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

AWS Bimbingan Preskriptif

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

Table of Contents

Praktik terbaik untuk menggunakan AWS CDK TypeScript untuk membuat proyek IAc

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

Februari 2024 (riwayat 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

Panduan ini dapat membantu Anda mencapai hasil bisnis yang ditargetkan berikut:

- 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
- Mengadopsi pendekatan pengembangan berbasis tes
- Gunakan rilis dan kontrol versi untuk konstruksi
- Menegakkan manajemen versi pustaka

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.

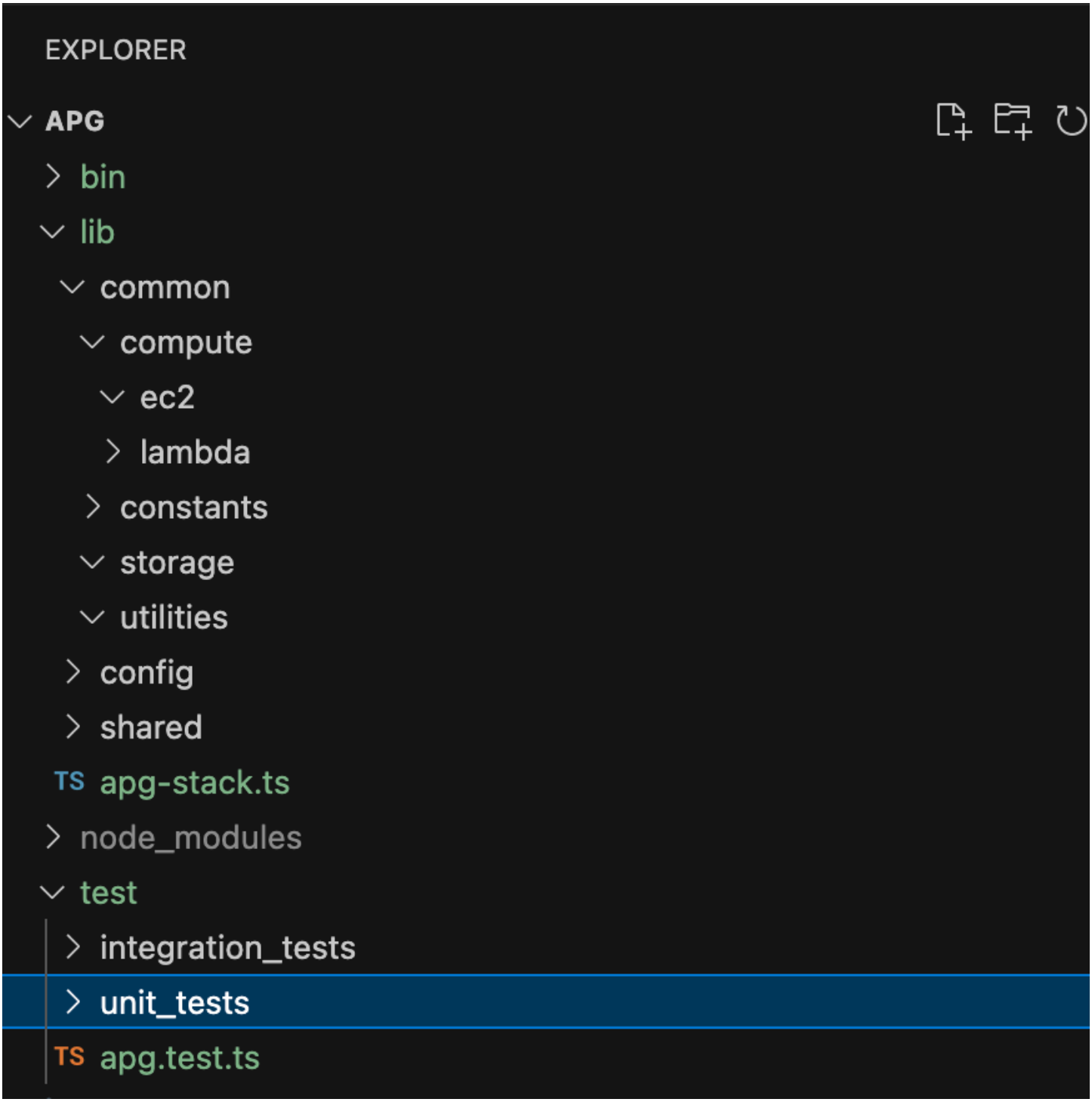
Cara mengatur kode Anda untuk skala

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 dokumentasi Refactoring.Guru.

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 di TypeScript dalam dokumentasi Refactoring.Guru.

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 dalam dokumentasi Refactoring.Guru.

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. Pustaka 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 LifeCycle Rule (), 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-. ApplicationLoadBalancedFargateService layanan merupakan arsitektur yang mencakup cluster AWS Fargate kontainer yang menggunakan Application Load Balancer. Contoh lain adalah aws-apigateway. LambdaRestKonstruksi api. 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 konstruk.

```
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
  }
}
```

Membuat atau memperluas konstruksi L2

Konstruksi L2 mewakili “komponen cloud” dan merangkum semua 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 mengakses `node.defaultChild` properti pada `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 konstruksi AwsCustomResource. 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 diketik dengan kuat 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: "my-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: "my-bucket",
    region: "us-east-1",
    encryption: false,
});

const myS3Bucket2 = new S3Bucket({
    name: "my-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 di TypeScript dalam dokumentasi Refactoring.Guru.

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 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.tsb`

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.

Sebagian <Type>

`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> = {};
```

Diperlukan <Type>

`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 yang umum dapat mencakup hal-hal berikut:

- Hak istimewa yang terlalu permisif AWS Identity and Access Management (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 cfn-nag untuk mengidentifikasi masalah keamanan infrastruktur, seperti aturan IAM permisif atau literal kata sandi, dalam templat. CloudFormation Untuk informasi lebih lanjut, lihat repositori GitHub cfn-nag dari Stelligent.
- Gunakan cdk-nag, terinspirasi oleh cfn-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 Amazon CodeWhisperer untuk melakukan pemindaian keamanan. CodeWhisperer adalah alat produktivitas bertenaga AI yang dapat menghasilkan saran kode secara real time. Ini dapat membantu Anda mengidentifikasi masalah keamanan, mengikuti praktik terbaik, dan meningkatkan produktivitas untuk penerapan 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. Tidak jarang pengembang 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 konstruksi

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 Construct Library

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"
  },
```

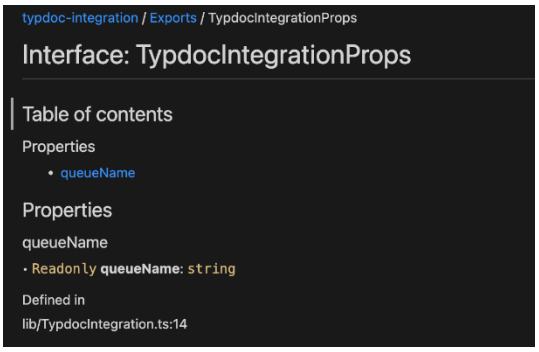
```
}
```

`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.

Mengadopsi pendekatan pengembangan berbasis tes

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.

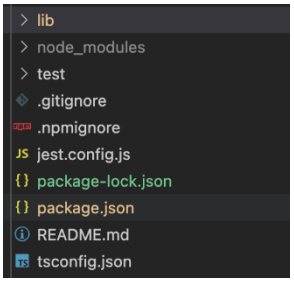
Anda dapat menggunakan TDD untuk menulis kasus uji terlebih dahulu. Ini membantu Anda memvalidasi infrastruktur dengan kendala desain yang berbeda dalam hal menegakkan kebijakan keamanan untuk sumber daya dan mengikuti konvensi penamaan unik untuk proyek. Pendekatan standar untuk menguji AWS CDK aplikasi 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). Pernyataan berbutir halus 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 {App,Stack} from 'aws-cdk-lib';
import { Template } from 'aws-cdk-lib/assertions';
import * as CodepipelineModule from '../lib/index';
import { Role, ServicePrincipal } from 'aws-cdk-lib/aws-iam';
import { Repository } from 'aws-cdk-lib/aws-codecommit';
import { PipelineProject } from 'aws-cdk-lib/aws-codebuild';

const testData:CodepipelineModule.CodepipelineModuleProps = {

    pipelineName: "validate-test-pipeline",
    serviceRoleARN: "",
    codeCommitRepositoryARN: "",
    branchName: "master",
    buildStages:[]
}


test('Code Pipeline Created', () => {
    const app = new App();
    const stack = new Stack(app, "TestStack");
    // WHEN
    const serviceRole = new Role(stack, "testRole", { assumedBy: new
 ServicePrincipal('codepipeline.amazonaws.com') })
    const codeCommit = new Repository(stack, "testRepo", {
        repositoryName: "validate-codeCommit-repo"
    });
    const codeBuildProject=new PipelineProject(stack,"TestCodeBuildProject",{});
    testData.serviceRoleARN = serviceRole.roleArn;
    testData.codeCommitRepositoryARN = codeCommit.repositoryArn;
    testData["buildStages"].push({
```

```
        stageName:"Deploy",
        codeBuildProject:codeBuildProject
    })
    new CodepipelineModule.CodepipelineModule(stack, 'MyTestConstruct', testData);
    // THEN
    const template = Template.fromStack(stack);

    template.hasResourceProperties('AWS::CodePipeline::Pipeline', {
        Name:testData.pipelineName
    });
});
```

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 one-to-one hubungan antara tes integrasi dan AWS CDK aplikasi. Untuk informasi selengkapnya, kunjungi integ-tests-alpha modul di Referensi AWS CDK API.

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 konstruksi AWS CDK

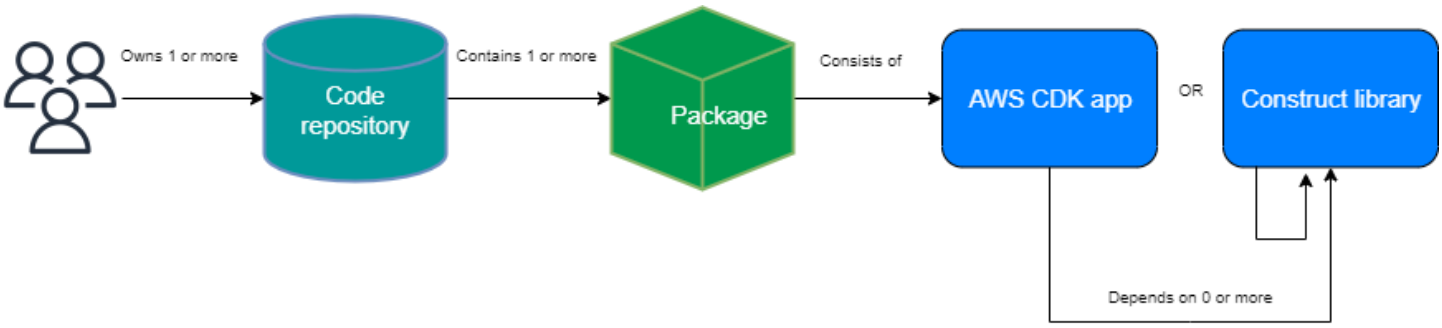
Karena AWS CDK konstruksi dikembangkan oleh tim yang berbeda dan digunakan oleh beberapa AWS CDK aplikasi, Anda dapat menggunakan repositori terpisah untuk setiap konstruksi. AWS CDK Ini juga dapat membantu Anda menegakkan kontrol akses. Setiap repositori dapat berisi semua kode sumber yang terkait dengan 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 di-host di cloud untuk menyimpan paket dapat sangat mengurangi biaya 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

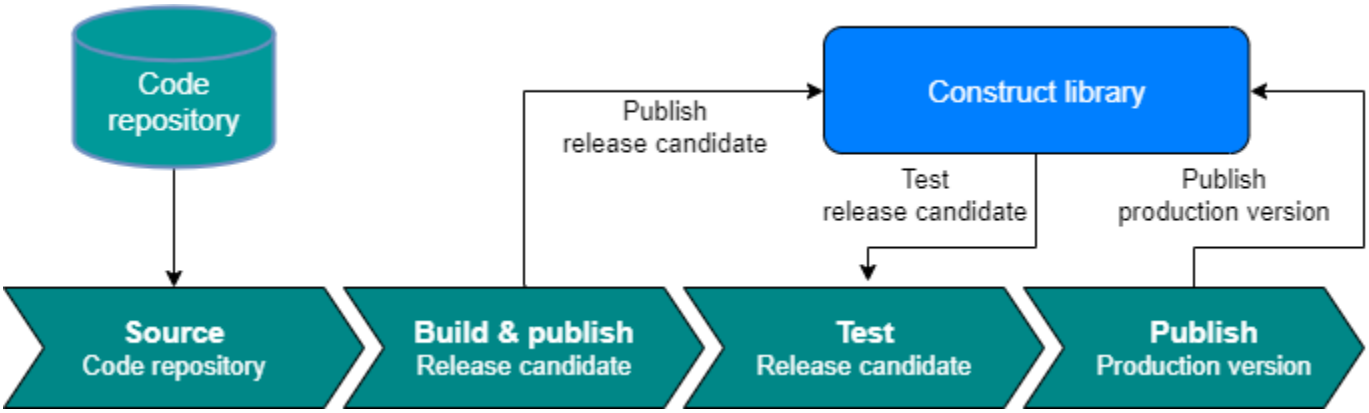
Dependensi pada paket dalam repositori paket dikelola oleh manajer paket bahasa Anda (misalnya, npm untuk atau aplikasi). TypeScript JavaScript Manajer paket Anda memastikan bahwa build dapat diulang dengan merekam versi spesifik dari setiap paket yang bergantung pada aplikasi Anda dan kemudian memungkinkan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



Bangun 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 IDE 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-alihAWS CloudFormation, jika organisasi Anda memiliki keahlian pengembangan untuk memanfaatkanAWS 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 diluncurkanLayanan AWS?

Anda dapat menggunakan penggantian mentah atau sumber daya CloudFormation khusus.

Apa saja bahasa pemrograman berbeda yang didukung olehAWS CDK?

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

Berapa AWS CDK biayanya?

Tidak ada biaya tambahan untukAWS 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 Kit Pengembangan Cloud (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 (dokumentasi TypeScript ESLint)

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
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

AWS Glosarium Panduan Preskriptif

Berikut ini adalah istilah yang umum digunakan dalam strategi, panduan, dan pola yang disediakan oleh Panduan AWS Preskriptif. Untuk menyarankan entri, silakan gunakan tautan Berikan umpan balik di akhir glosarium.

Nomor

7 Rs

Tujuh strategi migrasi umum untuk memindahkan aplikasi ke cloud. Strategi ini dibangun di atas 5 Rs yang diidentifikasi Gartner pada tahun 2011 dan terdiri dari yang berikut:

- Refactor/Re-Architect — Memindahkan aplikasi dan memodifikasi arsitekturnya dengan memanfaatkan sepenuhnya fitur cloud-native untuk meningkatkan kelincahan, kinerja, dan skalabilitas. Ini biasanya melibatkan porting sistem operasi dan database. Contoh: Migrasikan database Oracle lokal Anda ke Amazon Aurora PostgreSQL Compatible Edition.
- Replatform (angkat dan bentuk ulang) — Pindahkan aplikasi ke cloud, dan perkenalkan beberapa tingkat pengoptimalan untuk memanfaatkan kemampuan cloud. Contoh: Migrasikan database Oracle lokal Anda ke Amazon Relational Database Service (Amazon RDS) untuk Oracle di Cloud. AWS
- Pembelian kembali (drop and shop) - Beralih ke produk yang berbeda, biasanya dengan beralih dari lisensi tradisional ke model SaaS. Contoh: Migrasikan sistem manajemen hubungan pelanggan (CRM) Anda ke Salesforce.com.
- Rehost (lift dan shift) — Pindahkan aplikasi ke cloud tanpa membuat perubahan apa pun untuk memanfaatkan kemampuan cloud. Contoh: Migrasikan database Oracle lokal Anda ke Oracle pada instans EC2 di Cloud. AWS
- Relokasi (hypervisor-level lift and shift) — Pindahkan infrastruktur ke cloud tanpa membeli perangkat keras baru, menulis ulang aplikasi, atau memodifikasi operasi yang ada. Skenario migrasi ini khusus untuk VMware Cloud on AWS, yang mendukung kompatibilitas mesin virtual (VM) dan portabilitas beban kerja antara lingkungan lokal Anda dan. AWS Anda dapat menggunakan teknologi VMware Cloud Foundation dari pusat data lokal saat memigrasikan infrastruktur ke VMware Cloud. AWS Contoh: Pindahkan hypervisor yang menghosting database Oracle Anda ke VMware Cloud on. AWS
- Pertahankan (kunjungi kembali) - Simpan aplikasi di lingkungan sumber Anda. Ini mungkin termasuk aplikasi yang memerlukan refactoring besar, dan Anda ingin menunda pekerjaan itu

sampai nanti, dan aplikasi lama yang ingin Anda pertahankan, karena tidak ada pembenaran bisnis untuk memigrasikannya.

- Pensiun — Menonaktifkan atau menghapus aplikasi yang tidak lagi diperlukan di lingkungan sumber Anda.

A

ABAC

Lihat kontrol akses berbasis atribut.

layanan abstrak

Lihat layanan terkelola.

ASAM

Lihat atomisitas, konsistensi, isolasi, daya tahan.

migrasi aktif-aktif

Metode migrasi database di mana basis data sumber dan target tetap sinkron (dengan menggunakan alat replikasi dua arah atau operasi penulisan ganda), dan kedua database menangani transaksi dari menghubungkan aplikasi selama migrasi. Metode ini mendukung migrasi dalam batch kecil yang terkontrol alih-alih memerlukan pemotongan satu kali. Ini lebih fleksibel tetapi membutuhkan lebih banyak pekerjaan daripada migrasi aktif-pasif.

migrasi aktif-pasif

Metode migrasi database di mana database sumber dan target disimpan dalam sinkron, tetapi hanya database sumber yang menangani transaksi dari menghubungkan aplikasi sementara data direplikasi ke database target. Basis data target tidak menerima transaksi apa pun selama migrasi.

fungsi agregat

Fungsi SQL yang beroperasi pada sekelompok baris dan menghitung nilai pengembalian tunggal untuk grup. Contoh fungsi agregat meliputi `SUM` dan`MAX`.

AI

Lihat kecerdasan buatan.

AIOps

Lihat operasi kecerdasan buatan.

anonimisasi

Proses menghapus informasi pribadi secara permanen dalam kumpulan data. Anonimisasi dapat membantu melindungi privasi pribadi. Data anonim tidak lagi dianggap sebagai data pribadi.

anti-pola

Solusi yang sering digunakan untuk masalah berulang di mana solusinya kontra-produktif, tidak efektif, atau kurang efektif daripada alternatif.

kontrol aplikasi

Pendekatan keamanan yang memungkinkan penggunaan hanya aplikasi yang disetujui untuk membantu melindungi sistem dari malware.

portofolio aplikasi

Kumpulan informasi rinci tentang setiap aplikasi yang digunakan oleh organisasi, termasuk biaya untuk membangun dan memelihara aplikasi, dan nilai bisnisnya. Informasi ini adalah kunci untuk penemuan portofolio dan proses analisis dan membantu mengidentifikasi dan memprioritaskan aplikasi yang akan dimigrasi, dimodernisasi, dan dioptimalkan.

kecerdasan buatan (AI)

Bidang ilmu komputer yang didedikasikan untuk menggunakan teknologi komputasi untuk melakukan fungsi kognitif yang biasanya terkait dengan manusia, seperti belajar, memecahkan masalah, dan mengenali pola. Untuk informasi lebih lanjut, lihat Apa itu Kecerdasan Buatan?

operasi kecerdasan buatan (AIOps)

Proses menggunakan teknik pembelajaran mesin untuk memecahkan masalah operasional, mengurangi insiden operasional dan intervensi manusia, dan meningkatkan kualitas layanan. Untuk informasi selengkapnya tentang cara AIOps digunakan dalam strategi AWS migrasi, lihat panduan integrasi operasi.

enkripsi asimetris

Algoritma enkripsi yang menggunakan sepasang kunci, kunci publik untuk enkripsi dan kunci pribadi untuk dekripsi. Anda dapat berbagi kunci publik karena tidak digunakan untuk dekripsi, tetapi akses ke kunci pribadi harus sangat dibatasi.

atomisitas, konsistensi, isolasi, daya tahan (ACID)

Satu set properti perangkat lunak yang menjamin validitas data dan keandalan operasional database, bahkan dalam kasus kesalahan, kegagalan daya, atau masalah lainnya.

kontrol akses berbasis atribut (ABAC)

Praktik membuat izin berbutir halus berdasarkan atribut pengguna, seperti departemen, peran pekerjaan, dan nama tim. Untuk informasi selengkapnya, lihat ABAC untuk AWS dokumentasi AWS Identity and Access Management (IAM).

sumber data otoritatif

Lokasi di mana Anda menyimpan versi utama data, yang dianggap sebagai sumber informasi yang paling dapat diandalkan. Anda dapat menyalin data dari sumber data otoritatif ke lokasi lain untuk tujuan memproses atau memodifikasi data, seperti menganonimkan, menyunting, atau membuat nama samaran.

Zona Ketersediaan

Lokasi berbeda di dalam Wilayah AWS yang terisolasi dari kegagalan di Availability Zone lainnya dan menyediakan konektivitas jaringan latensi rendah yang murah ke Availability Zone lainnya di Wilayah yang sama.

AWS Kerangka Adopsi Cloud (AWS CAF)

Kerangka pedoman dan praktik terbaik AWS untuk membantu organisasi mengembangkan rencana yang efisien dan efektif untuk bergerak dengan sukses ke cloud. AWS CAF mengatur panduan ke dalam enam area fokus yang disebut perspektif: bisnis, orang, tata kelola, platform, keamanan, dan operasi. Perspektif bisnis, orang, dan tata kelola fokus pada keterampilan dan proses bisnis; perspektif platform, keamanan, dan operasi fokus pada keterampilan dan proses teknis. Misalnya, perspektif masyarakat menargetkan pemangku kepentingan yang menangani sumber daya manusia (SDM), fungsi kepegawaian, dan manajemen orang. Untuk perspektif ini, AWS CAF memberikan panduan untuk pengembangan, pelatihan, dan komunikasi orang untuk membantu mempersiapkan organisasi untuk adopsi cloud yang sukses. Untuk informasi lebih lanjut, lihat situs web AWS CAF dan whitepaper AWS CAF.

AWS Kerangka Kualifikasi Beban Kerja (AWS WQF)

Alat yang mengevaluasi beban kerja migrasi database, merekomendasikan strategi migrasi, dan memberikan perkiraan kerja. AWS WQF disertakan dengan AWS Schema Conversion Tool ()AWS SCT. Ini menganalisis skema database dan objek kode, kode aplikasi, dependensi, dan karakteristik kinerja, dan memberikan laporan penilaian.

B

bot buruk

Bot yang dimaksudkan untuk mengganggu atau membahayakan individu atau organisasi.

BCP

Lihat perencanaan kontinuitas bisnis.

grafik perilaku

Pandangan interaktif yang terpadu tentang perilaku dan interaksi sumber daya dari waktu ke waktu. Anda dapat menggunakan grafik perilaku dengan Amazon Detective untuk memeriksa upaya logon yang gagal, panggilan API yang mencurigakan, dan tindakan serupa. Untuk informasi selengkapnya, lihat Data dalam grafik perilaku di dokumentasi Detektif.

sistem big-endian

Sistem yang menyimpan byte paling signifikan terlebih dahulu. Lihat juga endianness.

klasifikasi biner

Sebuah proses yang memprediksi hasil biner (salah satu dari dua kelas yang mungkin). Misalnya, model ML Anda mungkin perlu memprediksi masalah seperti “Apakah email ini spam atau bukan spam?” atau “Apakah produk ini buku atau mobil?”

filter mekar

Struktur data probabilistik dan efisien memori yang digunakan untuk menguji apakah suatu elemen adalah anggota dari suatu himpunan.

deployment biru/hijau

Strategi penyebaran tempat Anda membuat dua lingkungan yang terpisah namun identik. Anda menjalankan versi aplikasi saat ini di satu lingkungan (biru) dan versi aplikasi baru di lingkungan lain (hijau). Strategi ini membantu Anda dengan cepat memutar kembali dengan dampak minimal.

bot

Aplikasi perangkat lunak yang menjalankan tugas otomatis melalui internet dan mensimulasikan aktivitas atau interaksi manusia. Beberapa bot berguna atau bermanfaat, seperti perayap web yang mengindeks informasi di internet. Beberapa bot lain, yang dikenal sebagai bot buruk, dimaksudkan untuk mengganggu atau membahayakan individu atau organisasi.

botnet

Jaringan bot yang terinfeksi oleh malware dan berada di bawah kendali satu pihak, yang dikenal sebagai bot herder atau operator bot. Botnet adalah mekanisme paling terkenal untuk skala bot dan dampaknya.

cabang

Area berisi repositori kode. Cabang pertama yang dibuat dalam repositori adalah cabang utama. Anda dapat membuat cabang baru dari cabang yang ada, dan Anda kemudian dapat mengembangkan fitur atau memperbaiki bug di cabang baru. Cabang yang Anda buat untuk membangun fitur biasanya disebut sebagai cabang fitur. Saat fitur siap dirilis, Anda menggabungkan cabang fitur kembali ke cabang utama. Untuk informasi selengkapnya, lihat Tentang cabang (GitHub dokumentasi).

akses break-glass

Dalam keadaan luar biasa dan melalui proses yang disetujui, cara cepat bagi pengguna untuk mendapatkan akses ke Akun AWS yang biasanya tidak memiliki izin untuk mengaksesnya. Untuk informasi lebih lanjut, lihat indikator Implementasikan prosedur break-glass dalam panduan Well-Architected AWS .

strategi brownfield

Infrastruktur yang ada di lingkungan Anda. Saat mengadopsi strategi brownfield untuk arsitektur sistem, Anda merancang arsitektur di sekitar kendala sistem dan infrastruktur saat ini. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

cache penyangga

Area memori tempat data yang paling sering diakses disimpan.

kemampuan bisnis

Apa yang dilakukan bisnis untuk menghasilkan nilai (misalnya, penjualan, layanan pelanggan, atau pemasaran). Arsitektur layanan mikro dan keputusan pengembangan dapat didorong oleh kemampuan bisnis. Untuk informasi selengkapnya, lihat bagian Terorganisir di sekitar kemampuan bisnis dari Menjalankan layanan mikro kontainer di whitepaper. AWS

perencanaan kelangsungan bisnis (BCP)

Rencana yang membahas dampak potensial dari peristiwa yang mengganggu, seperti migrasi skala besar, pada operasi dan memungkinkan bisnis untuk melanjutkan operasi dengan cepat.

C

KAFE

Lihat Kerangka Adopsi AWS Cloud.

penyebaran kenari

Rilis versi yang lambat dan bertahap untuk pengguna akhir. Ketika Anda yakin, Anda menyebarkan versi baru dan mengganti versi saat ini secara keseluruhan.

CCoE

Lihat Cloud Center of Excellence.

CDC

Lihat mengubah pengambilan data.

ubah pengambilan data (CDC)

Proses melacak perubahan ke sumber data, seperti tabel database, dan merekam metadata tentang perubahan tersebut. Anda dapat menggunakan CDC untuk berbagai tujuan, seperti mengaudit atau mereplikasi perubahan dalam sistem target untuk mempertahankan sinkronisasi.

rekayasa kekacauan

Sengaja memperkenalkan kegagalan atau peristiwa yang mengganggu untuk menguji ketahanan sistem. Anda dapat menggunakan AWS Fault Injection Service (AWS FIS) untuk melakukan eksperimen yang menekankan AWS beban kerja Anda dan mengevaluasi responsnya.

CI/CD

Lihat integrasi berkelanjutan dan pengiriman berkelanjutan.

klasifikasi

Proses kategorisasi yang membantu menghasilkan prediksi. Model ML untuk masalah klasifikasi memprediksi nilai diskrit. Nilai diskrit selalu berbeda satu sama lain. Misalnya, model mungkin perlu mengevaluasi apakah ada mobil dalam gambar atau tidak.

Enkripsi sisi klien

Enkripsi data secara lokal, sebelum target Layanan AWS menerimanya.

Cloud Center of Excellence (CCoE)

Tim multi-disiplin yang mendorong upaya adopsi cloud di seluruh organisasi, termasuk mengembangkan praktik terbaik cloud, memobilisasi sumber daya, menetapkan jadwal migrasi, dan memimpin organisasi melalui transformasi skala besar. Untuk informasi selengkapnya, lihat posting CCoE di Blog Strategi AWS Cloud Enterprise.

komputasi cloud

Teknologi cloud yang biasanya digunakan untuk penyimpanan data jarak jauh dan manajemen perangkat IoT. Cloud computing umumnya terhubung ke teknologi edge computing.

model operasi cloud

Dalam organisasi TI, model operasi yang digunakan untuk membangun, mematangkan, dan mengoptimalkan satu atau lebih lingkungan cloud. Untuk informasi selengkapnya, lihat Membangun Model Operasi Cloud Anda.

tahap adopsi cloud

Empat fase yang biasanya dilalui organisasi ketika mereka bermigrasi ke AWS Cloud:

- Proyek — Menjalankan beberapa proyek terkait cloud untuk bukti konsep dan tujuan pembelajaran
- Foundation — Melakukan investasi dasar untuk meningkatkan adopsi cloud Anda (misalnya, membuat landing zone, mendefinisikan CCoE, membuat model operasi)
- Migrasi — Migrasi aplikasi individual
- Re-invention — Mengoptimalkan produk dan layanan, dan berinovasi di cloud

Tahapan ini didefinisikan oleh Stephen Orban dalam posting blog The Journey Toward Cloud-First & the Stages of Adoption di blog AWS Cloud Enterprise Strategy. Untuk informasi tentang bagaimana kaitannya dengan strategi AWS migrasi, lihat panduan kesiapan migrasi.

CMDB

Lihat database manajemen konfigurasi.

repositori kode

Lokasi di mana kode sumber dan aset lainnya, seperti dokumentasi, sampel, dan skrip, disimpan dan diperbarui melalui proses kontrol versi. Repositori cloud umum termasuk GitHub atau. AWS CodeCommit Setiap versi kode disebut cabang. Dalam struktur layanan mikro, setiap repositori

dikhususkan untuk satu bagian fungsionalitas. Pipa CI/CD tunggal dapat menggunakan beberapa repositori.

cache dingin

Cache buffer yang kosong, tidak terisi dengan baik, atau berisi data basi atau tidak relevan. Ini mempengaruhi kinerja karena instance database harus membaca dari memori utama atau disk, yang lebih lambat daripada membaca dari cache buffer.

data dingin

Data yang jarang diakses dan biasanya historis. Saat menanyakan jenis data ini, kueri lambat biasanya dapat diterima. Memindahkan data ini ke tingkat atau kelas penyimpanan yang berkinerja lebih rendah dan lebih murah dapat mengurangi biaya.

visi komputer (CV)

Bidang AI yang menggunakan pembelajaran mesin untuk menganalisis dan mengekstrak informasi dari format visual seperti gambar dan video digital. Misalnya, AWS Panorama menawarkan perangkat yang menambahkan CV ke jaringan kamera lokal, dan Amazon SageMaker menyediakan algoritme pemrosesan gambar untuk CV.

konfigurasi drift

Untuk beban kerja, konfigurasi berubah dari status yang diharapkan. Ini dapat menyebabkan beban kerja menjadi tidak patuh, dan biasanya bertahap dan tidak disengaja.

database manajemen konfigurasi (CMDB)

Repositori yang menyimpan dan mengelola informasi tentang database dan lingkungan TI, termasuk komponen perangkat keras dan perangkat lunak dan konfigurasinya. Anda biasanya menggunakan data dari CMDB dalam penemuan portofolio dan tahap analisis migrasi.

paket kesesuaian

Kumpulan AWS Config aturan dan tindakan remediasi yang dapat Anda kumpulkan untuk menyesuaikan kepatuhan dan pemeriksaan keamanan Anda. Anda dapat menerapkan paket kesesuaian sebagai entitas tunggal di Akun AWS dan Region, atau di seluruh organisasi, dengan menggunakan templat YAMM. Untuk informasi selengkapnya, lihat Paket kesesuaian dalam dokumentasi. AWS Config

integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD)

Proses mengotomatiskan sumber, membangun, menguji, pementasan, dan tahap produksi dari proses rilis perangkat lunak. CI/CD umumnya digambarkan sebagai pipa. CI/CD dapat membantu

Anda mengotomatiskan proses, meningkatkan produktivitas, meningkatkan kualitas kode, dan memberikan lebih cepat. Untuk informasi lebih lanjut, lihat Manfaat pengiriman berkelanjutan. CD juga dapat berarti penerapan berkelanjutan. Untuk informasi selengkapnya, lihat Continuous Delivery vs Continuous Deployment.

CV

Lihat visi komputer.

D

data saat istirahat

Data yang stasioner di jaringan Anda, seperti data yang ada di penyimpanan.

klasifikasi data

Proses untuk mengidentifikasi dan mengkategorikan data dalam jaringan Anda berdasarkan kekritisan dan sensitivitasnya. Ini adalah komponen penting dari setiap strategi manajemen risiko keamanan siber karena membantu Anda menentukan perlindungan dan kontrol retensi yang tepat untuk data. Klasifikasi data adalah komponen pilar keamanan dalam AWS Well-Architected Framework. Untuk informasi selengkapnya, lihat Klasifikasi data.

penyimpangan data

Variasi yang berarti antara data produksi dan data yang digunakan untuk melatih model ML, atau perubahan yang berarti dalam data input dari waktu ke waktu. Penyimpangan data dapat mengurangi kualitas, akurasi, dan keadilan keseluruhan dalam prediksi model ML.

data dalam transit

Data yang aktif bergerak melalui jaringan Anda, seperti antara sumber daya jaringan.

jala data

Kerangka arsitektur yang menyediakan kepemilikan data terdistribusi dan terdesentralisasi dengan manajemen dan tata kelola terpusat.

minimalisasi data

Prinsip pengumpulan dan pemrosesan hanya data yang sangat diperlukan. Mempraktikkan minimalisasi data AWS Cloud dapat mengurangi risiko privasi, biaya, dan jejak karbon analitik Anda.

perimeter data

Satu set pagar pembatas pencegahan di AWS lingkungan Anda yang membantu memastikan bahwa hanya identitas tepercaya yang mengakses sumber daya tepercaya dari jaringan yang diharapkan. Untuk informasi selengkapnya, lihat Membangun perimeter data pada AWS.

prapemrosesan data

Untuk mengubah data mentah menjadi format yang mudah diuraikan oleh model ML Anda. Preprocessing data dapat berarti menghapus kolom atau baris tertentu dan menangani nilai yang hilang, tidak konsisten, atau duplikat.

asal data

Proses melacak asal dan riwayat data sepanjang siklus hidupnya, seperti bagaimana data dihasilkan, ditransmisikan, dan disimpan.

subjek data

Individu yang datanya dikumpulkan dan diproses.

gudang data

Sistem manajemen data yang mendukung intelijen bisnis, seperti analitik. Gudang data biasanya berisi sejumlah besar data historis, dan biasanya digunakan untuk kueri dan analisis.

bahasa definisi database (DDL)

Pernyataan atau perintah untuk membuat atau memodifikasi struktur tabel dan objek dalam database.

bahasa manipulasi basis data (DHTML)

Pernyataan atau perintah untuk memodifikasi (memasukkan, memperbarui, dan menghapus) informasi dalam database.

DDL

Lihat bahasa definisi database.

ansambel yang dalam

Untuk menggabungkan beberapa model pembelajaran mendalam untuk prediksi. Anda dapat menggunakan ansambel dalam untuk mendapatkan prediksi yang lebih akurat atau untuk memperkirakan ketidakpastian dalam prediksi.

pembelajaran mendalam

Subbidang ML yang menggunakan beberapa lapisan jaringan saraf tiruan untuk mengidentifikasi pemetaan antara data input dan variabel target yang diinginkan.

defense-in-depth

Pendekatan keamanan informasi di mana serangkaian mekanisme dan kontrol keamanan dilapisi dengan cermat di seluruh jaringan komputer untuk melindungi kerahasiaan, integritas, dan ketersediaan jaringan dan data di dalamnya. Saat Anda mengadopsi strategi ini AWS, Anda menambahkan beberapa kontrol pada lapisan AWS Organizations struktur yang berbeda untuk membantu mengamankan sumber daya. Misalnya, defense-in-depth pendekatan mungkin menggabungkan otentikasi multi-faktor, segmentasi jaringan, dan enkripsi.

administrator yang didelegasikan

Di AWS Organizations, layanan yang kompatibel dapat mendaftarkan akun AWS anggota untuk mengelola akun organisasi dan mengelola izin untuk layanan tersebut. Akun ini disebut administrator yang didelegasikan untuk layanan itu. Untuk informasi selengkapnya dan daftar layanan yang kompatibel, lihat Layanan yang berfungsi dengan AWS Organizations AWS Organizations dokumentasi.

deployment

Proses pembuatan aplikasi, fitur baru, atau perbaikan kode tersedia di lingkungan target. Deployment melibatkan penerapan perubahan dalam basis kode dan kemudian membangun dan menjalankan basis kode itu di lingkungan aplikasi.

lingkungan pengembangan

Lihat lingkungan.

kontrol detektif

Kontrol keamanan yang dirancang untuk mendeteksi, mencatat, dan memperingatkan setelah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan kedua, memperingatkan Anda tentang peristiwa keamanan yang melewati kontrol pencegahan di tempat. Untuk informasi selengkapnya, lihat Kontrol Detektif dalam Menerapkan kontrol keamanan pada. AWS

pemetaan aliran nilai pengembangan (DVSM)

Sebuah proses yang digunakan untuk mengidentifikasi dan memprioritaskan kendala yang mempengaruhi kecepatan dan kualitas dalam siklus hidup pengembangan perangkat lunak. DVSM memperluas proses pemetaan aliran nilai yang awalnya dirancang untuk praktik

manufaktur ramping. Ini berfokus pada langkah-langkah dan tim yang diperlukan untuk menciptakan dan memindahkan nilai melalui proses pengembangan perangkat lunak.

kembar digital

Representasi virtual dari sistem dunia nyata, seperti bangunan, pabrik, peralatan industri, atau jalur produksi. Kembar digital mendukung pemeliharaan prediktif, pemantauan jarak jauh, dan optimalisasi produksi.

tabel dimensi

Dalam skema bintang, tabel yang lebih kecil yang berisi atribut data tentang data kuantitatif dalam tabel fakta. Atribut tabel dimensi biasanya bidang teks atau angka diskrit yang berperilaku seperti teks. Atribut ini biasanya digunakan untuk pembatasan kueri, pemfilteran, dan pelabelan set hasil.

musibah

Peristiwa yang mencegah beban kerja atau sistem memenuhi tujuan bisnisnya di lokasi utama yang digunakan. Peristiwa ini dapat berupa bencana alam, kegagalan teknis, atau akibat dari tindakan manusia, seperti kesalahan konfigurasi yang tidak disengaja atau serangan malware.

pemulihan bencana (DR)

Strategi dan proses yang Anda gunakan untuk meminimalkan downtime dan kehilangan data yang disebabkan oleh bencana. Untuk informasi selengkapnya, lihat Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML~

Lihat bahasa manipulasi database.

desain berbasis domain

Pendekatan untuk mengembangkan sistem perangkat lunak yang kompleks dengan menghubungkan komponennya ke domain yang berkembang, atau tujuan bisnis inti, yang dilayani oleh setiap komponen. Konsep ini diperkenalkan oleh Eric Evans dalam bukunya, Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston: Addison-Wesley Professional, 2003). Untuk informasi tentang cara menggunakan desain berbasis domain dengan pola gambar pencekik, lihat Memodernisasi layanan web Microsoft ASP.NET (ASMX) lama secara bertahap menggunakan container dan Amazon API Gateway.

DR

Lihat pemulihan bencana.

deteksi drift

Melacak penyimpangan dari konfigurasi dasar. Misalnya, Anda dapat menggunakan AWS CloudFormation untuk mendeteksi penyimpangan dalam sumber daya sistem, atau Anda dapat menggunakannya AWS Control Tower untuk mendeteksi perubahan di landing zone yang mungkin memengaruhi kepatuhan terhadap persyaratan tata kelola.

DVSM

Lihat pemetaan aliran nilai pengembangan.

E

EDA

Lihat analisis data eksplorasi.

komputasi tepi

Teknologi yang meningkatkan daya komputasi untuk perangkat pintar di tepi jaringan IoT. Jika dibandingkan dengan komputasi awan, komputasi tepi dapat mengurangi latensi komunikasi dan meningkatkan waktu respons.

enkripsi

Proses komputasi yang mengubah data plaintext, yang dapat dibaca manusia, menjadi ciphertext.

kunci enkripsi

String kriptografi dari bit acak yang dihasilkan oleh algoritma enkripsi. Panjang kunci dapat bervariasi, dan setiap kunci dirancang agar tidak dapat diprediksi dan unik.

endianness

Urutan byte disimpan dalam memori komputer. Sistem big-endian menyimpan byte paling signifikan terlebih dahulu. Sistem little-endian menyimpan byte paling tidak signifikan terlebih dahulu.

titik akhir

Lihat titik akhir layanan.

layanan endpoint

Layanan yang dapat Anda host di cloud pribadi virtual (VPC) untuk dibagikan dengan pengguna lain. Anda dapat membuat layanan endpoint dengan AWS PrivateLink dan memberikan izin kepada prinsipal lain Akun AWS atau ke AWS Identity and Access Management (IAM). Akun atau prinsipal ini dapat terhubung ke layanan endpoint Anda secara pribadi dengan membuat titik akhir VPC antarmuka. Untuk informasi selengkapnya, lihat Membuat layanan titik akhir di dokumentasi Amazon Virtual Private Cloud (Amazon VPC).

perencanaan sumber daya perusahaan (ERP)

Sistem yang mengotomatiskan dan mengelola proses bisnis utama (seperti akuntansi, MES, dan manajemen proyek) untuk suatu perusahaan.

enkripsi amplop

Proses mengenkripsi kunci enkripsi dengan kunci enkripsi lain. Untuk informasi selengkapnya, lihat Enkripsi amplop dalam dokumentasi AWS Key Management Service (AWS KMS).

lingkungan

Sebuah contoh dari aplikasi yang sedang berjalan. Berikut ini adalah jenis lingkungan yang umum dalam komputasi awan:

- Development Environment — Sebuah contoh dari aplikasi yang berjalan yang hanya tersedia untuk tim inti yang bertanggung jawab untuk memelihara aplikasi. Lingkungan pengembangan digunakan untuk menguji perubahan sebelum mempromosikannya ke lingkungan atas. Jenis lingkungan ini kadang-kadang disebut sebagai lingkungan pengujian.
- lingkungan yang lebih rendah — Semua lingkungan pengembangan untuk aplikasi, seperti yang digunakan untuk build awal dan pengujian.
- lingkungan produksi — Sebuah contoh dari aplikasi yang berjalan yang dapat diakses oleh pengguna akhir. Dalam pipa CI/CD, lingkungan produksi adalah lingkungan penyebaran terakhir.
- lingkungan atas — Semua lingkungan yang dapat diakses oleh pengguna selain tim pengembangan inti. Ini dapat mencakup lingkungan produksi, lingkungan praproduksi, dan lingkungan untuk pengujian penerimaan pengguna.

epik

Dalam metodologi tangkas, kategori fungsional yang membantu mengatur dan memprioritaskan pekerjaan Anda. Epik memberikan deskripsi tingkat tinggi tentang persyaratan dan tugas

implementasi. Misalnya, epos keamanan AWS CAF mencakup manajemen identitas dan akses, kontrol detektif, keamanan infrastruktur, perlindungan data, dan respons insiden. Untuk informasi selengkapnya tentang epos dalam strategi AWS migrasi, lihat panduan implementasi program.

ERP

Lihat perencanaan sumber daya perusahaan.

analisis data eksplorasi (EDA)

Proses menganalisis dataset untuk memahami karakteristik utamanya. Anda mengumpulkan atau mengumpulkan data dan kemudian melakukan penyelidikan awal untuk menemukan pola, mendeteksi anomali, dan memeriksa asumsi. EDA dilakukan dengan menghitung statistik ringkasan dan membuat visualisasi data.

F

tabel fakta

Tabel tengah dalam skema bintang. Ini menyimpan data kuantitatif tentang operasi bisnis. Biasanya, tabel fakta berisi dua jenis kolom: kolom yang berisi ukuran dan yang berisi kunci asing ke tabel dimensi.

gagal cepat

Filosofi yang menggunakan pengujian yang sering dan bertahap untuk mengurangi siklus hidup pengembangan. Ini adalah bagian penting dari pendekatan tangkas.

batas isolasi kesalahan

Dalam AWS Cloud, batas seperti Availability Zone, Wilayah AWS, control plane, atau data plane yang membatasi efek kegagalan dan membantu meningkatkan ketahanan beban kerja. Untuk informasi selengkapnya, lihat Batas Isolasi AWS Kesalahan.

cabang fitur

Lihat cabang.

fitur

Data input yang Anda gunakan untuk membuat prediksi. Misalnya, dalam konteks manufaktur, fitur bisa berupa gambar yang diambil secara berkala dari lini manufaktur.

pentingnya fitur

Seberapa signifikan fitur untuk prediksi model. Ini biasanya dinyatakan sebagai skor numerik yang dapat dihitung melalui berbagai teknik, seperti Shapley Additive Explanations (SHAP) dan gradien terintegrasi. Untuk informasi lebih lanjut, lihat Interpretabilitas model pembelajaran mesin dengan:.AWS

transformasi fitur

Untuk mengoptimalkan data untuk proses ML, termasuk memperkaya data dengan sumber tambahan, menskalakan nilai, atau mengekstrak beberapa set informasi dari satu bidang data. Hal ini memungkinkan model ML untuk mendapatkan keuntungan dari data. Misalnya, jika Anda memecah tanggal “2021-05-27 00:15:37" menjadi “2021", “Mei”, “Kamis”, dan “15", Anda dapat membantu algoritme pembelajaran mempelajari pola bernuansa yang terkait dengan komponen data yang berbeda.

FGAC

Lihat kontrol akses berbutir halus.

kontrol akses berbutir halus (FGAC)

Penggunaan beberapa kondisi untuk mengizinkan atau menolak permintaan akses.

migrasi flash-cut

Metode migrasi database yang menggunakan replikasi data berkelanjutan melalui pengambilan data perubahan untuk memigrasikan data dalam waktu sesingkat mungkin, alih-alih menggunakan pendekatan bertahap. Tujuannya adalah untuk menjaga downtime seminimal mungkin.

G

pemblokiran geografis

Lihat pembatasan geografis.

pembatasan geografis (pemblokiran geografis)

Di Amazon CloudFront, opsi untuk mencegah pengguna di negara tertentu mengakses distribusi konten. Anda dapat menggunakan daftar izinkan atau daftar blokir untuk menentukan negara yang disetujui dan dilarang. Untuk informasi selengkapnya, lihat Membatasi distribusi geografis konten Anda dalam dokumentasi. CloudFront

Alur kerja Gitflow

Pendekatan di mana lingkungan bawah dan atas menggunakan cabang yang berbeda dalam repositori kode sumber. Alur kerja Gitflow dianggap warisan, dan alur kerja berbasis batang adalah pendekatan modern yang disukai.

strategi greenfield

Tidak adanya infrastruktur yang ada di lingkungan baru. Saat mengadopsi strategi greenfield untuk arsitektur sistem, Anda dapat memilih semua teknologi baru tanpa batasan kompatibilitas dengan infrastruktur yang ada, juga dikenal sebagai brownfield. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

pagar pembatas

Aturan tingkat tinggi yang membantu mengatur sumber daya, kebijakan, dan kepatuhan di seluruh unit organisasi (OU). Pagar pembatas preventif menegakkan kebijakan untuk memastikan keselarasan dengan standar kepatuhan. Mereka diimplementasikan dengan menggunakan kebijakan kontrol layanan dan batas izin IAM. Detective guardrails mendeteksi pelanggaran kebijakan dan masalah kepatuhan, dan menghasilkan peringatan untuk remediasi. Mereka diimplementasikan dengan menggunakan AWS Config, AWS Security Hub, Amazon GuardDuty AWS Trusted Advisor, Amazon Inspector, dan pemeriksaan khusus AWS Lambda .

H

HA

Lihat ketersediaan tinggi.

migrasi database heterogen

Memigrasi database sumber Anda ke database target yang menggunakan mesin database yang berbeda (misalnya, Oracle ke Amazon Aurora). Migrasi heterogen biasanya merupakan bagian dari upaya arsitektur ulang, dan mengubah skema dapat menjadi tugas yang kompleks. AWS menyediakan AWS SCT yang membantu dengan konversi skema.

ketersediaan tinggi (HA)

Kemampuan beban kerja untuk beroperasi terus menerus, tanpa intervensi, jika terjadi tantangan atau bencana. Sistem HA dirancang untuk gagal secara otomatis, secara konsisten memberikan kinerja berkualitas tinggi, dan menangani beban dan kegagalan yang berbeda dengan dampak kinerja minimal.

modernisasi sejarawan

Pendekatan yang digunakan untuk memodernisasi dan meningkatkan sistem teknologi operasional (OT) untuk melayani kebutuhan industri manufaktur dengan lebih baik. Sejarawan adalah jenis database yang digunakan untuk mengumpulkan dan menyimpan data dari berbagai sumber di pabrik.

migrasi database homogen

Memigrasi database sumber Anda ke database target yang berbagi mesin database yang sama (misalnya, Microsoft SQL Server ke Amazon RDS for SQL Server). Migrasi homogen biasanya merupakan bagian dari upaya rehosting atau replatforming. Anda dapat menggunakan utilitas database asli untuk memigrasi skema.

data panas

Data yang sering diakses, seperti data real-time atau data translasi terbaru. Data ini biasanya memerlukan tingkat atau kelas penyimpanan berkinerja tinggi untuk memberikan respons kueri yang cepat.

perbaikan terbaru

Perbaikan mendesak untuk masalah kritis dalam lingkungan produksi. Karena urgensinya, perbaikan terbaru biasanya dibuat di luar alur kerja DevOps rilis biasa.

periode hypercare

Segera setelah cutover, periode waktu ketika tim migrasi mengelola dan memantau aplikasi yang dimigrasi di cloud untuk mengatasi masalah apa pun. Biasanya, periode ini panjangnya 1-4 hari. Pada akhir periode hypercare, tim migrasi biasanya mentransfer tanggung jawab untuk aplikasi ke tim operasi cloud.

I

IAc

Lihat infrastruktur sebagai kode.

kebijakan berbasis identitas

Kebijakan yang dilampirkan pada satu atau beberapa prinsip IAM yang mendefinisikan izin mereka dalam lingkungan. AWS Cloud

aplikasi idle

Aplikasi yang memiliki penggunaan CPU dan memori rata-rata antara 5 dan 20 persen selama periode 90 hari. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini atau mempertahankannya di tempat.

IIoT

Lihat Internet of Things industri.

infrastruktur yang tidak dapat diubah

Model yang menyebarkan infrastruktur baru untuk beban kerja produksi alih-alih memperbarui, menambal, atau memodifikasi infrastruktur yang ada. Infrastruktur yang tidak dapat diubah secara inheren lebih konsisten, andal, dan dapat diprediksi daripada infrastruktur yang dapat berubah. Untuk informasi selengkapnya, lihat praktik terbaik Deploy using immutable infrastructure di AWS Well-Architected Framework.

masuk (masuknya) VPC

Dalam arsitektur AWS multi-akun, VPC yang menerima, memeriksa, dan merutekan koneksi jaringan dari luar aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

migrasi inkremental

Strategi cutover di mana Anda memigrasikan aplikasi Anda dalam bagian-bagian kecil alih-alih melakukan satu cutover penuh. Misalnya, Anda mungkin hanya memindahkan beberapa layanan mikro atau pengguna ke sistem baru pada awalnya. Setelah Anda memverifikasi bahwa semuanya berfungsi dengan baik, Anda dapat secara bertahap memindahkan layanan mikro atau pengguna tambahan hingga Anda dapat menonaktifkan sistem lama Anda. Strategi ini mengurangi risiko yang terkait dengan migrasi besar.

Industri 4.0

Sebuah istilah yang diperkenalkan oleh Klaus Schwab pada tahun 2016 untuk merujuk pada modernisasi proses manufaktur melalui kemajuan dalam konektivitas, data real-time, otomatisasi, analitik, dan AI/ML.

infrastruktur

Semua sumber daya dan aset yang terkandung dalam lingkungan aplikasi.

infrastruktur sebagai kode (IAc)

Proses penyediaan dan pengelolaan infrastruktur aplikasi melalui satu set file konfigurasi. IAc dirancang untuk membantu Anda memusatkan manajemen infrastruktur, menstandarisasi sumber daya, dan menskalakan dengan cepat sehingga lingkungan baru dapat diulang, andal, dan konsisten.

Internet of Things industri (IIoT)

Penggunaan sensor dan perangkat yang terhubung ke internet di sektor industri, seperti manufaktur, energi, otomotif, perawatan kesehatan, ilmu kehidupan, dan pertanian. Untuk informasi selengkapnya, lihat Membangun strategi transformasi digital Internet of Things (IIoT) industri.

inspeksi VPC

Dalam arsitektur AWS multi-akun, VPC terpusat yang mengelola inspeksi lalu lintas jaringan antara VPC (dalam hal yang sama atau berbeda Wilayah AWS), internet, dan jaringan lokal. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

Internet of Things (IoT)

Jaringan objek fisik yang terhubung dengan sensor atau prosesor tertanam yang berkomunikasi dengan perangkat dan sistem lain melalui internet atau melalui jaringan komunikasi lokal. Untuk informasi selengkapnya, lihat Apa itu IoT?

interpretabilitas

Karakteristik model pembelajaran mesin yang menggambarkan sejauh mana manusia dapat memahami bagaimana prediksi model bergantung pada inputnya. Untuk informasi selengkapnya, lihat Interpretabilitas model pembelajaran mesin dengan AWS.

IoT

Lihat Internet of Things.

Perpustakaan informasi TI (ITIL)

Serangkaian praktik terbaik untuk memberikan layanan TI dan menyelaraskan layanan ini dengan persyaratan bisnis. ITIL menyediakan dasar untuk ITSM.

Manajemen layanan TI (ITSM)

Kegiatan yang terkait dengan merancang, menerapkan, mengelola, dan mendukung layanan TI untuk suatu organisasi. Untuk informasi tentang mengintegrasikan operasi cloud dengan alat ITSM, lihat panduan integrasi operasi.

ITIL

Lihat perpustakaan informasi TI.

ITSM

Lihat manajemen layanan TI.

L

kontrol akses berbasis label (LBAC)

Implementasi kontrol akses wajib (MAC) di mana pengguna dan data itu sendiri masing-masing secara eksplisit diberi nilai label keamanan. Persimpangan antara label keamanan pengguna dan label keamanan data menentukan baris dan kolom mana yang dapat dilihat oleh pengguna.

landing zone

Landing zone adalah AWS lingkungan multi-akun yang dirancang dengan baik yang dapat diskalakan dan aman. Ini adalah titik awal dari mana organisasi Anda dapat dengan cepat meluncurkan dan menyebarkan beban kerja dan aplikasi dengan percaya diri dalam lingkungan keamanan dan infrastruktur mereka. Untuk informasi selengkapnya tentang zona pendaratan, lihat Menyiapkan lingkungan multi-akun AWS yang aman dan dapat diskalakan.

migrasi besar

Migrasi 300 atau lebih server.

LBAC

Lihat kontrol akses berbasis label.

hak istimewa paling sedikit

Praktik keamanan terbaik untuk memberikan izin minimum yang diperlukan untuk melakukan tugas. Untuk informasi selengkapnya, lihat Menerapkan izin hak istimewa terkecil dalam dokumentasi IAM.

angkat dan geser

Lihat 7 Rs.

sistem endian kecil

Sebuah sistem yang menyimpan byte paling tidak signifikan terlebih dahulu. Lihat juga endianness.

lingkungan yang lebih rendah

Lihat lingkungan.

M

pembelajaran mesin (ML)

Jenis kecerdasan buatan yang menggunakan algoritma dan teknik untuk pengenalan pola dan pembelajaran. ML menganalisis dan belajar dari data yang direkam, seperti data Internet of Things (IoT), untuk menghasilkan model statistik berdasarkan pola. Untuk informasi selengkapnya, lihat Machine Learning.

cabang utama

Lihat cabang.

malware

Perangkat lunak yang dirancang untuk membahayakan keamanan atau privasi komputer. Malware dapat mengganggu sistem komputer, membocorkan informasi sensitif, atau mendapatkan akses yang tidak sah. Contoh malware termasuk virus, worm, ransomware, Trojan horse, spyware, dan keyloggers.

layanan terkelola

Layanan AWS yang AWS mengoperasikan lapisan infrastruktur, sistem operasi, dan platform, dan Anda mengakses titik akhir untuk menyimpan dan mengambil data. Amazon Simple Storage Service (Amazon S3) dan Amazon DynamoDB adalah contoh layanan terkelola. Ini juga dikenal sebagai layanan abstrak.

sistem eksekusi manufaktur (MES)

Sistem perangkat lunak untuk melacak, memantau, mendokumentasikan, dan mengendalikan proses produksi yang mengubah bahan baku menjadi produk jadi di lantai toko.

PETA

Lihat Program Percepatan Migrasi.

mekanisme

Proses lengkap di mana Anda membuat alat, mendorong adopsi alat, dan kemudian memeriksa hasilnya untuk melakukan penyesuaian. Mekanisme adalah siklus yang memperkuat dan meningkatkan dirinya sendiri saat beroperasi. Untuk informasi lebih lanjut, lihat Membangun mekanisme di AWS Well-Architected Framework.

akun anggota

Semua Akun AWS selain akun manajemen yang merupakan bagian dari organisasi di AWS Organizations. Akun dapat menjadi anggota dari hanya satu organisasi pada suatu waktu.

MES

Lihat sistem eksekusi manufaktur.

Transportasi Telemetri Antrian Pesan (MQTT)

Protokol komunikasi ringan machine-to-machine (M2M), berdasarkan pola terbitkan/berlangganan, untuk perangkat IoT yang dibatasi sumber daya.

layanan mikro

Layanan kecil dan independen yang berkomunikasi melalui API yang terdefinisi dengan baik dan biasanya dimiliki oleh tim kecil yang mandiri. Misalnya, sistem asuransi mungkin mencakup layanan mikro yang memetakan kemampuan bisnis, seperti penjualan atau pemasaran, atau subdomain, seperti pembelian, klaim, atau analitik. Manfaat layanan mikro termasuk kelincahan, penskalaan yang fleksibel, penyebaran yang mudah, kode yang dapat digunakan kembali, dan ketahanan. Untuk informasi selengkapnya, lihat Mengintegrasikan layanan mikro dengan menggunakan layanan tanpa AWS server.

arsitektur microservices

Pendekatan untuk membangun aplikasi dengan komponen independen yang menjalankan setiap proses aplikasi sebagai layanan mikro. Layanan mikro ini berkomunikasi melalui antarmuka yang terdefinisi dengan baik dengan menggunakan API ringan. Setiap layanan mikro dalam arsitektur ini dapat diperbarui, digunakan, dan diskalakan untuk memenuhi permintaan fungsi tertentu dari suatu aplikasi. Untuk informasi selengkapnya, lihat Menerapkan layanan mikro di AWS.

Program Percepatan Migrasi (MAP)

AWS Program yang menyediakan dukungan konsultasi, pelatihan, dan layanan untuk membantu organisasi membangun fondasi operasional yang kuat untuk pindah ke cloud, dan untuk membantu mengimbangi biaya awal migrasi. MAP mencakup metodologi migrasi untuk mengeksekusi migrasi lama dengan cara metodis dan seperangkat alat untuk mengotomatisasi dan mempercepat skenario migrasi umum.

migrasi dalam skala

Proses memindahkan sebagian besar portofolio aplikasi ke cloud dalam gelombang, dengan lebih banyak aplikasi bergerak pada tingkat yang lebih cepat di setiap gelombang. Fase ini menggunakan praktik terbaik dan pelajaran yang dipetik dari fase sebelumnya untuk mengimplementasikan pabrik migrasi tim, alat, dan proses untuk merampingkan migrasi beban kerja melalui otomatisasi dan pengiriman tangkas. Ini adalah fase ketiga dari strategi AWS migrasi.

pabrik migrasi

Tim lintas fungsi yang merampingkan migrasi beban kerja melalui pendekatan otomatis dan gesit. Tim pabrik migrasi biasanya mencakup operasi, analis dan pemilik bisnis, insinyur migrasi, pengembang, dan DevOps profesional yang bekerja di sprint. Antara 20 dan 50 persen portofolio aplikasi perusahaan terdiri dari pola berulang yang dapat dioptimalkan dengan pendekatan pabrik. Untuk informasi selengkapnya, lihat diskusi tentang pabrik migrasi dan panduan Pabrik Migrasi Cloud di kumpulan konten ini.

metadata migrasi

Informasi tentang aplikasi dan server yang diperlukan untuk menyelesaikan migrasi. Setiap pola migrasi memerlukan satu set metadata migrasi yang berbeda. Contoh metadata migrasi termasuk subnet target, grup keamanan, dan akun. AWS

pola migrasi

Tugas migrasi berulang yang merinci strategi migrasi, tujuan migrasi, dan aplikasi atau layanan migrasi yang digunakan. Contoh: Rehost migrasi ke Amazon EC2 AWS dengan Layanan Migrasi Aplikasi.

Penilaian Portofolio Migrasi (MPA)

Alat online yang menyediakan informasi untuk memvalidasi kasus bisnis untuk bermigrasi ke Cloud. AWS MPA menyediakan penilaian portofolio terperinci (ukuran kanan server, harga,

perbandingan TCO, analisis biaya migrasi) serta perencanaan migrasi (analisis data aplikasi dan pengumpulan data, pengelompokan aplikasi, prioritas migrasi, dan perencanaan gelombang). Alat MPA (memerlukan login) tersedia gratis untuk semua AWS konsultan dan konsultan APN Partner.

Penilaian Kesiapan Migrasi (MRA)

Proses mendapatkan wawasan tentang status kesiapan cloud organisasi, mengidentifikasi kekuatan dan kelemahan, dan membangun rencana aksi untuk menutup kesenjangan yang diidentifikasi, menggunakan CAF. AWS Untuk informasi selengkapnya, lihat panduan kesiapan migrasi. MRA adalah tahap pertama dari strategi AWS migrasi.

strategi migrasi

Pendekatan yang digunakan untuk memigrasikan beban kerja ke Cloud. AWS Untuk informasi lebih lanjut, lihat entri 7 Rs di glosarium ini dan lihat Memobilisasi organisasi Anda untuk mempercepat migrasi skala besar.

ML

Lihat pembelajaran mesin.

modernisasi

Mengubah aplikasi usang (warisan atau monolitik) dan infrastrukturnya menjadi sistem yang gesit, elastis, dan sangat tersedia di cloud untuk mengurangi biaya, mendapatkan efisiensi, dan memanfaatkan inovasi. Untuk informasi selengkapnya, lihat Strategi untuk memodernisasi aplikasi di. AWS Cloud

penilaian kesiapan modernisasi

Evaluasi yang membantu menentukan kesiapan modernisasi aplikasi organisasi; mengidentifikasi manfaat, risiko, dan dependensi; dan menentukan seberapa baik organisasi dapat mendukung keadaan masa depan aplikasi tersebut. Hasil penilaian adalah cetak biru arsitektur target, peta jalan yang merinci fase pengembangan dan tonggak untuk proses modernisasi, dan rencana aksi untuk mengatasi kesenjangan yang diidentifikasi. Untuk informasi selengkapnya, lihat Mengevaluasi kesiapan modernisasi untuk aplikasi di Cloud. AWS

aplikasi monolitik (monolit)

Aplikasi yang berjalan sebagai layanan tunggal dengan proses yang digabungkan secara ketat. Aplikasi monolitik memiliki beberapa kelemahan. Jika satu fitur aplikasi mengalami lonjakan permintaan, seluruh arsitektur harus diskalakan. Menambahkan atau meningkatkan fitur aplikasi monolitik juga menjadi lebih kompleks ketika basis kode tumbuh. Untuk mengatasi masalah ini,

Anda dapat menggunakan arsitektur microservices. Untuk informasi lebih lanjut, lihat Menguraikan monolit menjadi layanan mikro.

MPA

Lihat Penilaian Portofolio Migrasi.

MQTT

Lihat Transportasi Telemetri Antrian Pesan.

klasifikasi multiclass

Sebuah proses yang membantu menghasilkan prediksi untuk beberapa kelas (memprediksi satu dari lebih dari dua hasil). Misalnya, model ML mungkin bertanya "Apakah produk ini buku, mobil, atau telepon?" atau "Kategori produk mana yang paling menarik bagi pelanggan ini?"

infrastruktur yang bisa berubah

Model yang memperbarui dan memodifikasi infrastruktur yang ada untuk beban kerja produksi. Untuk meningkatkan konsistensi, keandalan, dan prediktabilitas, AWS Well-Architected Framework merekomendasikan penggunaan infrastruktur yang tidak dapat diubah sebagai praktik terbaik.

O

OAC

Lihat kontrol akses asal.

OAI

Lihat identitas akses asal.

OCM

Lihat manajemen perubahan organisasi.

migrasi offline

Metode migrasi di mana beban kerja sumber diturunkan selama proses migrasi. Metode ini melibatkan waktu henti yang diperpanjang dan biasanya digunakan untuk beban kerja kecil dan tidak kritis.

OI

Lihat integrasi operasi.

OLA

Lihat perjanjian tingkat operasional.

migrasi online

Metode migrasi di mana beban kerja sumber disalin ke sistem target tanpa diambil offline. Aplikasi yang terhubung ke beban kerja dapat terus berfungsi selama migrasi. Metode ini melibatkan waktu henti nol hingga minimal dan biasanya digunakan untuk beban kerja produksi yang kritis.

OPC-UA

Lihat Komunikasi Proses Terbuka - Arsitektur Terpadu.

Komunikasi Proses Terbuka - Arsitektur Terpadu (OPC-UA)

Protokol komunikasi machine-to-machine (M2M) untuk otomasi industri. OPC-UA menyediakan standar interoperabilitas dengan enkripsi data, otentikasi, dan skema otorisasi.

perjanjian tingkat operasional (OLA)

Perjanjian yang menjelaskan apa yang dijanjikan kelompok TI fungsional untuk diberikan satu sama lain, untuk mendukung perjanjian tingkat layanan (SLA).

Tinjauan Kesiapan Operasional (ORR)

Daftar pertanyaan dan praktik terbaik terkait yang membantu Anda memahami, mengevaluasi, mencegah, atau mengurangi ruang lingkup insiden dan kemungkinan kegagalan. Untuk informasi lebih lanjut, lihat Ulasan Kesiapan Operasional (ORR) dalam Kerangka Kerja Well-Architected AWS .

teknologi operasional (OT)

Sistem perangkat keras dan perangkat lunak yang bekerja dengan lingkungan fisik untuk mengendalikan operasi industri, peralatan, dan infrastruktur. Di bidang manufaktur, integrasi sistem OT dan teknologi informasi (TI) adalah fokus utama untuk transformasi Industri 4.0.

integrasi operasi (OI)

Proses modernisasi operasi di cloud, yang melibatkan perencanaan kesiapan, otomatisasi, dan integrasi. Untuk informasi selengkapnya, lihat panduan integrasi operasi.

jejak organisasi

Jejak yang dibuat oleh AWS CloudTrail itu mencatat semua peristiwa untuk semua Akun AWS dalam organisasi di AWS Organizations. Jejak ini dibuat di setiap Akun AWS bagian organisasi dan melacak aktivitas di setiap akun. Untuk informasi selengkapnya, lihat Membuat jejak untuk organisasi dalam CloudTrail dokumentasi.

manajemen perubahan organisasi (OCM)

Kerangka kerja untuk mengelola transformasi bisnis utama yang mengganggu dari perspektif orang, budaya, dan kepemimpinan. OCM membantu organisasi mempersiapkan, dan transisi ke, sistem dan strategi baru dengan mempercepat adopsi perubahan, mengatasi masalah transisi, dan mendorong perubahan budaya dan organisasi. Dalam strategi AWS migrasi, kerangka kerja ini disebut percepatan orang, karena kecepatan perubahan yang diperlukan dalam proyek adopsi cloud. Untuk informasi lebih lanjut, lihat panduan OCM.

kontrol akses asal (OAC)

Di CloudFront, opsi yang disempurnakan untuk membatasi akses untuk mengamankan konten Amazon Simple Storage Service (Amazon S3) Anda. OAC mendukung semua bucket S3 di semua Wilayah AWS, enkripsi sisi server dengan AWS KMS (SSE-KMS), dan dinamis dan permintaan ke bucket S3. `PUT` `DELETE`

identitas akses asal (OAI)

Di CloudFront, opsi untuk membatasi akses untuk mengamankan konten Amazon S3 Anda. Saat Anda menggunakan OAI, CloudFront buat prinsipal yang dapat diautentikasi oleh Amazon S3. Prinsipal yang diautentikasi dapat mengakses konten dalam bucket S3 hanya melalui distribusi tertentu. CloudFront Lihat juga OAC, yang menyediakan kontrol akses yang lebih terperinci dan ditingkatkan.

ORR

Lihat tinjauan kesiapan operasional.

OT

Lihat teknologi operasional.

keluar (jalan keluar) VPC

Dalam arsitektur AWS multi-akun, VPC yang menangani koneksi jaringan yang dimulai dari dalam aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan

Anda dengan VPC masuk, keluar, dan inspeksi untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

P

batas izin

Kebijakan manajemen IAM yang dilampirkan pada prinsipal IAM untuk menetapkan izin maksimum yang dapat dimiliki pengguna atau peran. Untuk informasi selengkapnya, lihat Batas izin dalam dokumentasi IAM.

Informasi Identifikasi Pribadi (PII)

Informasi yang, jika dilihat secara langsung atau dipasangkan dengan data terkait lainnya, dapat digunakan untuk menyimpulkan identitas individu secara wajar. Contoh PII termasuk nama, alamat, dan informasi kontak.

PII

Lihat informasi yang dapat diidentifikasi secara pribadi.

buku pedoman

Serangkaian langkah yang telah ditentukan sebelumnya yang menangkap pekerjaan yang terkait dengan migrasi, seperti mengirimkan fungsi operasi inti di cloud. Buku pedoman dapat berupa skrip, runbook otomatis, atau ringkasan proses atau langkah-langkah yang diperlukan untuk mengoperasikan lingkungan modern Anda.

PLC

Lihat pengontrol logika yang dapat diprogram.

PLM

Lihat manajemen siklus hidup produk.

kebijakan

Objek yang dapat menentukan izin (lihat kebijakan berbasis identitas), menentukan kondisi akses (lihat kebijakanberbasis sumber daya), atau menentukan izin maksimum untuk semua akun dalam organisasi di (lihat kebijakan kontrol layanan). AWS Organizations

persistensi poliglot

Secara independen memilih teknologi penyimpanan data microservice berdasarkan pola akses data dan persyaratan lainnya. Jika layanan mikro Anda memiliki teknologi penyimpanan data yang sama, mereka dapat menghadapi tantangan implementasi atau mengalami kinerja yang buruk. Layanan mikro lebih mudah diimplementasikan dan mencapai kinerja dan skalabilitas yang lebih baik jika mereka menggunakan penyimpanan data yang paling sesuai dengan kebutuhan mereka. Untuk informasi selengkapnya, lihat Mengaktifkan persistensi data di layanan mikro.

penilaian portofolio

Proses menemukan, menganalisis, dan memprioritaskan portofolio aplikasi untuk merencanakan migrasi. Untuk informasi selengkapnya, lihat Mengevaluasi kesiapan migrasi.

predikat

Kondisi kueri yang mengembalikan `true` atau`false`, biasanya terletak di `WHERE` klausa.

predikat pushdown

Teknik optimasi kueri database yang menyaring data dalam kueri sebelum transfer. Ini mengurangi jumlah data yang harus diambil dan diproses dari database relasional, dan meningkatkan kinerja kueri.

kontrol preventif

Kontrol keamanan yang dirancang untuk mencegah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan pertama untuk membantu mencegah akses tidak sah atau perubahan yang tidak diinginkan ke jaringan Anda. Untuk informasi selengkapnya, lihat Kontrol pencegahan dalam Menerapkan kontrol keamanan pada. AWS

principal

Entitas AWS yang dapat melakukan tindakan dan mengakses sumber daya. Entitas ini biasanya merupakan pengguna root untuk Akun AWS, peran IAM, atau pengguna. Untuk informasi selengkapnya, lihat Prinsip dalam istilah dan konsep Peran dalam dokumentasi IAM.

Privasi oleh Desain

Pendekatan dalam rekayasa sistem yang memperhitungkan privasi di seluruh proses rekayasa.

zona host pribadi

Container yang menyimpan informasi tentang bagaimana Anda ingin Amazon Route 53 merespons kueri DNS untuk domain dan subdomainnya dalam satu atau beberapa VPC. Untuk

informasi selengkapnya, lihat Bekerja dengan zona yang dihosting pribadi di dokumentasi Route 53.

kontrol proaktif

Kontrol keamanan yang dirancang untuk mencegah penyebaran sumber daya yang tidak sesuai. Kontrol ini memindai sumber daya sebelum disediakan. Jika sumber daya tidak sesuai dengan kontrol, maka itu tidak disediakan. Untuk informasi selengkapnya, lihat panduan referensi Kontrol dalam AWS Control Tower dokumentasi dan lihat Kontrol proaktif dalam Menerapkan kontrol keamanan pada AWS.

manajemen siklus hidup produk (PLM)

Manajemen data dan proses untuk suatu produk di seluruh siklus hidupnya, mulai dari desain, pengembangan, dan peluncuran, melalui pertumbuhan dan kematangan, hingga penurunan dan penghapusan.

lingkungan produksi

Lihat lingkungan.

pengontrol logika yang dapat diprogram (PLC)

Di bidang manufaktur, komputer yang sangat andal dan mudah beradaptasi yang memantau mesin dan mengotomatiskan proses manufaktur.

pseudonimisasi

Proses penggantian pengenal pribadi dalam kumpulan data dengan nilai placeholder. Pseudonimisasi dapat membantu melindungi privasi pribadi. Data pseudonim masih dianggap sebagai data pribadi.

terbitkan/berlangganan (pub/sub)

Pola yang memungkinkan komunikasi asinkron antara layanan mikro untuk meningkatkan skalabilitas dan daya tanggap. Misalnya, dalam MES berbasis layanan mikro, layanan mikro dapat mempublikasikan pesan peristiwa ke saluran yang dapat berlangganan oleh layanan mikro lainnya. Sistem dapat menambahkan layanan mikro baru tanpa mengubah layanan penerbitan.

Q

rencana kueri

Serangkaian langkah, seperti instruksi, yang digunakan untuk mengakses data dalam sistem database relasional SQL.

regresi rencana kueri

Ketika pengoptimal layanan database memilih rencana yang kurang optimal daripada sebelum perubahan yang diberikan ke lingkungan database. Hal ini dapat disebabkan oleh perubahan statistik, kendala, pengaturan lingkungan, pengikatan parameter kueri, dan pembaruan ke mesin database.

R

Matriks RACI

Lihat bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI).

ransomware

Perangkat lunak berbahaya yang dirancang untuk memblokir akses ke sistem komputer atau data sampai pembayaran dilakukan.

Matriks RASCI

Lihat bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI).

RCAC

Lihat kontrol akses baris dan kolom.

replika baca

Salinan database yang digunakan untuk tujuan read-only. Anda dapat merutekan kueri ke replika baca untuk mengurangi beban pada database utama Anda.

arsitek ulang

Lihat 7 Rs.

tujuan titik pemulihan (RPO)

Jumlah waktu maksimum yang dapat diterima sejak titik pemulihan data terakhir. Ini menentukan apa yang dianggap sebagai hilangnya data yang dapat diterima antara titik pemulihan terakhir dan gangguan layanan.

tujuan waktu pemulihan (RTO)

Penundaan maksimum yang dapat diterima antara gangguan layanan dan pemulihan layanan.

refactor

Lihat 7 Rs.

Wilayah

Kumpulan AWS sumber daya di wilayah geografis. Masing-masing Wilayah AWS terisolasi dan independen dari yang lain untuk memberikan toleransi kesalahan, stabilitas, dan ketahanan. Untuk informasi selengkapnya, lihat Menentukan Wilayah AWS akun yang dapat digunakan.

regresi

Teknik ML yang memprediksi nilai numerik. Misalnya, untuk memecahkan masalah “Berapa harga rumah ini akan dijual?” Model ML dapat menggunakan model regresi linier untuk memprediksi harga jual rumah berdasarkan fakta yang diketahui tentang rumah (misalnya, luas persegi).

rehost

Lihat 7 Rs.

melepaskan

Dalam proses penyebaran, tindakan mempromosikan perubahan pada lingkungan produksi.

memindahkan

Lihat 7 Rs.

memplatform ulang

Lihat 7 Rs.

pembelian kembali

Lihat 7 Rs.

ketahanan

Kemampuan aplikasi untuk melawan atau pulih dari gangguan. Ketersediaan tinggi dan pemulihan bencana adalah pertimbangan umum ketika merencanakan ketahanan di. AWS Cloud Untuk informasi lebih lanjut, lihat AWS Cloud Ketahanan.

kebijakan berbasis sumber daya

Kebijakan yang dilampirkan ke sumber daya, seperti bucket Amazon S3, titik akhir, atau kunci enkripsi. Jenis kebijakan ini menentukan prinsipal mana yang diizinkan mengakses, tindakan yang didukung, dan kondisi lain yang harus dipenuhi.

matriks yang bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI)

Matriks yang mendefinisikan peran dan tanggung jawab untuk semua pihak yang terlibat dalam kegiatan migrasi dan operasi cloud. Nama matriks berasal dari jenis tanggung jawab yang didefinisikan dalam matriks: bertanggung jawab (R), akuntabel (A), dikonsultasikan (C), dan diinformasikan (I). Jenis dukungan (S) adalah opsional. Jika Anda menyertakan dukungan, matriks disebut matriks RASCI, dan jika Anda mengecualikannya, itu disebut matriks RACI.

kontrol responsif

Kontrol keamanan yang dirancang untuk mendorong remediasi efek samping atau penyimpangan dari garis dasar keamanan Anda. Untuk informasi selengkapnya, lihat Kontrol responsif dalam Menerapkan kontrol keamanan pada AWS.

melestarikan

Lihat 7 Rs.

pensiun

Lihat 7 Rs.

rotasi

Proses memperbarui rahasia secara berkala untuk membuatnya lebih sulit bagi penyerang untuk mengakses kredensil.

kontrol akses baris dan kolom (RCAC)

Penggunaan ekspresi SQL dasar dan fleksibel yang telah menetapkan aturan akses. RCAC terdiri dari izin baris dan topeng kolom.

RPO

Lihat tujuan titik pemulihan.

RTO

Lihat tujuan waktu pemulihan.

buku runbook

Satu set prosedur manual atau otomatis yang diperlukan untuk melakukan tugas tertentu. Ini biasanya dibangun untuk merampingkan operasi berulang atau prosedur dengan tingkat kesalahan yang tinggi.

D

SAML 2.0

Standar terbuka yang digunakan oleh banyak penyedia identitas (IdPs). Fitur ini memungkinkan sistem masuk tunggal gabungan (SSO), sehingga pengguna dapat masuk ke AWS Management Console atau memanggil operasi AWS API tanpa Anda harus membuat pengguna di IAM untuk semua orang di organisasi Anda. Untuk informasi lebih lanjut tentang federasi berbasis SAMP 2.0, lihat Tentang federasi berbasis SAMP 2.0 dalam dokumentasi IAM.

PENIPUAN

Lihat kontrol pengawasan dan akuisisi data.

SCP

Lihat kebijakan kontrol layanan.

Rahasia

Dalam AWS Secrets Manager, informasi rahasia atau terbatas, seperti kata sandi atau kredensil pengguna, yang Anda simpan dalam bentuk terenkripsi. Ini terdiri dari nilai rahasia dan metadata-nya. Nilai rahasia dapat berupa biner, string tunggal, atau beberapa string. Untuk informasi selengkapnya, lihat Rahasia dalam dokumentasi Secrets Manager.

kontrol keamanan

Pagar pembatas teknis atau administratif yang mencegah, mendeteksi, atau mengurangi kemampuan pelaku ancaman untuk mengeksploitasi kerentanan keamanan. Ada empat jenis kontrol keamanan utama: preventif, detektif, responsif, dan proaktif.

pengerasan keamanan

Proses mengurangi permukaan serangan untuk membuatnya lebih tahan terhadap serangan. Ini dapat mencakup tindakan seperti menghapus sumber daya yang tidak lagi diperlukan, menerapkan praktik keamanan terbaik untuk memberikan hak istimewa paling sedikit, atau menonaktifkan fitur yang tidak perlu dalam file konfigurasi.

sistem informasi keamanan dan manajemen acara (SIEM)

Alat dan layanan yang menggabungkan sistem manajemen informasi keamanan (SIM) dan manajemen acara keamanan (SEM). Sistem SIEM mengumpulkan, memantau, dan menganalisis data dari server, jaringan, perangkat, dan sumber lain untuk mendeteksi ancaman dan pelanggaran keamanan, dan untuk menghasilkan peringatan.

otomatisasi respons keamanan

Tindakan yang telah ditentukan dan diprogram yang dirancang untuk secara otomatis merespons atau memulihkan peristiwa keamanan. Otomatisasi ini berfungsi sebagai kontrol keamanan detektif atau responsif yang membantu Anda menerapkan praktik terbaik AWS keamanan. Contoh tindakan respons otomatis termasuk memodifikasi grup keamanan VPC, menambal instans Amazon EC2, atau memutar kredensil.

enkripsi sisi server

Enkripsi data di tujuannya, oleh Layanan AWS yang menerimanya.

kebijakan kontrol layanan (SCP)

Kebijakan yang menyediakan kontrol terpusat atas izin untuk semua akun di organisasi. AWS Organizations SCP menentukan pagar pembatas atau menetapkan batasan pada tindakan yang dapat didelegasikan oleh administrator kepada pengguna atau peran. Anda dapat menggunakan SCP sebagai daftar izin atau daftar penolakan, untuk menentukan layanan atau tindakan mana yang diizinkan atau dilarang. Untuk informasi selengkapnya, lihat Kebijakan kontrol layanan dalam AWS Organizations dokumentasi.

titik akhir layanan

URL titik masuk untuk file Layanan AWS. Anda dapat menggunakan endpoint untuk terhubung secara terprogram ke layanan target. Untuk informasi selengkapnya, lihat Layanan AWS titik akhir di Referensi Umum AWS.

perjanjian tingkat layanan (SLA)

Perjanjian yang menjelaskan apa yang dijanjikan tim TI untuk diberikan kepada pelanggan mereka, seperti waktu kerja dan kinerja layanan.

indikator tingkat layanan (SLI)

Pengukuran aspek kinerja layanan, seperti tingkat kesalahan, ketersediaan, atau throughputnya.

tujuan tingkat layanan (SLO)

Metrik target yang mewakili kesehatan layanan, yang diukur dengan indikator tingkat layanan.

model tanggung jawab bersama

Model yang menjelaskan tanggung jawab yang Anda bagikan AWS untuk keamanan dan kepatuhan cloud. AWS bertanggung jawab atas keamanan cloud, sedangkan Anda bertanggung jawab atas keamanan di cloud. Untuk informasi selengkapnya, lihat Model tanggung jawab bersama.

SIEM

Lihat informasi keamanan dan sistem manajemen acara.

titik kegagalan tunggal (SPOF)

Kegagalan dalam satu komponen penting dari aplikasi yang dapat mengganggu sistem.

SLA

Lihat perjanjian tingkat layanan.

SLI

Lihat indikator tingkat layanan.

SLO

Lihat tujuan tingkat layanan.

split-and-seed model

Pola untuk menskalakan dan mempercepat proyek modernisasi. Ketika fitur baru dan rilis produk didefinisikan, tim inti berpisah untuk membuat tim produk baru. Ini membantu meningkatkan kemampuan dan layanan organisasi Anda, meningkatkan produktivitas pengembang, dan

mendukung inovasi yang cepat. Untuk informasi lebih lanjut, lihat Pendekatan bertahap untuk memodernisasi aplikasi di. AWS Cloud

SPOF

Lihat satu titik kegagalan.

skema bintang

Struktur organisasi database yang menggunakan satu tabel fakta besar untuk menyimpan data transaksional atau terukur dan menggunakan satu atau lebih tabel dimensi yang lebih kecil untuk menyimpan atribut data. Struktur ini dirancang untuk digunakan dalam gudang data atau untuk tujuan intelijen bisnis.

pola ara pencekik

Pendekatan untuk memodernisasi sistem monolitik dengan menulis ulang secara bertahap dan mengganti fungsionalitas sistem sampai sistem warisan dapat dinonaktifkan. Pola ini menggunakan analogi pohon ara yang tumbuh menjadi pohon yang sudah mapan dan akhirnya mengatasi dan menggantikan inangnya. Pola ini diperkenalkan oleh Martin Fowler sebagai cara untuk mengelola risiko saat menulis ulang sistem monolitik. Untuk contoh cara menerapkan pola ini, lihat Memodernisasi layanan web Microsoft ASP.NET (ASMX) lama secara bertahap menggunakan container dan Amazon API Gateway.

subnet

Rentang alamat IP dalam VPC Anda. Subnet harus berada di Availability Zone tunggal.

kontrol pengawasan dan akuisisi data (SCADA)

Di bidang manufaktur, sistem yang menggunakan perangkat keras dan perangkat lunak untuk memantau aset fisik dan operasi produksi.

enkripsi simetris

Algoritma enkripsi yang menggunakan kunci yang sama untuk mengenkripsi dan mendekripsi data.

pengujian sintetis

Menguji sistem dengan cara yang mensimulasikan interaksi pengguna untuk mendeteksi potensi masalah atau untuk memantau kinerja. Anda dapat menggunakan Amazon CloudWatch Synthetics untuk membuat tes ini.

T

tag

Pasangan nilai kunci yang bertindak sebagai metadata untuk mengatur sumber daya Anda. AWS Tanda dapat membantu Anda mengelola, mengidentifikasi, mengatur, dan memfilter sumber daya. Untuk informasi selengkapnya, lihat Menandai AWS sumber daya Anda.

variabel target

Nilai yang Anda coba prediksi dalam ML yang diawasi. Ini juga disebut sebagai variabel hasil. Misalnya, dalam pengaturan manufaktur, variabel target bisa menjadi cacat produk.

daftar tugas

Alat yang digunakan untuk melacak kemajuan melalui runbook. Daftar tugas berisi ikhtisar runbook dan daftar tugas umum yang harus diselesaikan. Untuk setiap tugas umum, itu termasuk perkiraan jumlah waktu yang dibutuhkan, pemilik, dan kemajuan.

lingkungan uji

Lihat lingkungan.

pelatihan

Untuk menyediakan data bagi model ML Anda untuk dipelajari. Data pelatihan harus berisi jawaban yang benar. Algoritma pembelajaran menemukan pola dalam data pelatihan yang memetakan atribut data input ke target (jawaban yang ingin Anda prediksi). Ini menghasilkan model ML yang menangkap pola-pola ini. Anda kemudian dapat menggunakan model ML untuk membuat prediksi pada data baru yang Anda tidak tahu targetnya.

gerbang transit

Hub transit jaringan yang dapat Anda gunakan untuk menghubungkan VPC dan jaringan lokal Anda. Untuk informasi selengkapnya, lihat Apa itu gateway transit dalam AWS Transit Gateway dokumentasi.

alur kerja berbasis batang

Pendekatan di mana pengembang membangun dan menguji fitur secara lokal di cabang fitur dan kemudian menggabungkan perubahan tersebut ke cabang utama. Cabang utama kemudian dibangun untuk pengembangan, praproduksi, dan lingkungan produksi, secara berurutan.

akses tepercaya

Memberikan izin ke layanan yang Anda tentukan untuk melakukan tugas di organisasi Anda di dalam AWS Organizations dan di akunnya atas nama Anda. Layanan tepercaya menciptakan peran terkait layanan di setiap akun, ketika peran itu diperlukan, untuk melakukan tugas manajemen untuk Anda. Untuk informasi selengkapnya, lihat Menggunakan AWS Organizations dengan AWS layanan lain dalam AWS Organizations dokumentasi.

penyetelan

Untuk mengubah aspek proses pelatihan Anda untuk meningkatkan akurasi model ML. Misalnya, Anda dapat melatih model ML dengan membuat set pelabelan, menambahkan label, dan kemudian mengulangi langkah-langkah ini beberapa kali di bawah pengaturan yang berbeda untuk mengoptimalkan model.

tim dua pizza

Sebuah DevOps tim kecil yang bisa Anda beri makan dengan dua pizza. Ukuran tim dua pizza memastikan peluang terbaik untuk berkolaborasi dalam pengembangan perangkat lunak.

U

waswas

Sebuah konsep yang mengacu pada informasi yang tidak tepat, tidak lengkap, atau tidak diketahui yang dapat merusak keandalan model ML prediktif. Ada dua jenis ketidakpastian: ketidakpastian epistemik disebabkan oleh data yang terbatas dan tidak lengkap, sedangkan ketidakpastian aleatorik disebabkan oleh kebisingan dan keacakan yang melekat dalam data. Untuk informasi lebih lanjut, lihat panduan Mengukur ketidakpastian dalam sistem pembelajaran mendalam.

tugas yang tidak terdiferensiasi

Juga dikenal sebagai angkat berat, pekerjaan yang diperlukan untuk membuat dan mengoperasikan aplikasi tetapi itu tidak memberikan nilai langsung kepada pengguna akhir atau memberikan keunggulan kompetitif. Contoh tugas yang tidak terdiferensiasi termasuk pengadaan, pemeliharaan, dan perencanaan kapasitas.

lingkungan atas

Lihat lingkungan.

V

menyedot debu

Operasi pemeliharaan database yang melibatkan pembersihan setelah pembaruan tambahan untuk merebut kembali penyimpanan dan meningkatkan kinerja.

kendali versi

Proses dan alat yang melacak perubahan, seperti perubahan kode sumber dalam repositori.

Peering VPC

Koneksi antara dua VPC yang memungkinkan Anda merutekan lalu lintas dengan menggunakan alamat IP pribadi. Untuk informasi selengkapnya, lihat Apa itu peering VPC di dokumentasi VPC Amazon.

kerentanan

Kelemahan perangkat lunak atau perangkat keras yang membahayakan keamanan sistem.

W

cache hangat

Cache buffer yang berisi data saat ini dan relevan yang sering diakses. Instance database dapat membaca dari cache buffer, yang lebih cepat daripada membaca dari memori utama atau disk.

data hangat

Data yang jarang diakses. Saat menanyakan jenis data ini, kueri yang cukup lambat biasanya dapat diterima.

fungsi jendela

Fungsi SQL yang melakukan perhitungan pada sekelompok baris yang berhubungan dengan catatan saat ini. Fungsi jendela berguna untuk memproses tugas, seperti menghitung rata-rata bergerak atau mengakses nilai baris berdasarkan posisi relatif dari baris saat ini.

beban kerja

Kumpulan sumber daya dan kode yang memberikan nilai bisnis, seperti aplikasi yang dihadapi pelanggan atau proses backend.

aliran kerja

Grup fungsional dalam proyek migrasi yang bertanggung jawab atas serangkaian tugas tertentu. Setiap alur kerja independen tetapi mendukung alur kerja lain dalam proyek. Misalnya, alur kerja portofolio bertanggung jawab untuk memprioritaskan aplikasi, perencanaan gelombang, dan mengumpulkan metadata migrasi. Alur kerja portofolio mengirimkan aset ini ke alur kerja migrasi, yang kemudian memigrasikan server dan aplikasi.

CACING

Lihat menulis sekali, baca banyak.

WQF

Lihat Kerangka Kualifikasi Beban Kerja AWS.

tulis sekali, baca banyak (WORM)

Model penyimpanan yang menulis data satu kali dan mencegah data dihapus atau dimodifikasi. Pengguna yang berwenang dapat membaca data sebanyak yang diperlukan, tetapi mereka tidak dapat mengubahnya. Infrastruktur penyimpanan data ini dianggap tidak dapat diubah.

Z

eksploitasi zero-day

Serangan, biasanya malware, yang memanfaatkan kerentanan zero-day.

kerentanan zero-day

Cacat atau kerentanan yang tak tanggung-tanggung dalam sistem produksi. Aktor ancaman dapat menggunakan jenis kerentanan ini untuk menyerang sistem. Pengembang sering menyadari kerentanan sebagai akibat dari serangan tersebut.

aplikasi zombie

Aplikasi yang memiliki CPU rata-rata dan penggunaan memori di bawah 5 persen. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini.

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