak, memelihara perangkat lunak berkualitas tinggi, dan membantu transfer pengetahuan antar pengembang.

Ada dua kategori dokumentasi: dokumentasi di dalam kode dan dokumentasi pendukung tentang kode. Dokumentasi di dalam kode adalah dalam bentuk komentar. Dokumentasi pendukung tentang kode dapat berupa file README dan dokumen eksternal. Bukan hal yang aneh bagi pengembang untuk menganggap dokumentasi sebagai overhead, karena kode itu sendiri mudah dimengerti. Ini bisa berlaku untuk proyek kecil, tetapi dokumentasi sangat penting untuk proyek skala besar di mana banyak tim terlibat.

Ini adalah praktik terbaik bagi penulis kode untuk menulis dokumentasi karena mereka memiliki pemahaman yang baik tentang fungsinya. Pengembang dapat berjuang dengan biaya tambahan untuk memelihara dokumentasi pendukung yang terpisah. Untuk mengatasi tantangan ini, pengembang dapat menambahkan komentar dalam kode dan komentar tersebut dapat diekstraksi secara otomatis sehingga setiap versi kode dan dokumentasi akan disinkronkan.

Ada berbagai alat yang berbeda untuk membantu pengembang mengekstrak komentar dari kode dan menghasilkan dokumentasi untuk itu. Panduan ini berfokus pada TypeDoc sebagai alat yang disukai untuk AWS CDK konstruksi.

Mengapa dokumentasi kode diperlukan untuk AWS CDK membangun

AWS CDK konstruksi umum dibuat oleh beberapa tim dalam suatu organisasi dan dibagikan di berbagai tim untuk konsumsi. Dokumentasi yang baik membantu konsumen perpustakaan konstruksi dengan mudah mengintegrasikan konstruksi dan membangun infrastruktur mereka dengan sedikit usaha. Menyinkronkan semua dokumen adalah tugas besar. Kami menyarankan Anda memelihara dokumen di dalam kode, yang akan diekstraksi menggunakan TypeDoc perpustakaan.

Menggunakan TypeDoc dengan AWS Membangun Perpustakaan

TypeDoc adalah generator dokumen untuk TypeScript. Anda dapat menggunakan TypeDoc untuk membaca file TypeScript sumber Anda, mengurai komentar dalam file tersebut, dan kemudian menghasilkan situs statis yang berisi dokumentasi untuk kode Anda.

Kode berikut menunjukkan cara mengintegrasikan TypeDoc dengan AWS Construct Library, dan kemudian menambahkan paket-paket berikut dalam `package.json` file Anda di`devDependencies`.

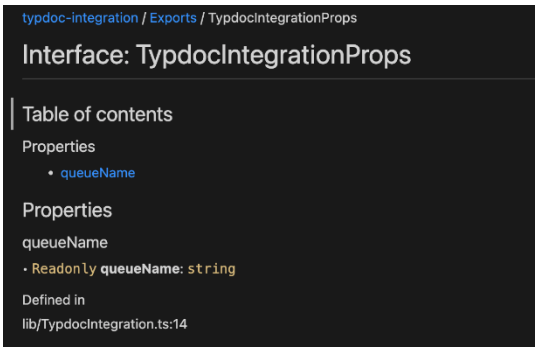
```
{

  "devDependencies": {

    "typedoc-plugin-markdown": "^3.11.7",
    "typescript": "~3.9.7"
  },

}
```

`typedoc.json`Untuk menambahkan folder perpustakaan CDK, gunakan kode berikut.

```
{
    "$schema": "https://typedoc.org/schema.json",
    "entryPoints": ["./lib"],
}
```

Untuk menghasilkan file README, jalankan `npx typedoc` perintah di direktori root proyek pustaka AWS CDK konstruksi.

Contoh dokumen berikut dihasilkan oleh TypeDoc.



Untuk informasi selengkapnya tentang opsi TypeDoc integrasi, lihat Komentar Dokumen di TypeDoc dokumentasi.

Adopsi pendekatan pengembangan berbasis pengujian

Kami menyarankan Anda mengikuti pendekatan pengembangan berbasis tes (TDD) dengan. AWS CDK TDD adalah pendekatan pengembangan perangkat lunak di mana Anda mengembangkan

kasus uji untuk menentukan dan memvalidasi kode Anda. Secara sederhana, pertama Anda membuat kasus uji untuk setiap fungsi dan jika tes gagal, maka Anda menulis kode baru untuk lulus tes dan membuat kode sederhana dan bebas bug.

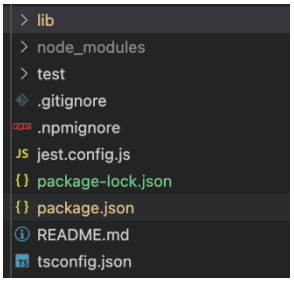
Anda dapat menggunakan TDD untuk menulis kasus uji terlebih dahulu. Ini membantu Anda memvalidasi infrastruktur dengan kendala desain yang berbeda dalam hal menegakkan kebijakan keamanan untuk sumber daya dan mengikuti konvensi penamaan unik untuk proyek. Pendekatan standar untuk menguji AWS CDK aplikasi adalah dengan menggunakan modul AWS CDKpernyataan dan kerangka pengujian populer, seperti Jest untuk TypeScript dan atau pytest untuk JavaScript Python.

Ada dua kategori tes yang dapat Anda tulis untuk AWS CDK aplikasi Anda:

- Gunakan pernyataan berbutir halus untuk menguji aspek tertentu dari CloudFormation template yang dihasilkan, seperti “sumber daya ini memiliki properti ini dengan nilai ini.” Tes ini dapat mendeteksi regresi dan juga berguna saat Anda mengembangkan fitur baru menggunakan TDD (tulis tes terlebih dahulu, lalu lulus dengan menulis implementasi yang benar). Fine-grained pernyataan adalah tes yang paling sering Anda tulis.
- Gunakan tes snapshot untuk menguji template yang disintesis terhadap CloudFormation templat dasar yang disimpan sebelumnya. Tes snapshot memungkinkan untuk melakukan refactor secara bebas, karena Anda dapat yakin bahwa kode refactored bekerja persis dengan cara yang sama seperti aslinya. Jika perubahan itu disengaja, Anda dapat menerima baseline baru untuk pengujian masa depan. Namun, AWS CDK peningkatan juga dapat menyebabkan templat yang disintesis berubah, sehingga Anda tidak dapat hanya mengandalkan snapshot untuk memastikan implementasi Anda benar.

Tes unit

Panduan ini berfokus pada integrasi pengujian unit secara TypeScript khusus. Untuk mengaktifkan pengujian, pastikan `package.json` file Anda memiliki pustaka berikut:`@types/jest`,`jest`, dan `ts-jest` didevDependencies. Untuk menambahkan paket-paket ini, jalankan `cdk init lib --language=typescript` perintah. Setelah Anda menjalankan perintah sebelumnya, Anda melihat struktur berikut.



Kode berikut adalah contoh `package.json` file yang diaktifkan dengan pustaka Jest.

```
{
  ...
  "scripts": {
    "build": "npm run lint && tsc",
    "watch": "tsc -w",
    "test": "jest",
  },
  "devDependencies": {
    ...
    "@types/jest": "27.5.2",
    "jest": "27.5.1",
    "ts-jest": "27.1.5",
    ...
  }
}
```

Di bawah folder Uji, Anda dapat menulis kasus uji. Contoh berikut menunjukkan kasus uji untuk AWS CodePipeline konstruksi.

```
import { Stack } from 'aws-cdk-lib';
import { Template } from 'aws-cdk-lib/assertions';
import * as CodePipeline from 'aws-cdk-lib/aws-codepipeline';
import * as CodePipelineActions from 'aws-cdk-lib/aws-codepipeline-actions';
import { MyPipelineStack } from '../lib/my-pipeline-stack';
test('Pipeline Created with GitHub Source', () => {
  // ARRANGE
  const stack = new Stack();
  // ACT
  new MyPipelineStack(stack, 'MyTestStack');
  // ASSERT
  const template = Template.fromStack(stack);
  // Verify that the pipeline resource is created
```

```
  template.resourceCountIs('AWS::CodePipeline::Pipeline', 1);
  // Verify that the pipeline has the expected stages with GitHub source
  template.hasResourceProperties('AWS::CodePipeline::Pipeline', {
    Stages: [
      {
        Name: 'Source',
        Actions: [
          {
            Name: 'SourceAction',
            ActionTypeId: {
              Category: 'Source',
              Owner: 'ThirdParty',
              Provider: 'GitHub',
              Version: '1'
            },
            Configuration: {
              Owner: {
                'Fn::Join': [
                  '',
                  [
                    '{{resolve:secretsmanager:',
                    {
                      Ref: 'GitHubTokenSecret'
                    },
                    ':SecretString:owner}}'
                  ]
                ]
              },
              Repo: {
                'Fn::Join': [
                  '',
                  [
                    '{{resolve:secretsmanager:',
                    {
                      Ref: 'GitHubTokenSecret'
                    },
                    ':SecretString:repo}}'
                  ]
                ]
              },
              Branch: 'main',
              OAuthToken: {
                'Fn::Join': [
```

```
                '',
                [
                  '{{resolve:secretsmanager:',
                  {
                    Ref: 'GitHubTokenSecret'
                  },
                  ':SecretString:token}}'
                ]
              ]
            }
          },
          OutputArtifacts: [
            {
              Name: 'SourceOutput'
            }
          ],
          RunOrder: 1
        }
      ]
    },
    {
      Name: 'Build',
      Actions: [
        {
          Name: 'BuildAction',
          ActionTypeId: {
            Category: 'Build',
            Owner: 'AWS',
            Provider: 'CodeBuild',
            Version: '1'
          },
          InputArtifacts: [
            {
              Name: 'SourceOutput'
            }
          ],
          OutputArtifacts: [
            {
              Name: 'BuildOutput'
            }
          ],
          RunOrder: 1
        }
```

```
        ]
      }
      // Add more stage checks as needed
    ]
  });
  // Verify that a GitHub token secret is created
  template.resourceCountIs('AWS::SecretsManager::Secret', 1);
});
);
```

Untuk menjalankan pengujian, jalankan `npm run test` perintah dalam proyek Anda. Tes mengembalikan hasil berikut.

```
PASS  test/codepipeline-module.test.ts (5.972 s)
  # Code Pipeline Created (97 ms)
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        6.142 s, estimated 9 s
```

Untuk informasi selengkapnya tentang kasus pengujian, lihat Menguji konstruksi di Panduan AWS Cloud Development Kit (AWS CDK) Pengembang.

Tes integrasi

Tes integrasi untuk AWS CDK konstruksi juga dapat disertakan dengan menggunakan `integ-tests` modul. Tes integrasi harus didefinisikan sebagai AWS CDK aplikasi. Harus ada hubungan satu-ke-satu antara tes integrasi dan aplikasi AWS CDK . Untuk informasi selengkapnya, kunjungi modul integ-tests-alpha di Referensi API.AWS CDK

Gunakan rilis dan kontrol versi untuk konstruksi

Kontrol versi untuk AWS CDK

AWS CDK konstruksi umum dapat dibuat oleh beberapa tim dan dibagikan di seluruh organisasi untuk konsumsi. Biasanya, pengembang merilis fitur baru atau perbaikan bug dalam AWS CDK konstruksi umum mereka. Konstruksi ini digunakan oleh AWS CDK aplikasi atau AWS CDK konstruksi lain yang ada sebagai bagian dari ketergantungan. Untuk alasan ini, sangat penting

bahwa pengembang memperbarui dan merilis konstruksi mereka dengan versi semantik yang tepat secara independen. AWS CDK Aplikasi hilir atau AWS CDK konstruksi lain dapat memperbarui ketergantungannya untuk menggunakan versi konstruksi yang baru dirilis AWS CDK .

Versi semantik (Semver) adalah seperangkat aturan, atau metode, untuk menyediakan nomor perangkat lunak unik untuk perangkat lunak komputer. Versi didefinisikan sebagai berikut:

- Versi MAJOR terdiri dari perubahan API yang tidak kompatibel atau perubahan yang melanggar.
- Versi MINOR terdiri dari fungsionalitas yang ditambahkan dengan cara yang kompatibel ke belakang.
- Versi PATCH terdiri dari perbaikan bug yang kompatibel ke belakang.

Untuk informasi lebih lanjut tentang pembuatan versi semantik, lihat Spesifikasi Versi Semantik () dalam dokumentasi Pembuatan Versi Semantik. SemVer

Repositori dan kemasan untuk AWS CDK membangun

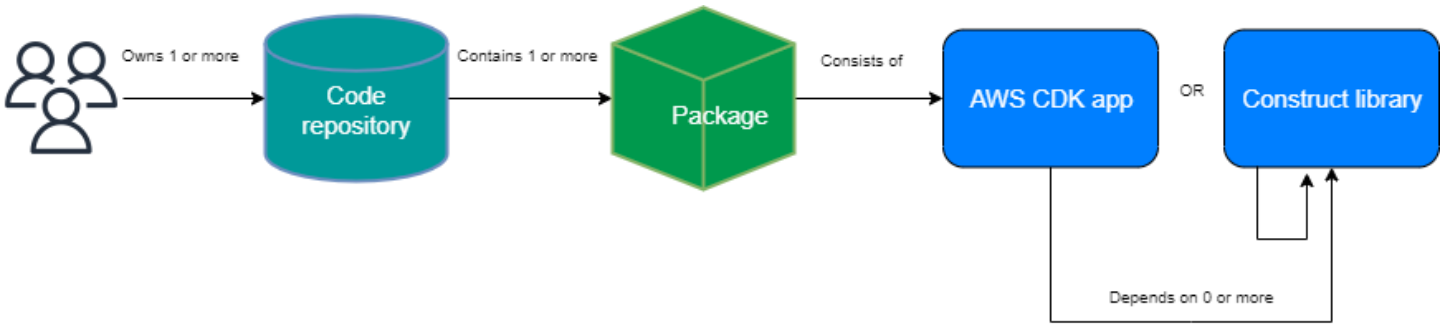
Karena AWS CDK konstruksi dikembangkan oleh tim yang berbeda dan digunakan oleh beberapa AWS CDK aplikasi, Anda dapat menggunakan repositori terpisah untuk setiap konstruksi. AWS CDK Ini juga dapat membantu Anda menegakkan kontrol akses. Setiap repositori dapat berisi semua kode sumber yang terkait dengan AWS CDK konstruksi yang sama bersama dengan semua dependensinya. Dengan menyimpan satu aplikasi (yaitu, sebuah AWS CDK konstruksi) dalam satu repositori, Anda dapat mengurangi cakupan dampak perubahan selama penerapan.

Ini AWS CDK tidak hanya menghasilkan CloudFormation template untuk menerapkan infrastruktur, tetapi juga menggabungkan aset runtime seperti fungsi Lambda dan gambar Docker dan menerapkannya di samping infrastruktur Anda. Tidak hanya mungkin untuk menggabungkan kode yang mendefinisikan infrastruktur Anda dan kode yang mengimplementasikan logika runtime Anda ke dalam satu konstruksi — ini adalah praktik terbaik. Kedua jenis kode ini tidak perlu tinggal di repositori terpisah atau bahkan dalam paket terpisah.

Untuk menggunakan paket di seluruh batas repositori, Anda harus memiliki repositori paket pribadi—mirip dengan npm,, atau Maven Central PyPi, tetapi internal organisasi Anda. Anda juga harus memiliki proses rilis yang membangun, menguji, dan menerbitkan paket ke repositori paket pribadi. Anda dapat membuat repositori pribadi, seperti PyPi server, dengan menggunakan mesin virtual lokal (VM) atau Amazon S3. Saat Anda merancang atau membuat registri paket pribadi, penting untuk mempertimbangkan risiko gangguan layanan karena ketersediaan dan skalabilitas

yang tinggi. Layanan terkelola tanpa server yang dihosting di cloud untuk menyimpan paket dapat sangat mengurangi overhead pemeliharaan. Misalnya, Anda dapat menggunakan AWS CodeArtifactuntuk meng-host paket untuk sebagian besar bahasa pemrograman populer. Anda juga dapat menggunakan CodeArtifact untuk mengatur koneksi repositori eksternal dan mereplikasi mereka di dalamnya. CodeArtifact

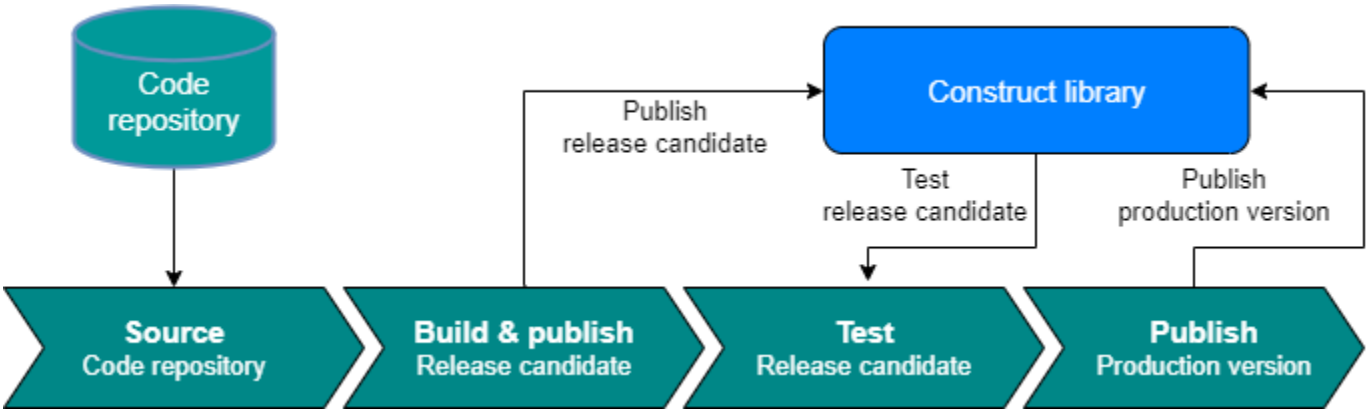
Dependensi pada paket dalam repositori paket dikelola oleh manajer paket bahasa Anda (misalnya, npm untuk atau aplikasi). TypeScript JavaScript Manajer paket Anda memastikan bahwa build dapat diulang dengan merekam versi spesifik dari setiap paket yang bergantung pada aplikasi Anda dan kemudian memungkinkan Anda memutakhirkan dependensi tersebut secara terkontrol, seperti yang ditunjukkan diagram berikut.



Membangun rilis untuk AWS CDK

Kami menyarankan Anda membuat pipeline otomatis Anda sendiri untuk membangun dan merilis versi AWS CDK konstruksi baru. Jika Anda menerapkan proses persetujuan permintaan tarik yang tepat, maka setelah Anda melakukan dan mendorong kode sumber Anda ke cabang utama repositori, pipeline dapat membangun dan membuat versi kandidat rilis. Versi itu dapat didorong CodeArtifact dan diuji sebelum merilis versi siap produksi. Secara opsional, Anda dapat menguji versi AWS CDK konstruksi baru Anda secara lokal sebelum menggabungkan kode dengan cabang utama. Hal ini menyebabkan pipeline merilis versi siap produksi. Mempertimbangkan bahwa konstruksi dan paket bersama harus diuji secara independen dari aplikasi yang dikonsumsi, seolah-olah mereka dirilis ke publik.

Diagram berikut menunjukkan contoh pipa rilis AWS CDK versi.



Anda dapat menggunakan perintah contoh berikut untuk membangun, menguji, dan menerbitkan paket npm. Pertama, masuk ke repositori artefak dengan menjalankan perintah berikut.

```
aws codeartifact login --tool npm --domain <Domain Name> --domain-owner $(aws sts get-
caller-identity --output text --query 'Account')  \
--repository <Repository Name> --region <AWS Region Name>
```

Kemudian, selesaikan langkah-langkah berikut:

1. Instal paket yang diperlukan berdasarkan `package.json` file: `npm install`
2. Buat versi kandidat rilis: `npm version prerelease --preid rc`
3. Bangun paket npm: `npm run build`
4. Uji paket npm: `npm run test`
5. Publikasikan paket npm: `npm publish`

Menegakkan manajemen versi perpustakaan

Manajemen siklus hidup merupakan tantangan yang signifikan ketika Anda mempertahankan basis AWS CDK kode. Misalnya, asumsikan bahwa Anda memulai AWS CDK proyek dengan versi 1.97 dan kemudian versi 1.169 tersedia nanti. Versi 1.169 menawarkan fitur baru dan perbaikan bug, tetapi Anda telah menerapkan infrastruktur Anda dengan menggunakan versi lama. Sekarang, memperbarui konstruksi menjadi menantang karena kesenjangan ini meningkat karena perubahan yang melanggar yang dapat diperkenalkan di versi baru. Ini bisa menjadi tantangan jika Anda memiliki banyak sumber daya di lingkungan Anda. Pola yang diperkenalkan di bagian ini dapat membantu Anda mengelola versi AWS CDK pustaka menggunakan otomatisasi. Berikut alur kerja pola ini:

1. Saat Anda meluncurkan produk CodeArtifact Service Catalog baru, versi AWS CDK library dan dependensinya disimpan dalam file. `package.json`
2. Anda menerapkan pipeline umum yang melacak semua repositori sehingga Anda dapat menerapkan upgrade otomatis ke mereka jika tidak ada perubahan yang melanggar.
3. AWS CodeBuild Tahap memeriksa pohon ketergantungan dan mencari perubahan yang melanggar.
4. Pipeline membuat cabang fitur dan kemudian berjalan `cdk synth` dengan versi baru untuk mengonfirmasi tidak ada kesalahan.
5. Versi baru diterapkan di lingkungan pengujian dan akhirnya menjalankan tes integrasi untuk memastikan penerapannya sehat.
6. Anda dapat menggunakan dua antrian Amazon Simple Queue Service (Amazon SQS) untuk melacak tumpukan. Pengguna dapat meninjau tumpukan secara manual dalam antrian pengecualian dan perubahan pemutusan alamat. Item yang lulus uji integrasi diizinkan untuk digabungkan dan dirilis.

Pertanyaan yang Sering Diajukan

Masalah apa yang bisa TypeScript dipecahkan?

Biasanya, Anda dapat menghilangkan bug dalam kode Anda dengan menulis tes otomatis, memverifikasi secara manual bahwa kode berfungsi seperti yang diharapkan, dan akhirnya meminta orang lain memvalidasi kode Anda. Memvalidasi koneksi antara setiap bagian dari proyek Anda memakan waktu. Untuk mempercepat proses validasi, Anda dapat menggunakan bahasa yang diperiksa tipe seperti TypeScript mengotomatiskan validasi kode dan memberikan umpan balik instan selama pengembangan.

Mengapa saya harus menggunakan TypeScript?

TypeScript adalah bahasa sumber terbuka yang menyederhanakan JavaScript kode, sehingga lebih mudah dibaca dan di-debug. TypeScript juga menyediakan alat pengembangan yang sangat produktif untuk JavaScript IDEs dan praktik, seperti pemeriksaan statis. Selain itu, TypeScript menawarkan manfaat ECMAScript 6 (ES6) dan dapat meningkatkan produktivitas Anda. Terakhir, TypeScript dapat membantu Anda menghindari bug menyakitkan yang biasanya dialami pengembang saat menulis JavaScript berdasarkan jenis memeriksa kode.

Haruskah saya menggunakan AWS CDK atau CloudFormation?

Kami menyarankan Anda menggunakan AWS Cloud Development Kit (AWS CDK) alih-alih AWS CloudFormation, jika organisasi Anda memiliki keahlian pengembangan untuk memanfaatkan AWS CDK. Ini karena AWS CDK lebih fleksibel daripada CloudFormation, karena Anda dapat menggunakan bahasa pemrograman dan konsep OOP. Perlu diingat bahwa Anda dapat menggunakan CloudFormation untuk membuat AWS sumber daya secara teratur dan dapat diprediksi. Dalam CloudFormation, sumber daya ditulis dalam file teks dengan menggunakan format JSON atau YAMAL.

Bagaimana jika AWS CDK tidak mendukung yang baru diluncurkan Layanan AWS?

Anda dapat menggunakan penggantian mentah atau sumber daya CloudFormation khusus.

Apa saja bahasa pemrograman berbeda yang didukung oleh AWS CDK?

AWS CDK umumnya tersedia di JavaScript,, Python TypeScript, Java, C #, dan Go (di Pratinjau Pengembang).

Berapa AWS CDK biayanya?

Tidak ada biaya tambahan untuk AWS CDK. Anda membayar AWS sumber daya (seperti instans Amazon EC2 atau penyeimbang beban Elastic Load Balancing) yang dibuat saat Anda AWS CDK menggunakannya dengan cara yang sama seperti jika Anda membuatnya secara manual. Anda hanya membayar untuk apa yang Anda gunakan, saat Anda menggunakannya. Tidak ada biaya minimum dan tidak ada komitmen dimuka yang diperlukan.

Langkah selanjutnya

Kami menyarankan Anda mulai membangun dengan AWS Cloud Development Kit (AWS CDK) in TypeScript. Untuk informasi lebih lanjut, lihat Workshop AWS CDK Immersion Day.

Sumber daya

Referensi

- AWS Konstruksi Solusi (AWS Solusi)
- AWS Cloud Development Kit (AWS CDK) (GitHub)
- AWS Membangun referensi API Perpustakaan (Dokumentasi AWS CDK Referensi)
- AWS CDK Dokumentasi AWS CDK Referensi (Referensi Dokumentasi)
- AWS CDK Lokakarya Hari Perendaman (Studio AWS Lokakarya)

Alat

- cdk-nag () GitHub
- TypeScript ESLint(TypeScript ESLint dokumentasi)

Panduan dan pola

- AWS Solusi Membangun pola (AWS dokumentasi)

Riwayat dokumen

Tabel berikut menjelaskan perubahan signifikan pada panduan ini. Jika Anda ingin diberi tahu tentang pembaruan masa depan, Anda dapat berlangganan umpan RSS.

Perubahan	Deskripsi	Tanggal
Praktik terbaik yang diperbarui	Kami menghapus rekomendasi untuk menggunakan cfn-nag di bagian Alat pengembangan umum. Di bagian Tentukan konvensi penamaan standar, kami menambahkan konvensi penamaan standar untuk konstanta global dan menambahkan contoh penamaan.	Oktober 23, 2025
Perbarui kode	Kami memperbarui contoh kode di bagian Ikuti praktik TypeScript terbaik.	Februari 16, 2024
Tambahkan bagian	Kami menambahkan bagian Use utility types dan Integration test.	10 Januari 2024
Pembaruan kecil	Contoh kode yang diperbarui untuk membuat konstruksi L3.	Juni 16, 2023
Publikasi awal	—	Desember 8, 2022

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.