AWS CDK で を使用して IaC プロジェクト TypeScript を作成するためのベストプラクティス

AWS 規範ガイダンス

AWS 規範ガイダンス: AWS CDK で を使用して IaC プロジェクト TypeScript を作成するためのベストプラクティス

Table of Contents

で AWS CDK を使用して IaC プロジェクトを作成 TypeScript するためのベストプラクティス

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

2024 年 2 月 (ドキュメント履歴)

このガイドでは、 AWS Cloud Development Kit (AWS CDK) を使用して大規模な Infrastructure as Code (IaC) プロジェクトを構築およびデプロイ TypeScript するための推奨事項とベストプラクティスについて説明します。 AWS CDK は、コードでクラウドインフラストラクチャを定義し、 を通じてそのインフラストラクチャをプロビジョニングするためのフレームワークです AWS CloudFormation。プロジェクト構造が明確に定義されていない場合、大規模なプロジェクトの AWS CDK コードベースを構築して管理するのは難しい場合があります。こうした課題に対処するために、大規模プロジェクトでアンチパターンを使用する組織もありますが、そうしたパターンはプロジェクトの進行を遅らせ、組織に悪影響を及ぼす他の問題を引き起こす可能性があります。例えば、アンチパターンは、開発者のオンボーディング、バグ修正、新機能の採用を複雑にし、遅らせる可能性があります。

このガイドでは、アンチパターンに代わる方法を示し、スケーラビリティ、テスト、セキュリティのベストプラクティスとの整合性を考慮してコードを整理する方法を説明します。このガイドを参考にして、IaC プロジェクトのコード品質を向上させ、ビジネスのアジリティを最大化することも可能です。このガイドは、アーキテクト、テクニカルリード、インフラストラクチャエンジニア、および大規模なプロジェクト向けに適切に設計された AWS CDK プロジェクトを構築しようとするその他の役割を対象としています。

目的

このガイドを通じて、次のようなターゲットを絞ったビジネス成果を達成できます。

- コスト削減 — を使用してAWS CDK、組織のセキュリティ、コンプライアンス、ガバナンス要件を満たす独自の再利用可能なコンポーネントを設計できます。また、組織全体でコンポーネントを簡単に共有できるため、デフォルトでベストプラクティスに沿った新しいプロジェクトを迅速に立ち上げることができます。
- 市場投入までの時間を短縮する — で使い慣れた機能を活用してAWS CDK、開発プロセスを高速化します。これにより、デプロイの再利用性が向上し、開発作業が軽減されます。
- デベロッパーの生産性の向上 — デベロッパーは使い慣れたプログラミング言語を使用してインフラストラクチャを定義できます。これにより、デベロッパーは AWSリソースを表現および維持できます。これにより、デベロッパーの効率とコラボレーションが向上します。

ベストプラクティス

このセクションでは、以下のベストプラクティスの概要を説明します。

- 大規模プロジェクトのコード編成
- 再利用可能なパターンの開発
- コンストラクトの作成または拡張
- TypeScript ベストプラクティスに従う
- セキュリティの脆弱性やフォーマットエラーのスキャン
- ドキュメントの作成と改善
- テスト駆動型の開発アプローチを採用
- コンストラクトにリリースとバージョン管理を使用する
- ライブラリのバージョン管理を強制する

大規模プロジェクトのコード編成

コード編成が重要な理由

大規模な AWS CDK プロジェクトでは、高品質で明確に定義された構造を持つことが重要です。プロジェクトが大きくなり、サポートの対象となる機能やコンストラクトの数が増えるにつれて、コードナビゲーションはより難しくなります。この難しさが生産性に影響を及ぼし、開発者のオンボーディングを遅らせることにもなりかねません。

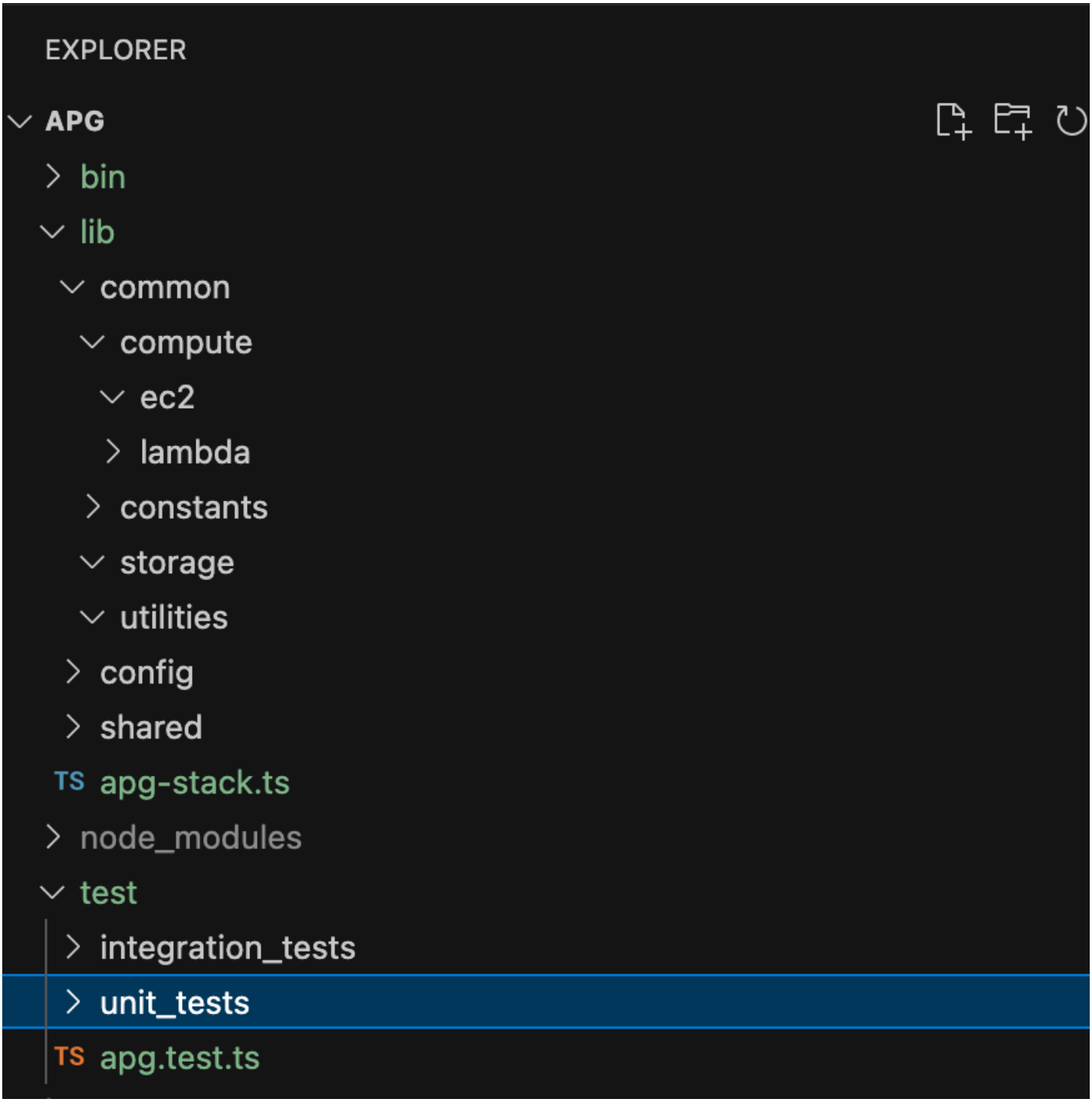
規模に見合ったコード編成の方法

高いレベルの柔軟性と可読性を持つコードを実現するには、コードを機能に基づいて論理的に分割することを推奨します。このような分割は、ほとんどのコンストラクトがさまざまなビジネスドメインで使用されているという事実を反映するものです。例えば、フロントエンドアプリケーションとバックエンドアプリケーションの両方に AWS Lambda 関数が必要で、同じソースコードを消費する可能性があります。ファクトリーでは、作成ロジックをクライアントに公開せずにオブジェクトを作成し、共通のインターフェイスを使用して新しく作成されたオブジェクトを参照できます。コードベースで整合性のある動作を作成するための効果的なパターンとして、ファクトリーを使用できます。さらに、ファクトリーは信頼できる単一のソースとしても機能し、コードの繰り返しを避け、トラブルシューティングを容易にします。

ファクトリーの仕組みをよりよく理解するために、自動車メーカーを例に考えてみましょう。自動車メーカーは、タイヤ製造に必要な知識やインフラストラクチャを持っている必要はありません。自動車メーカーはその専門知識をタイヤの専門メーカーに外注し、必要に応じてそのメーカーにタイヤを注文すればよいのです。同じ原則がコードにも当てはまります。例えば、高品質の Lambda 関数を構築できる Lambda ファクトリーを作成しておけば、Lambda 関数を作成する必要があるときはいつでも、コード内の Lambda ファクトリーを呼び出すことができます。同様に、これと同じアウトソーシングプロセスを使用してアプリケーションを切り離し、モジュール型コンポーネントを構築できます。

サンプルコードの構成

次の TypeScript サンプルプロジェクトには、次の図に示すように、すべてのコンストラクトまたは一般的な機能を保持できる共通フォルダが含まれています。



例えば、コンピューティングフォルダー (共通フォルダー内にあります) には、さまざまなコンピューティングコンストラクトのあらゆるロジックが格納されます。新任の開発者は、他のリソースに影響を与えることなく、新しいコンピューティングコンストラクトを簡単に追加できます。他のすべてのコンストラクトは、内部で新しいリソースを作成する必要はありません。これらのコンストラ

クトが共通コンストラクトファクトリーを呼び出すだけでよいのです。ストレージなど他のコンストラクトも同様に編成可能です。

設定には環境ベースのデータが含まれるため、ロジックを保存している共通フォルダーから切り離す必要があります。共有するフォルダー内には、共通の設定データを格納することを推奨します。また、ユーティリティフォルダーを使用して、すべてのヘルパー関数とクリーンアップスクリプトを提供することも推奨しています。

再利用可能なパターンの開発

ソフトウェアのデザインパターンは、ソフトウェア開発に共通する問題に対する再利用可能なソリューションです。これらは、ソフトウェアエンジニアがベストプラクティスに従った製品を開発するのに役立つガイドまたはパラダイムとして機能します。このセクションでは、 AWS CDK コードベースで使用できる 2 つの再利用可能なパターンの概要について説明します。抽象ファクトリーパターンと Chain of Responsibility パターンです。各パターンをブループリントとして使用し、コード内にある特定の設計上の問題に合わせてカスタマイズできます。デザインパターンの詳細については、Refactoring.Guru ドキュメントの「デザインパターン」を参照してください。

Abstract Factory

Abstract Factory パターンは、具体的なクラスを指定せずに関連オブジェクトや依存関係オブジェクトのファミリーを作成するためのインターフェイスを提供します。このパターンは、次のようなユースケースに適用されます。

- クライアントが、システム内のオブジェクトの作成方法や構成方法に依存しない場合
- システムが複数のオブジェクトファミリーで構成されており、これらのファミリーを一緒に使用できるように設計されている場合
- 特定の依存関係を構築するためのランタイム値が必要な場合

Abstract Factory パターンの詳細については、Refactoring.Guru ドキュメントの「 の Abstract Factory TypeScript」を参照してください。

次のコード例は、Abstract Factory パターンを使用して Amazon Elastic Block Store (Amazon EBS) ストレージファクトリを構築する方法を示しています。

```
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();
```

Chain of Responsibility

Chain of Responsibility とは、ハンドラー候補チェーンの 1 人がリクエストを処理するまで、ハンドラー候補チェーンに沿ってリクエストを渡せるようにする行動デザインパターンです。Chain of Responsibility パターンは、次のようなユースケースに適用されます。

- 実行時に決定された複数のオブジェクトがリクエストを処理する候補となる場合
- コードでハンドラーを明示的に指定しない場合
- レシーバーを明示的に指定せずに、複数のオブジェクトのうちの 1 つにリクエストを発行したい場合

責任連鎖パターンの詳細については、Refactoring.Guru ドキュメントの「Chain of Responsibility in TypeScript」を参照してください。

以下のコードは、Chain of Responsibility パターンを使用して、タスクの完了に必要な一連のアクションを構築する方法の例を示しています。

```
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);
    }
}
```

コンストラクトの作成または拡張

コンストラクトとは

コンストラクトは、 AWS CDK アプリケーションの基本的な構成要素です。コンストラクトは、Amazon Simple Storage Service (Amazon S3) バケットなどの単一の AWS リソースを表すことも、複数の AWS 関連リソースで構成される高レベルの抽象化にすることもできます。コンストラクトのコンポーネントには、関連するコンピューティング能力を持つワーカーキュー、またはモニタリングリソースとダッシュボードを持つスケジュールされたジョブが含まれることがあります。には、 コンストラクトライブラリと呼ばれる AWS コンストラクトのコレクション AWS CDK が含まれています。ライブラリには、すべての のコンストラクトが含まれています AWS のサービス。Construct Hub を使用して、、サードパーティー AWS、オープンソース AWS CDK コミュニティから追加のコンストラクトを見つけることができます。

コンストラクトのさまざまなタイプ

には 3 つの異なるタイプのコンストラクトがあります AWS CDK。

- L1 コンストラクト – レイヤー 1、または L1 コンストラクトは、 CloudFormationで定義されるリソースです。それ以上、それ以上ではありません。設定に必要なリソースは自分で用意しなければなりません。これらの L1 コンストラクトは非常に基本的なため、手動で設定する必要があります。L1 コンストラクトには`Cfn`プレフィックスがあり、 CloudFormation 仕様に直接対応しています。新しい AWS のサービス は、 CloudFormation がこれらのサービスをサポートする AWS CDK とすぐに でサポートされます。 CfnBucketは L1 コンストラクトの良い例です。このクラスは S3 バケットを表し、すべてのプロパティを明示的に設定する必要があります。L1 コンストラクトは、L2 または L3 コンストラクトが見つからない場合にのみ使用することをお勧めします。
- L2 コンストラクト— レイヤー 2 (L2) のコンストラクトには、共通の定型コードとグルーロジックがあります。これらのコンストラクトには便利なデフォルトが付属しており、それらについて知っておく必要のある知識の量を削減できます。L2 コンストラクトはインテントベースAPIsを使用してリソースを構築し、通常は対応する L1 モジュールをカプセル化します。L2 コンストラクトの良い例がバケットです。このクラスは、デフォルトのプロパティと bucket.addLifeCycleRule() などのメソッドを使用して S3 バケットを作成し、バケットにライフサイクルルールを追加します。
- L3 コンストラクト — レイヤー 3 (L3) のコンストラクトはパターンと呼ばれます。L3 コンストラクトは、 で一般的なタスクを完了するのに役立つように設計されており AWS、多くの場合、複数の種類のリソースが含まれます。これらは L2 コンストラクトよりもさらに限定された特定用途向けとなっていて、ある決まったユースケースに使用されます。例えば、aws-ecs-patterns.ApplicationLoadBalancedFargateService コンストラクトは、Application Load Balancer を使用する AWS Fargate コンテナクラスターを含むアーキテクチャを表します。もう 1 つの例は、aws-apigateway.LambdaRestApi コンストラクトです。このコンストラクトは、Lambda 関数によってAPIバックアップされる Amazon API Gateway を表します。

コンストラクトレベルが高くなるにつれて、これらのコンストラクトがどのように使用されるかについて、より多くの仮定がなされます。これにより、極めて特殊なユースケースに対して、より効果的なデフォルトをインターフェイスに提供できるようになります。

独自のコンストラクトを作成する方法

独自のコンストラクトを定義するには、特定の方法に従う必要があります。これは、すべてのコンストラクトが `Construct` クラスを拡張するためです。`Construct` クラスはコンストラクトツリーの構成要素です。コンストラクトは、`Construct` 基本クラスを拡張するクラスで実装されます。すべてのコンストラクトは、初期化時に次の 3 つのパラメータを取ります。

- スコープ – コンストラクトの親または所有者。スタックまたは別のコンストラクトのいずれかであり、コンストラクトツリー内の場所を決定します。通常は `this` (または Python の `self`) にパスしなければならず、それがスコープの現在のオブジェクトを表します。

- id — このスコープ内で一意でなければならない識別子です。識別子は、現在のコンストラクト内で定義されているすべての名前空間として機能し、リソース名や CloudFormation 論理 などの一意の ID を割り当てるために使用されますIDs。
- Props – コンストラクトの初期設定を定義する一連のプロパティ。

次の例は、コンストラクトを定義する方法を示しています。

```
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
  }
}
```

L2 コンストラクトの作成または拡張

L2 コンストラクトは「クラウドコンポーネント」を表し、コンポーネントの作成 CloudFormation に必要なすべてをカプセル化します。L2 コンストラクトには 1 つ以上の AWS リソースを含めることができ、コンストラクトは自由にカスタマイズできます。L2 コンストラクトを作成または拡張する利点は、コードを再定義せずに CloudFormation スタック内のコンポーネントを再利用できることです。コンストラクトをクラスとしてインポートするだけで済みます。

既存のコンストラクトと「is a」関係がある場合は、既存のコンストラクトを拡張してデフォルトの機能を追加できます。既存の L2 コンストラクトのプロパティを再利用するのがベストプラクティスです。コンストラクターでプロパティを直接変更することで、プロパティを上書きできます。

次の例は、ベストプラクティスに沿って、`s3.Bucket` という既存の L2 コンストラクトを拡張する方法を示しています。この拡張は、`versioned`、`publicReadAccess`、`blockPublicAccess` のようなデフォルトプロパティを設定し、この新しいコンストラクトから作成されたすべてのオブジェクト (この例では S3 バケット) に、これらのデフォルト値が常に設定されるようにします。

```
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
    });
  }
}
```

L3 コンストラクトの作成

既存のコンストラクトコンポジションと「has a」関係がある場合は、コンポジションが適しています。コンポジションとは、他の既存のコンストラクトに独自のコンストラクトを構築することです。独自のパターンを作成して、すべてのリソースとそのデフォルト値を、共有可能な単一上位レベルの L3 コンストラクトにカプセル化できます。独自の L3 コンストラクト (パターン) を作成する利点は、コードを再定義しなくてもコンポーネントをスタックで再利用できることです。コンストラクトをクラスとしてインポートするだけで済みます。これらのパターンは、消費者が共通のパターンに基づいて、限られた知識で複数のリソースを簡潔にプロビジョニングできるように設計されています。

次のコード例では、 という AWS CDK コンストラクトを作成しますExampleConstruct。このコンストラクトをテンプレートとして使用して、クラウドコンポーネントを定義できます。

```
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
  }
}
```

次の例は、新しく作成されたコンストラクトを AWS CDK アプリケーションまたはスタックにインポートする方法を示しています。

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

次の例は、基本クラスから拡張したコンストラクトのインスタンスをインスタンス化する方法を示しています。

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

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

詳細については、AWS CDK ワークショップドキュメントの AWS CDK 「ワークショップ」を参照してください。

エスケープハッチ

でエスケープハッチを使用して抽象化レベル AWS CDK を上げると、より低いレベルのコンストラクトにアクセスできます。エスケープハッチは、 の最新バージョンでは公開されていない AWS が、 で利用可能な機能のコンストラクトを拡張するために使用されます CloudFormation。

次のようなシナリオでは、エスケープハッチの使用を推奨します。

- AWS のサービス 機能は を通じて使用できますが CloudFormation、Constructコンストラクトはありません。
- AWS のサービス 機能は を通じて利用 CloudFormation でき、サービスのConstructコンストラクトがありますが、まだその機能は公開されていません。Construct コンストラクトは「手動で」開発されるため、 CloudFormation リソースコンストラクトより遅れることがあります。

次のコード例は、エスケープハッチを使用する一般的なユースケースを示しています。この例では、上位レベルのコンストラクトにはまだ実装されていない機能を、オートスケーリング LaunchConfiguration 用の httpPutResponseHopLimit に追加するためのものです。

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

前述のコード例は、次のワークフローを示しています。

1. L2 コンストラクトを使用して AutoScalingGroup を定義します。L2 コンストラクトは の更新をサポートしていないためhttpPutResponseHopLimit、エスケープハッチを使用する必要があります。
2. L2 AutoScalingGroup コンストラクトの node.defaultChild プロパティにアクセスし、CfnLaunchConfiguration のリソースとしてキャストします。
3. これで、L1 CfnLaunchConfiguration の launchConfig.metadataOptions のプロパティを設定できるようになりました。

カスタムリソース

カスタムリソースを使用して、スタックを作成、更新 (カスタムリソースを変更した場合) 、または削除するたびに CloudFormation 実行されるテンプレートにカスタムプロビジョニングロジックを記述できます。たとえば、 で使用できないリソースを含める場合は、カスタムリソースを使用できます AWS CDK。この方法により、すべての関連リソースを 1 つのスタックで管理できます。

カスタムリソースを構築するには、リソースの CREATE、UPDATE、DELETE ライフサイクルイベントに応答する Lambda 関数を書き込む必要があります。カスタムリソースが 1 回のAPI呼び出しのみを行う必要がある場合は、 AwsCustomResourceコンストラクトの使用を検討してください。これにより、 CloudFormation デプロイ中に任意のSDK呼び出しを実行できます。それ以外の場合は、必要な作業を実行するための独自の Lambda 関数を作成することを推奨します。

カスタムリソースの詳細については、 CloudFormation ドキュメントの「カスタムリソース」を参照してください。カスタムリソースの使用方法の例については、 のカスタムリソースリポジトリを参照してください GitHub。

次の例は、カスタムリソースクラスを作成して Lambda 関数を開始し、 CloudFormation 成功または失敗シグナルを送信する方法を示しています。

```
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();
    }
}
```

次の例は、カスタムリソースの主なロジックです。

```
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)
```

次の例は、 AWS CDK スタックがカスタムリソースを呼び出す方法を示しています。

```
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();
```

TypeScript ベストプラクティスに従う

TypeScript は、 の機能を拡張する言語です JavaScript。これは厳密に型指定されたオブジェクト指向の言語です。を使用して TypeScript 、コード内で渡されるデータのタイプを指定でき、タイプが一致しない場合にエラーをレポートできます。このセクションでは、 TypeScript のベストプラクティスの概要を説明します。

データの説明

を使用して TypeScript 、コード内のオブジェクトと関数のシェイプを記述できます。any タイプを使用することは、変数の型検査をオプトアウトすることと同じです。コードに any を使用しないことを推奨します。以下はその例です。

```
type Result = "success" | "failure"
function verifyResult(result: Result) {
    if (result === "success") {
        console.log("Passed");
    } else {
        console.log("Failed")
    }
}
```

列挙型を使用する

列挙型を使用して名前付き定数のセットを定義し、さらにコードベースで再利用できる標準を定義できます。列挙型をグローバルレベルで一度エクスポートし、他のクラスにその列挙型をインポートして使用することを推奨します。コードベースでイベントをキャプチャするための一連のアクションを作成するとします。 TypeScript には、数値と文字列ベースの列挙型の両方が用意されています。次の例では列挙型を使用します。

```
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)
```

インターフェイスを使用する

インターフェイスはクラスに関する契約です。契約を作成する場合、ユーザーはその契約を順守する必要があります。次の例では、インターフェイスを使用して props を標準化し、このクラスを使用する際には、予想されるパラメータを呼び出し元が確実に提供できるようにしています。

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

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

}

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

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

```
})
```

プロパティの中には、オブジェクトが最初に作成されたときにしか変更できないものもあります。これを指定するには、次の例で示しているように、プロパティ名の前に readonly を入力します。

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

インターフェイスを拡張する

インターフェイスを拡張すると、インターフェイス間でプロパティをコピーする必要がなくなるため、重複が減ります。また、コードの読者がアプリケーション内の関係を容易に理解できるようになります。

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

空のインターフェイスを回避する

空のインターフェイスはリスクを生じる可能性があるため、使用しないことをお勧めします。次の例では、 という空のインターフェイスがありますBucketProps。myS3Bucket1 と myS3Bucket2 のオブジェクトはどちらも有効ですが、インターフェイスは契約を強制しないため、異なる標準に従っています。次のコードはプロパティをコンパイルして出力しますが、これによりアプリケーションに矛盾が生じます。

```
interface BucketProps {}

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

```
}

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

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

ファクトリーを使用する

Abstract Factory パターンでは、インターフェイスはクラスを明示的に指定しなくても、関連するオブジェクトのファクトリーを作成します。例えば、Lambda 関数を作成するための Lambda ファクトリーを作成できます。コンストラクト内に新しい Lambda 関数を作成する代わりに、作成プロセスをファクトリに委任します。この設計パターンの詳細については、Refactoring.Guru ドキュメントの「 の抽象ファクト TypeScriptリー」を参照してください。

プロパティにデストラクチャリングを使用する

6 (ES6) ECMAScript で導入された破壊は、配列またはオブジェクトから複数のデータを抽出し、独自の変数に割り当てる JavaScript 機能です。

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

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

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

標準の命名規則を定義する

命名規則を適用することでコードベースの整合性が保たれ、変数の命名方法を考える際のオーバーヘッドが軽減されます。次の構成を推奨します。

- 変数名と関数名 camelCase には を使用します。

- クラス名とインターフェイス名 PascalCase には を使用します。
- インターフェイスメンバー camelCase には を使用します。
- タイプ名と列挙型名 PascalCase には を使用します。
- でファイルに名前を付ける camelCase （ 例: `ebsVolumes.tsx`または `storage.tsb` ）

var キーワードを使用しない

`let` ステートメントは、 でローカル変数を宣言するために使用されます TypeScript。これは `var`キーワードに似ていますが、 `var` キーワードと比較してスコープに制限があります。`let` のあるブロック内で宣言された変数は、そのブロック内でのみ使用できます。`var` キーワードはブロックスコープにすることはできません。つまり、特定のブロック (で表される{}) の外部でアクセスできますが、定義されている関数の外部ではアクセスできません。`var` 変数を再宣言および更新できます。`var` キーワードを使用しないことがベストプラクティスです。

ESLint と Prettier の使用を検討する

ESLint はコードを静的に分析して問題をすばやく検出します。ESLint を使用して、コードの外観や動作を定義する一連のアサーション (lint ルールと呼ばれる) を作成できます。 には、コードの改善に役立つ自動修正の提案ESLintもあります。最後に、 ESLintを使用して、共有プラグインから lint ルールをロードできます。

Prettier は、さまざまなプログラミング言語をサポートする有名なコードフォーマッタです。Prettier を使用してコードスタイルを設定できるため、コードを手動でフォーマットしないで済みます。インストール後、`package.json` ファイルを更新し `npm run format` と `npm run lint` のコマンドを実行できます。

次の例は、プロジェクトの ESLintと Prettier フォーマッターを有効にする方法を示しています 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)'"
}
```

アクセス修飾子を使用する

のプライベート修飾子は、可視性を同じクラスのみ TypeScript に制限します。プライベート修飾子をプロパティまたはメソッドに追加すると、同じクラス内でそのプロパティまたはメソッドにアクセスできます。

パブリック修飾子を使用すると、クラスのプロパティとメソッドにあらゆる場所からアクセスできます。プロパティとメソッドにアクセス修飾子を指定しない場合、デフォルトでパブリック修飾子が使用されます。

保護された修飾子を使うと、同一クラス内およびサブクラス内でクラスのプロパティとメソッドにアクセスできるようになります。 AWS CDK アプリケーションでサブクラスを作成する場合は、保護された修飾子を使用します。

ユーティリティタイプを使用する

のユーティリティタイプ TypeScript は、既存のタイプに対して変換とオペレーションを実行する事前定義されたタイプ関数です。これにより、既存のタイプに基づいて新しいタイプを作成できます。例えば、プロパティの変更や抽出、プロパティのオプションまたは必須化、タイプのイミュータブルバージョンの作成を行うことができます。ユーティリティタイプを使用すると、より正確なタイプを定義し、コンパイル時に潜在的なエラーをキャッチできます。

部分 < タイプ >

Partial は、入力タイプのすべてのメンバーをオプションTypeとしてマークします。このユーティリティは、特定のタイプのすべてのサブセットを表すタイプを返します。Partial の例を次に示します。

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

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

必須<Type>

Required は とは逆の処理を行いますPartial。これにより、入力タイプのすべてのメンバーがオプションTypeでなくなります (つまり、必須) 。Required の例を次に示します。

```
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
};
```

セキュリティの脆弱性やフォーマットエラーのスキャン

Infrastructure as Code (IaC) と自動化は、企業にとって不可欠なものとなっています。IaC は非常に堅牢であるため、セキュリティリスクを管理する責任は大きくなります。IaC の一般的なセキュリティリスクには次のようなものがあります。

- 許可が過剰 AWS Identity and Access Management （ IAM) の権限
- オープンなセキュリティグループ
- 暗号化されていないリソース
- オンになっていないアクセスログ

セキュリティのアプローチとツール

以下のセキュリティアプローチの実装をお勧めします。

- 開発中の脆弱性検出 — ソフトウェアパッチの開発と配布は複雑なため、本番稼働環境での脆弱性の修復には費用と時間がかかります。さらに、本番稼働環境の脆弱性には悪用されるリスクもあります。本番稼働環境にリリースする前に脆弱性を検出して修正できるように、IaC リソースでコードスキャンを使用することを推奨します。
- コンプライアンスと自動修復 — AWS はマネージドルール AWS Config を提供します。これらのルールは、コンプライアンスを強制し、AWS Systems Manager 自動化を使用して自動修復を試みるのに役立ちます。 AWS Config ルールを使用してカスタムオートメーションドキュメントを作成して関連付けることもできます。

一般的な開発ツール

このセクションで説明するツールは、独自のカスタムルールを持つ組み込み機能の拡張に役立ちます。カスタムルールを組織の標準に合わせることをお勧めします。以下は一般的な開発ツールの一部です。

- cfn-nag を使用して、 CloudFormation テンプレート内の許容IAMルールやパスワードリテラルなどのインフラストラクチャのセキュリティ問題を特定します。詳細については、Steligent の GitHub 「cfn-nag リポジトリ」を参照してください。
- cfn-nag にヒントを得た cdk-nag を使用すると、特定のスコープ内のコンストラクトが、定義された一連のルールに準拠しているかどうかを検証できます。cdk-nag はルール抑制やコンプライアンスレポートにも使用できます。cdk-nag ツールは、 の側面を拡張してコンストラクトを検証します AWS CDK。詳細については、 AWS DevOps ブログの「Manage application security and compliance with the AWS Cloud Development Kit (AWS CDK) and cdk-nag」を参照してください。
- オープンソースツールの Checkov を使用して、IaC 環境で静的分析を実行します。Checkov は、Kubernetes、Terraform、または でインフラストラクチャコードをスキャンすることで、クラウドの設定ミスを特定するのに役立ちます CloudFormation。Checkov を使用して、JSON、、JUnit XMLなど、さまざまな形式の出力を取得できますCLI。Checkov は、動的なコード依存関係を示すグラフを作成することで、変数を効果的に処理できます。詳細については、Bridgecrew の GitHub 「Checkov リポジトリ」を参照してください。
- TFLint を使用してエラーや廃止された構文をチェックし、ベストプラクティスを適用できるようにします。はプロバイダー固有の問題を検証しないTFLint場合があります。の詳細については TFLint、Terraform Linters のリポジトリを参照してください GitHub TFLint。
- Amazon Q Developer を使用してセキュリティスキャンを実行します。統合開発環境 (IDE) で使用すると、Amazon Q Developer は AI を活用したソフトウェア開発支援を提供します。コードに関するチャット、インラインコード補完の提供、新しいコードの生成、コードのセキュリティ脆弱性のスキャン、コードのアップグレードと改善を行うことができます。

ドキュメントの作成と改善

プロジェクトの成功にはドキュメントが不可欠です。ドキュメントはコードの仕組みを説明するだけでなく、開発者がアプリケーションの特徴や機能をよりよく理解するのにも役立ちます。質の高いドキュメントを作成・改善することで、ソフトウェア開発プロセスの強化や、高品質なソフトウェアの維持が可能となり、開発者間の知識移転にも役立ちます。

ドキュメントには、コード内のドキュメントと、コードに関するサポートドキュメントという 2 つのカテゴリがあります。コード内のドキュメントはコメント形式です。コードに関するサポートドキュメントは、READMEファイルと外部ドキュメントです。コード自体は理解しやすいため、開発者がドキュメントをオーバーヘッドと考えるのは珍しくありません。小規模なプロジェクトにはそうした考えが当てはまるかもしれませんが、複数のチームが関与する大規模なプロジェクトではドキュメントは不可欠です。

コードの作成者は、その機能を十分に理解しているため、ドキュメントを記述するのがベストプラクティスです。開発者は、個別のサポートドキュメントの管理から生じる追加のオーバーヘッドに苦労することがあります。この課題を解決するために、開発者はコードにコメントを追加し、それらのコメントを自動的に抽出して、すべてのバージョンのコードとドキュメントを同期させることができます。

開発者がコードからコメントを抽出してドキュメントを生成する際に役立つ、さまざまなツールがあります。このガイドでは、 AWS CDK コンストラクトの推奨ツール TypeDocとして に焦点を当てています。

コンストラクトに AWS CDK コードドキュメントが必要な理由

AWS CDK 共通コンストラクトは、組織内の複数のチームによって作成され、消費のために異なるチーム間で共有されます。優れたドキュメントがあれば、コンストラクトライブラリーの利用者は最小限の労力で容易にコンストラクトを統合し、インフラストラクチャを構築できます。すべてのドキュメントを同期させることは大変な作業です。ドキュメントは、 TypeDocライブラリを使用して抽出されるコード内に保持することをお勧めします。

コンストラクトライブラリ TypeDoc での AWS の使用

TypeDoc はドキュメントジェネレーターです TypeScript。 TypeDoc を使用して TypeScript ソースファイルを読み取って、それらのファイル内のコメントを解析し、コードのドキュメントを含む静的サイトを生成できます。

次のコードは、 AWS コンストラクトライブラリ TypeDoc と統合し、 の `package.json` ファイルに次のパッケージを追加する方法を示していますdevDependencies。

```
{

  "devDependencies": {

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

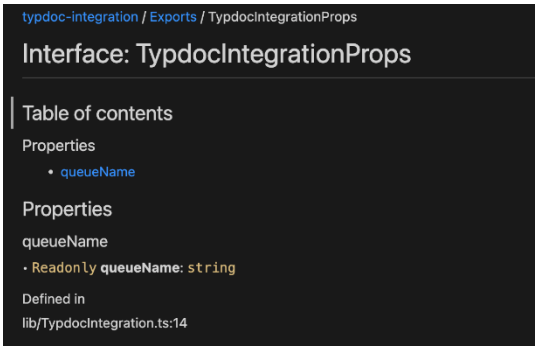
```
  },

}
```

CDK ライブラリフォルダtypedoc.jsonに を追加するには、次のコードを使用します。

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

README ファイルを生成するには、 AWS CDK コンストラクトライブラリプロジェクトのルートディレクトリで npx typedoc コマンドを実行します。

次のサンプルドキュメントは によって生成されます TypeDoc。



TypeDoc 統合オプションの詳細については、 TypeDoc ドキュメントの「Doc Comments」を参照してください。

テスト駆動型の開発アプローチを採用

では、テスト駆動型開発 (TDD) アプローチに従うことをお勧めします AWS CDK。 TDDは、コードを指定して検証するためのテストケースを開発するソフトウェア開発アプローチです。簡単に言うと、まず機能ごとにテストケースを作成し、テストが失敗したら、テストをパスする新しいコードを書き、コードをシンプルでバグのないものにします。

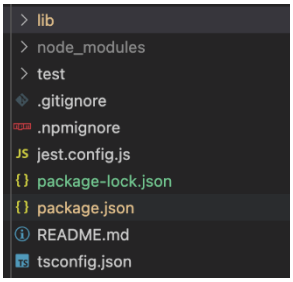
を使用してTDD、最初にテストケースを記述できます。これにより、リソースにセキュリティポリシーを適用し、プロジェクト固有の命名規則に従うという点で、さまざまな設計上の制約があるインフラストラクチャを検証できます。 AWS CDK アプリケーションをテストするための標準的なアプローチは、 AWS CDK アサーションモジュールと、Jestfor TypeScript や JavaScript orpytestfor Python などの一般的なテストフレームワークを使用することです。

アプリケーション用に記述できるテストには 2 つのカテゴリがあります AWS CDK 。

- きめ細かなアサーションを使用して、「このリソースにはこの値を持つこのプロパティがあります」など、生成された CloudFormation テンプレートの特定の側面をテストします。これらのテストでは回帰を検出でき、 を使用して新機能を開発する場合にも役立ちます TDD (最初にテストを記述してから、正しい実装を記述して合格させます) 。きめ細かいアサーションは、最も多く作成するテストです。
- スナップショットテストを使用して、以前に保存したベースライン CloudFormation テンプレートに対して合成されたテンプレートをテストします。スナップショットテストでは、リファクタリングされたコードが元のコードとまったく同じように動作することを確認できるため、自由にリファクタリングできます。変更が意図的なものであった場合は、将来のテストのために新しいベースラインを受け入れることができます。ただし、 AWS CDK アップグレードによって合成されたテンプレートが変更される可能性があるため、実装が正しいことを確認するためにスナップショットのみに依存することはできません。

ユニットテスト

このガイドでは、 TypeScript 特に のユニットテスト統合に焦点を当てています。テストを有効にするには、 `package.json` `ts-jest`ファイルに 、`@types/jest`、 のライブラリがあることを確認します`devDependencies`。これらのパッケージを追加するには、`cdk init lib --language=typescript` のコマンドを実行します。上記のコマンドを実行すると、次の構造が表示されます。



次のコードは、Jest ライブラリで有効になっている`package.json`ファイルの例です。

```
{
  ...
  "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",
    ...
  }
}
```

テストフォルダの下にテストケースを書き込めます。次の例は、 AWS CodePipeline コンストラクトのテストケースを示しています。

```
import { Stack } from 'aws-cdk-lib';
import { Template } from 'aws-cdk-lib/assertions';
import * as CodePipeline from 'aws-cdk-lib/aws-codepipeline';
import * as CodePipelineActions from 'aws-cdk-lib/aws-codepipeline-actions';
import { MyPipelineStack } from '../lib/my-pipeline-stack';
test('Pipeline Created with GitHub Source', () => {
  // ARRANGE
  const stack = new Stack();
  // ACT
  new MyPipelineStack(stack, 'MyTestStack');
  // ASSERT
  const template = Template.fromStack(stack);
  // Verify that the pipeline resource is created
  template.resourceCountIs('AWS::CodePipeline::Pipeline', 1);
  // Verify that the pipeline has the expected stages with GitHub source
  template.hasResourceProperties('AWS::CodePipeline::Pipeline', {
    Stages: [
      {
        Name: 'Source',
        Actions: [
          {
            Name: 'SourceAction',
            ActionTypeId: {
              Category: 'Source',
              Owner: 'ThirdParty',
              Provider: 'GitHub',
              Version: '1'
            },
            Configuration: {
              Owner: {
                'Fn::Join': [
```

```
                '',
                [
                  '{{resolve:secretsmanager:',
                  {
                    Ref: 'GitHubTokenSecret'
                  },
                  ':SecretString:owner}}'
                ]
              ]
            },
            Repo: {
              'Fn::Join': [
                '',
                [
                  '{{resolve:secretsmanager:',
                  {
                    Ref: 'GitHubTokenSecret'
                  },
                  ':SecretString:repo}}'
                ]
              ]
            },
            Branch: 'main',
            OAuthToken: {
              'Fn::Join': [
                '',
                [
                  '{{resolve:secretsmanager:',
                  {
                    Ref: 'GitHubTokenSecret'
                  },
                  ':SecretString:token}}'
                ]
              ]
            }
          },
          OutputArtifacts: [
            {
              Name: 'SourceOutput'
            }
          ],
          RunOrder: 1
        }
      ]
```

```
      },
      {
        Name: 'Build',
        Actions: [
          {
            Name: 'BuildAction',
            ActionTypeId: {
              Category: 'Build',
              Owner: 'AWS',
              Provider: 'CodeBuild',
              Version: '1'
            },
            InputArtifacts: [
              {
                Name: 'SourceOutput'
              }
            ],
            OutputArtifacts: [
              {
                Name: 'BuildOutput'
              }
            ],
            RunOrder: 1
          }
        ]
      }
      // Add more stage checks as needed
    ]
  });
  // Verify that a GitHub token secret is created
  template.resourceCountIs('AWS::SecretsManager::Secret', 1);
});
);
```

テストを実行するには、プロジェクト内の `npm run test` コマンドを実行します。テストは次の結果を返します。

```
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
```

テストケースの詳細については、「 AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」の「コンストラクトのテスト」を参照してください。

統合テスト

AWS CDK コンストラクトの統合テストは、 `integ-tests`モジュールを使用して含めることもできます。統合テストは AWS CDK アプリケーションとして定義する必要があります。統合テストと AWS CDK アプリケーションの間には one-to-one関係があるはずです。詳細については、「 AWS CDK APIリファレンス」の「 integ-tests-alpha モジュール」を参照してください。

コンストラクトにリリースとバージョン管理を使用する

のバージョン管理 AWS CDK

AWS CDK 共通コンストラクトは、複数のチームによって作成され、組織全体で共有して使用できます。通常、デベロッパーは共通の AWS CDK コンストラクトで新機能やバグ修正をリリースします。これらのコンストラクトは、依存関係の一部としてアプリケーションまたは他の既存の AWS CDK コンストラクトによって AWS CDK 使用されます。このため、開発者はコンストラクトを適切なセマンティックバージョンで個別に更新しリリースすることが重要です。ダウンストリーム AWS CDK アプリケーションやその他の AWS CDK コンストラクトは、新しくリリースされた AWS CDK コンストラクトバージョンを使用するように依存関係を更新できます。

セマンティックバージョニング (Semver) とは、コンピュータソフトウェアに一意のソフトウェア番号を付与するための一連のルールまたはメソッドです。バージョンは次のように定義されます。

- MAJOR バージョンは、互換性のないAPI変更または重大な変更で構成されます。
- MINOR バージョンは、下位互換性のある方法で追加された機能で構成されます。
- PATCH バージョンは、下位互換性のあるバグ修正で構成されます。

セマンティックバージョニングの詳細については、セマンティックバージョニングドキュメントの「セマンティックバージョニング仕様 (SemVer) 」を参照してください。

コンストラクトの AWS CDK リポジトリとパッケージ化

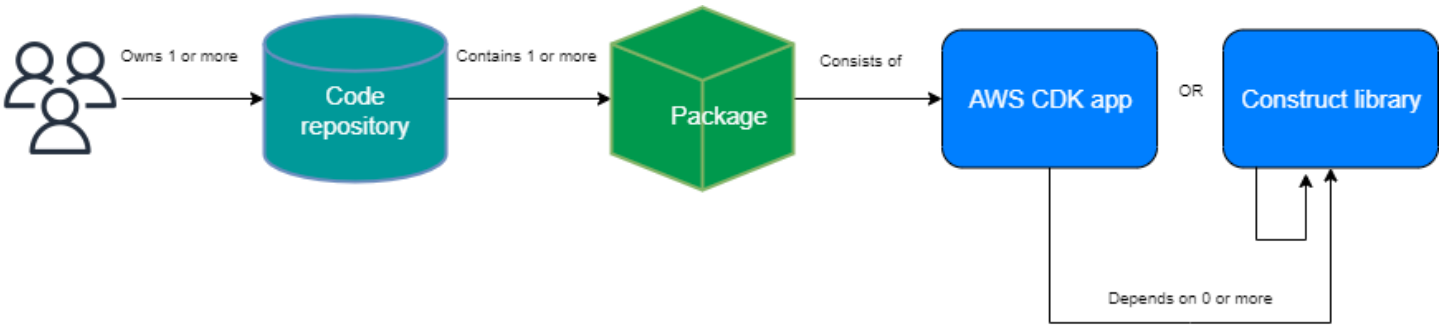
AWS CDK コンストラクトは異なるチームによって開発され、複数の AWS CDK アプリケーションで使用されるため、 AWS CDK コンストラクトごとに個別のリポジトリを使用できます。これはアクセスコントロールの強化にも役立ちます。各リポジトリには、同じ AWS CDK コンストラクトに関連するすべてのソースコードとそのすべての依存関係を含めることができます。1 つのアプリケー

ション (AWS CDK コンストラクト) を 1 つのリポジトリに保持することで、デプロイ中の変更の影響範囲を減らすことができます。

はインフラストラクチャをデプロイするための CloudFormation テンプレートを生成する AWS CDK だけでなく、Lambda 関数や Docker イメージなどのランタイムアセットをバンドルし、インフラストラクチャと一緒にデプロイします。インフラストラクチャを定義するコードとランタイムロジックを実装するコードを 1 つのコンストラクトに結合できるだけでなく、ベストプラクティスです。これら 2 種類のコードは、別々のリポジトリに置く必要も、別々のパッケージに置く必要もありません。

リポジトリの境界を越えてパッケージを使用するには、npm PyPiや Maven Central に似たプライベートパッケージリポジトリが必要ですが、これは組織内で使用できます。また、パッケージを構築し、テストし、プライベートパッケージリポジトリに公開するリリースプロセスも必要です。ローカル仮想マシン (VM) または Amazon S3 を使用して、 PyPi サーバーなどのプライベートリポジトリを作成できます。プライベートパッケージレジストリを設計または作成するときは、高い可用性とスケーラビリティによってサービスが中断されるリスクを考慮することが重要です。パッケージを保存するためにクラウドでホストされているサーバーレスマネージドサービスは、メンテナンスオーバーヘッドを大幅に削減できます。例えば、 を使用して、ほとんどの一般的なプログラミング言語のパッケージAWS CodeArtifactをホストできます。 CodeArtifact を使用して外部リポジトリ接続を設定し、その内部でレプリケートすることもできます CodeArtifact。

パッケージリポジトリ内のパッケージへの依存関係は、言語のパッケージマネージャー (TypeScript または JavaScript アプリケーションの npm など) によって管理されます。パッケージマネージャーは、アプリケーションが依存するすべてのパッケージの特定のバージョンを記録し、次の図のように制御された方法で依存関係をアップグレードできるようにすることで、ビルドが繰り返し可能であることを確認します。

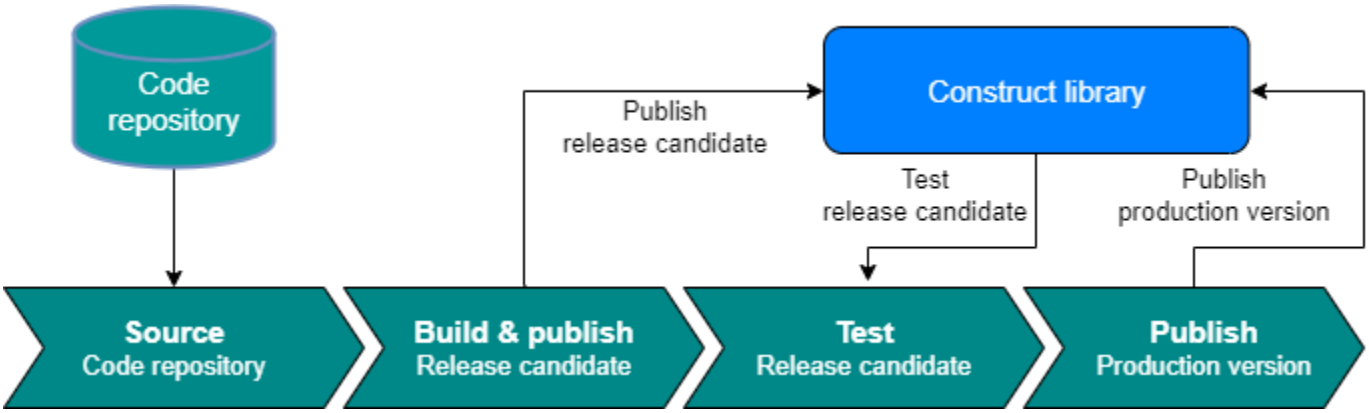


のコンストラクトリリース AWS CDK

新しい AWS CDK コンストラクトバージョンを構築してリリースするには、独自の自動パイプラインを作成することをお勧めします。プルリクエストの承認プロセスを適切に行い、ソースコードを

コミットしてリポジトリのメインブランチにプッシュすれば、パイプラインはリリース候補バージョンを構築· 作成できるようになります。そのバージョンは、本番稼働対応バージョンをリリースする前に にプッシュ CodeArtifact してテストできます。必要に応じて、コードをメインブランチとマージする前に、新しい AWS CDK コンストラクトバージョンをローカルでテストできます。これにより、パイプラインから本番稼働環境に対応したバージョンがリリースされます。共有コンストラクトやパッケージは、あたかも一般に公開されているかのように、利用側アプリケーションとは独立してテストする必要があることを考慮します。

次の図は、 AWS CDK バージョンリリースパイプラインの例を示しています。



以下のサンプルコマンドを使用すると npm パッケージのビルド、テスト、公開が可能です。まず、以下のコマンドを実行してアーティファクトリポジトリにサインインします。

```
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>
```

以下のステップを実行します。

1. package.json ファイル npm install に基づいて必要なパッケージをインストールする
2. リリース候補バージョン npm version prerelease --preid rc を作成する
3. npm パッケージ npm run build をビルドする
4. npm パッケージ npm run test をテストする
5. npm パッケージ npm publish を公開する

ライブラリのバージョン管理を強制する

ライフサイクル管理は、 AWS CDK コードベースを維持する場合の重要な課題です。例えば、バージョン 1.97 で AWS CDK プロジェクトを開始し、後でバージョン 1.169 が利用可能になるとします。バージョン 1.169 には新機能とバグ修正が含まれていますが、古いバージョンを使用してインフラストラクチャをデプロイしています。現在、新しいバージョンでは重大な変更が加えられる可能性があるため、このギャップが拡大するにつれ、コンストラクトの更新が難しくなります。環境に多くのリソースがある場合、これは難しい場合があります。このセクションで紹介するパターンは、自動化を使用して AWS CDK ライブラリのバージョンを管理するのに役立ちます。このパターンのワークフローは次のとおりです。

1. 新しい CodeArtifact Service Catalog 製品を起動すると、 AWS CDK ライブラリのバージョンとその依存関係が `package.json` ファイルに保存されます。
2. すべてのリポジトリを追跡する共通のパイプラインをデプロイして、重大な変更がない場合はリポジトリに自動アップグレードを適用できるようにします。
3. AWS CodeBuild ステージは依存関係ツリーをチェックし、重大な変更を探します。
4. パイプラインは機能ブランチを作成し、エラーがないことを確認するために新しいバージョンで `cdk synth` を実行します。
5. 新しいバージョンをテスト環境にデプロイし、最後に統合テストを実行してデプロイが正常であることを確認します。
6. 2 つの Amazon Simple Queue Service (Amazon SQS) キューを使用して、スタックを追跡できます。ユーザーは例外キューのスタックを手動で確認することで、重大な変更に対処できます。統合テストをパスしたアイテムは、マージおよびリリースが可能となります。

よくある質問

どのような問題が TypeScript 解決できますか？

通常、自動テストを作成し、コードが想定どおりに動作することを手動で検証し、最後に別の人にコードを検証してもらうことで、コードのバグを排除できます。プロジェクトのすべての部分間の接続を検証するには時間がかかります。検証プロセスを高速化するには、 のようなタイプチェック言語を使用してコード検証を自動化し TypeScript 、開発中に即座にフィードバックを提供できます。

を使用する理由 TypeScript

TypeScript は、 JavaScript コードを簡素化し、読み取りとデバッグを容易にするオープンソースの言語です。 は、静的チェックなどの JavaScript IDEs とプラクティス向けに、生産性の高い開発ツール TypeScript も提供します。さらに、 TypeScript には ECMAScript 6 (ES6) の利点があり、生産性を高めることができます。最後に、 TypeScript は、デベロッパーがコードをタイプチェック JavaScript して書き込むときによく遭遇する厄介なバグを回避するのに役立ちます。

AWS CDK または を使うべきです CloudFormationか？

組織で を活用するための開発専門知識がある場合はAWS CloudFormation、 AWS Cloud Development Kit (AWS CDK)の代わりに を使用することをお勧めしますAWS CDK。これは、プログラミング言語と OOP の概念を使用できるため CloudFormation、 AWS CDKは よりも柔軟であるためです。を使用して CloudFormation 、順序的かつ予測可能な方法でAWSリソースを作成できることに注意してください。では CloudFormation、リソースは JSON 形式または YAML 形式を使用してテキストファイルに書き込まれます。

AWS CDK が新しく起動された をサポートしていない場合はどうなりますかAWS のサービス？

raw オーバーライドまたは CloudFormation カスタムリソース を使用できます。

でサポートされているさまざまなプログラミング言語は何ですかAWS CDK？

AWS CDK は JavaScript、、、Python TypeScript、Java、C#、Go (デベロッパープレビュー) で一般利用可能です。

AWS CDK コストはいくらですか？

には追加料金はかかりませんAWS CDK。使用時に作成されるAWSリソース (Amazon EC2 インスタンスや Elastic Load Balancing ロードバランサーなど) はAWS CDK、手動で作成した場合と同じ方法で支払います。使用した分だけをお支払いいただきます。最低料金や前払いの義務はありません。

次のステップ

AWS Cloud Development Kit (AWS CDK) で を使用して構築を開始することをお勧めします TypeScript。詳細については、「AWS CDK Immersion Day Workshop」を参照してください。

リソース

リファレンス

- AWS ソリューション構築 (AWS ソリューション)
- AWS Cloud Development Kit (AWS CDK) (GitHub)
- AWS コンストラクトライブラリ API リファレンス (AWS CDK リファレンスドキュメント)
- AWS CDK リファレンスドキュメント (AWS CDK リファレンスドキュメント)
- AWS CDK Immersion Day Workshop (AWS Workshop Studio)

ツール

- cdk-nag (GitHub)
- TypeScript ESLint (TypeScript ESLint ドキュメント)

ガイドとパターン

- AWS ソリューション構築パターン (AWS ドキュメント)

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、RSS フィード をサブスクライブできます。

変更	説明	日付
コードの更新	TypeScript 「ベストプラクティスに従う」セクションのコードサンプルを更新しました。	2024 年 2 月 16 日
セクションを追加する	ユーティリティタイプの使用と統合テストのセクションを追加しました。	2024 年 1 月 10 日
マイナーな更新	L3 コンストラクトを作成するコード例を更新しました。	2023 年 6 月 16 日
初版発行	—	2022 年 12 月 8 日

AWS 規範的ガイダンスの用語集

以下は、 AWS 規範的ガイダンスが提供する戦略、ガイド、パターンで一般的に使用される用語です。エントリを提案するには、用語集の最後のフィードバックの提供リンクを使用します。

数字

7 Rs

アプリケーションをクラウドに移行するための 7 つの一般的な移行戦略。これらの戦略は、ガートナーが 2011 年に特定した 5 Rs に基づいて構築され、以下で構成されています。

- リファクタリング/アーキテクチャの再設計 — クラウドネイティブ特徴を最大限に活用して、俊敏性、パフォーマンス、スケーラビリティを向上させ、アプリケーションを移動させ、アーキテクチャを変更します。これには、通常、オペレーティングシステムとデータベースの移植が含まれます。例: オンプレミスの Oracle データベースを Amazon Aurora Postgre SQL互換エディションに移行します。
- リプラットフォーム (リフトアンドリシェイプ) – アプリケーションをクラウドに移行し、クラウド機能を活用するためにある程度の最適化を導入します。例: でオンプレミスの Oracle データベースを Oracle 用 Amazon Relational Database Service (Amazon RDS) に移行します AWS クラウド。
- 再購入 (ドロップアンドショップ) — 通常、従来のライセンスから SaaS モデルに移行して、別の製品に切り替えます。例: 顧客関係管理 (CRM) システムを Salesforce.com に移行します。
- リホスト (リフトアンドシフト) — クラウド機能を活用するための変更を加えずに、アプリケーションをクラウドに移行します。例: オンプレミスの Oracle データベースを のEC2インスタンス上の Oracle に移行します AWS クラウド。
- 再配置 (ハイパーバイザーレベルのリフトアンドシフト) – 新しいハードウェアを購入したり、アプリケーションを書き換えたり、既存の運用を変更したりすることなく、インフラストラクチャをクラウドに移行できます。オンプレミスプラットフォームから同じプラットフォームのクラウドサービスにサーバーを移行します。例: の移行 Microsoft Hyper-V アプリケーション AWS。
- 保持 (再アクセス) — アプリケーションをお客様のソース環境で保持します。これには、主要なリファクタリングを必要とするアプリケーションや、お客様がその作業を後日まで延期したいアプリケーション、およびそれらを移行するためのビジネス上の正当性がないため、お客様が保持するレガシーアプリケーションなどがあります。
- 使用停止 — お客様のソース環境で不要になったアプリケーションを停止または削除します。

A

ABAC

「属性ベースのアクセスコントロール」を参照してください。

抽象化されたサービス

「 マネージドサービス」を参照してください。

ACID

不可分性、一貫性、分離性、耐久性を参照してください。

アクティブ - アクティブ移行

(双方向レプリケーションツールまたは二重書き込み操作を使用して) ソースデータベースとターゲットデータベースを同期させ、移行中に両方のデータベースが接続アプリケーションからのトランザクションを処理するデータベース移行方法。この方法では、1 回限りのカットオーバーの必要がなく、管理された小規模なバッチで移行できます。アクティブ/パッシブ移行よりも柔軟性はありますが、より多くの作業が必要です。

アクティブ - パッシブ移行

ソースデータベースとターゲットデータベースを同期させながら、データがターゲットデータベースにレプリケートされている間、接続しているアプリケーションからのトランザクションをソースデータベースのみで処理するデータベース移行の方法。移行中、ターゲットデータベースはトランザクションを受け付けません。

集計関数

行のグループを操作し、グループの単一の戻り値を計算するSQL関数。集計関数の例には、SUMや などがありますMAX。

AI

「人工知能」を参照してください。

AIOps

「人工知能オペレーション」を参照してください。

匿名化

データセット内の個人情報を完全に削除するプロセス。匿名化は個人のプライバシー保護に役立ちます。匿名化されたデータは、もはや個人データとは見なされません。

アンチパターン

繰り返し起こる問題に対して頻繁に用いられる解決策で、その解決策が逆効果であったり、効果がなかったり、代替案よりも効果が低かったりするもの。

アプリケーションコントロール

マルウェアからシステムを保護するために承認されたアプリケーションのみを使用できるようにするセキュリティアプローチ。

アプリケーションポートフォリオ

アプリケーションの構築と維持にかかるコスト、およびそのビジネス価値を含む、組織が使用する各アプリケーションに関する詳細情報の集まり。この情報は、ポートフォリオの検出と分析プロセス の需要要素であり、移行、モダナイズ、最適化するアプリケーションを特定し、優先順位を付けるのに役立ちます。

人工知能 (AI)

コンピューティングテクノロジーを使用し、学習、問題の解決、パターンの認識など、通常は人間に関連づけられる認知機能の実行に特化したコンピュータサイエンスの分野。詳細については、「人工知能 (AI) とは何ですか?」を参照してください。

人工知能オペレーション (AIOps)

機械学習技術を使用して運用上の問題を解決し、運用上のインシデントと人の介入を減らし、サービス品質を向上させるプロセス。移行戦略での AWS AIOpsの使用方法の詳細については、「 オペレーション統合ガイド」を参照してください。

非対称暗号化

暗号化用のパブリックキーと復号用のプライベートキーから成る 1 組のキーを使用した、暗号化のアルゴリズム。パブリックキーは復号には使用されないため共有しても問題ありませんが、プライベートキーの利用は厳しく制限する必要があります。

アトミック性、一貫性、分離性、耐久性 (ACID)

エラー、停電、その他の問題が発生した場合でも、データベースのデータ有効性と運用上の信頼性を保証する一連のソフトウェアプロパティ。

属性ベースのアクセスコントロール (ABAC)

部署、役職、チーム名など、ユーザーの属性に基づいてアクセス許可をきめ細かく設定する方法。詳細については、 AWS Identity and Access Management （ IAM) ドキュメントの「 for ABAC AWS」を参照してください。

信頼できるデータソース

最も信頼性のある情報源とされるデータのプライマリーバージョンを保存する場所。匿名化、編集、仮名化など、データを処理または変更する目的で、信頼できるデータソースから他の場所にデータをコピーすることができます。

アベイラビリティーゾーン

他のアベイラビリティーゾーンの障害から AWS リージョン 隔離され、同じリージョン内の他のアベイラビリティーゾーンへの低コストで低レイテンシーのネットワーク接続を提供する 内の別の場所。

AWS クラウド導入フレームワーク (AWS CAF)

組織がクラウドに正常に移行するための効率的で効果的な計画を立て AWS るのに役立つ、 からのガイドラインとベストプラクティスのフレームワークです。 AWS CAF は、ビジネス、人材、ガバナンス、プラットフォーム、セキュリティ、運用という 6 つの重点分野にガイダンスを編成しています。ビジネス、人材、ガバナンスの観点では、ビジネススキルとプロセスに重点を置き、プラットフォーム、セキュリティ、オペレーションの視点は技術的なスキルとプロセスに焦点を当てています。例えば、人材の観点では、人事 (HR)、人材派遣機能、および人材管理を扱うステークホルダーを対象としています。この観点から、 AWS CAF は、クラウド導入を成功させるための組織の準備に役立つ人材開発、トレーニング、コミュニケーションに関するガイダンスを提供します。詳細については、 AWS CAFウェブサイトとAWS CAFホワイトペーパーを参照してください。

AWS ワークロード認定フレームワーク (AWS WQF)

データベース移行ワークロードを評価し、移行戦略を推奨し、作業の見積もりを提供するツール。 AWS WQF は AWS Schema Conversion Tool （ AWS SCT) に含まれています。データベーススキーマとコードオブジェクト、アプリケーションコード、依存関係、およびパフォーマンス特性を分析し、評価レポートを提供します。

B

不正なボット

個人や組織に混乱を与えたり、損害を与えたりすることを意図したボット。

BCP

事業継続計画を参照してください。

動作グラフ

リソースの動作とインタラクションを経時的に示した、一元的なインタラクティブビュー。Amazon Detective で動作グラフを使用して、失敗したログオン試行、不審なAPI呼び出し、および同様のアクションを調べることができます。詳細については、Detective ドキュメントのData in a behavior graphを参照してください。

ビッグエンディアンシステム

最上位バイトを最初に格納するシステム。エンディアンネスも参照してください。

二項分類

バイナリ結果 (2 つの可能なクラスのうちの 1 つ) を予測するプロセス。例えば、お客様の機械学習モデルで「この E メールはスパムですか、それともスパムではありませんか」などの問題を予測する必要があるかもしれません。または「この製品は書籍ですか、車ですか」などの問題を予測する必要があるかもしれません。

ブルームフィルター

要素がセットのメンバーであるかどうかをテストするために使用される、確率的でメモリ効率の高いデータ構造。

ブルー/グリーンデプロイ

2 つの異なる同一の環境を作成するデプロイ戦略。現在のアプリケーションバージョンは 1 つの環境 (青) で実行し、新しいアプリケーションバージョンは別の環境 (緑) で実行します。この戦略は、最小限の影響で迅速にロールバックするのに役立ちます。

ボット

インターネット経由で自動タスクを実行し、人間のアクティビティややり取りをシミュレートするソフトウェアアプリケーション。インターネット上の情報のインデックスを作成するウェブクローラーなど、一部のボットは有用または有益です。悪質なボットと呼ばれる他のボットの中には、個人や組織に混乱を与えたり、損害を与えたりすることを意図しているものがあります。

ボットネット

マルウェアに感染し、ボットのヘルダーまたはボットオペレーターとして知られる、単一の当事者によって管理されているボットのネットワーク。ボットは、ボットとその影響をスケールするための最もよく知られているメカニズムです。

ブランチ

コードリポジトリに含まれる領域。リポジトリに最初に作成するブランチは、メインブランチといいます。既存のブランチから新しいブランチを作成し、その新しいブランチで機能を開発した

り、バグを修正したりできます。機能を構築するために作成するブランチは、通常、機能ブランチと呼ばれます。機能をリリースする準備ができたら、機能ブランチをメインブランチに統合します。詳細については、「About branches (GitHub documentation)」を参照してください。

ブレークグラスアクセス

例外的な状況では、承認されたプロセスを通じて、ユーザーが AWS アカウント 通常アクセス許可を持たない にすばやくアクセスできるようにします。詳細については、 Well-Architected ガイダンスの「ブレークグラスプロシージャの実装」インジケータ AWS を参照してください。

ブラウンフィールド戦略

環境の既存インフラストラクチャ。システムアーキテクチャにブラウンフィールド戦略を導入する場合、現在のシステムとインフラストラクチャの制約に基づいてアーキテクチャを設計します。既存のインフラストラクチャを拡張している場合は、ブラウンフィールド戦略とグリーンフィールド戦略を融合させることもできます。

バッファキャッシュ

アクセス頻度が最も高いデータが保存されるメモリ領域。

ビジネス能力

価値を生み出すためにビジネスが行うこと (営業、カスタマーサービス、マーケティングなど)。マイクロサービスのアーキテクチャと開発の決定は、ビジネス能力によって推進できます。詳細については、ホワイトペーパー AWSでのコンテナ化されたマイクロサービスの実行 の ビジネス機能を中心に組織化 セクションを参照してください。

事業継続計画 (BCP)

大規模移行など、中断を伴うイベントが運用に与える潜在的な影響に対処し、ビジネスを迅速に再開できるようにする計画。

C

CAF

AWS 「クラウド導入フレームワーク」を参照してください。

Canary デプロイ

エンドユーザーへのバージョンのスローリリースと増分リリース。確信できたら、新しいバージョンをデプロイし、現在のバージョン全体を置き換えます。

CCoE

「Cloud Center of Excellence」を参照してください。

CDC

「変更データキャプチャ」を参照してください。

変更データキャプチャ (CDC)

データソース (データベーステーブルなど) の変更を追跡し、その変更に関するメタデータを記録するプロセス。CDC は、同期を維持するために、ターゲットシステムの変更の監査やレプリケーションなど、さまざまな目的で使用できます。

カオスエンジニアリング

障害や破壊的なイベントを意図的に導入して、システムの耐障害性をテストする。AWS Fault Injection Service （ AWS FIS ） を使用して、 AWS ワークロードにストレスを与え、その応答を評価する実験を実行できます。

CI/CD

「継続的インテグレーションと継続的デリバリー」を参照してください。

分類

予測を生成するのに役立つ分類プロセス。分類問題の機械学習モデルは、離散値を予測します。離散値は、常に互いに区別されます。例えば、モデルがイメージ内に車があるかどうかを評価する必要がある場合があります。

クライアント側の暗号化

ターゲットがデータ AWS のサービス を受信する前に、ローカルでデータを暗号化します。

Cloud Center of Excellence (CCoE)

クラウドのベストプラクティスの作成、リソースの移動、移行のタイムラインの確立、大規模変革を通じて組織をリードするなど、組織全体のクラウド導入の取り組みを推進する学際的なチーム。詳細については、 AWS クラウド エンタープライズ戦略ブログCCoEの投稿を参照してください。

クラウドコンピューティング

リモートデータストレージと IoT デバイス管理に通常使用されるクラウドテクノロジー。クラウドコンピューティングは、一般的にエッジコンピューティングテクノロジーに接続されています。

クラウド運用モデル

IT 組織において、1 つ以上のクラウド環境を構築、成熟、最適化するために使用される運用モデル。詳細については、「クラウド運用モデルの構築」を参照してください。

導入のクラウドステージ

組織は通常、 に移行する際に次の 4 つのフェーズを実行します AWS クラウド。

- プロジェクト — 概念実証と学習を目的として、クラウド関連のプロジェクトをいくつか実行する
- 基盤 — クラウド導入を拡大するための基本的な投資 (ランディングゾーンの作成、 の定義 CCoE、運用モデルの確立など)
- 移行 — 個々のアプリケーションの移行
- 再発明 — 製品とサービスの最適化、クラウドでのイノベーション

これらのステージは、 AWS クラウド エンタープライズ戦略ブログのブログ記事「クラウドファーストへのジャーニー」と「導入のステージ」で Stephen Orban によって定義されています。移行戦略とどのように関連しているかについては、 AWS「移行準備ガイド」を参照してください。

CMDB

「設定管理データベース」を参照してください。

コードリポジトリ

ソースコードやその他の資産 (ドキュメント、サンプル、スクリプトなど) が保存され、バージョン管理プロセスを通じて更新される場所。一般的なクラウドリポジトリには以下が含まれます。GitHub or Bitbucket Cloud。 コードの各バージョンはブランチと呼ばれます。マイクロサービスの構造では、各リポジトリは 1 つの機能専用です。1 つの CI/CD パイプラインで複数のリポジトリを使用できます。

コールドキャッシュ

空である、または、かなり空きがある、もしくは、古いデータや無関係なデータが含まれているバッファキャッシュ。データベースインスタンスはメインメモリまたはディスクから読み取る必要があり、バッファキャッシュから読み取るよりも時間がかかるため、パフォーマンスに影響します。

コールドデータ

めったにアクセスされず、通常は過去のデータです。この種類のデータをクエリする場合、通常は低速なクエリでも問題ありません。このデータを低パフォーマンスで安価なストレージ階層またはクラスに移動すると、コストを削減することができます。

コンピュータビジョン (CV)

機械学習を使用してデジタルイメージやビデオなどのビジュアル形式から情報を分析および抽出する AI の分野。例えば、 はオンプレミスのカメラネットワークに CV を追加するデバイス AWS Panorama を提供し、Amazon SageMaker AI は CV のイメージ処理アルゴリズムを提供します。

設定ドリフト

ワークロードの場合、設定は想定した状態から変化します。これにより、ワークロードが非準拠になる可能性があり、通常は段階的で意図的ではありません。

設定管理データベース (CMDB)

データベースとその IT 環境 (ハードウェアとソフトウェアの両方のコンポーネントとその設定を含む) に関する情報を保存、管理するリポジトリ。通常、移行のポートフォリオ検出および分析段階で CMDB のデータを使用します。

コンフォーマンスパック

コンプライアンスチェックとセキュリティチェックをカスタマイズするためにアセンブルできる AWS Config ルールと修復アクションのコレクション。YAML テンプレートを使用して、コンフォーマンスパックを AWS アカウント とリージョン、または組織全体の単一のエンティティとしてデプロイできます。詳細については、 AWS Config ドキュメントの「コンフォーマンスパック」を参照してください。

継続的インテグレーションと継続的デリバリー (CI/CD)

ソフトウェアリリースプロセスのソース、ビルド、テスト、ステージング、本番稼働の各段階を自動化するプロセス。 CI/CD is commonly described as a pipeline. CI/CDは、プロセスの自動化、生産性の向上、コード品質の向上、迅速な提供に役立ちます。詳細については、「継続的デリバリーの利点」を参照してください。CD は継続的デプロイ (Continuous Deployment) の略語でもあります。詳細については「継続的デリバリーと継続的なデプロイ」を参照してください。

CV

「コンピュータビジョン」を参照してください。

D

保管中のデータ

ストレージ内にあるデータなど、常に自社のネットワーク内にあるデータ。

データ分類

ネットワーク内のデータを重要度と機密性に基づいて識別、分類するプロセス。データに適した保護および保持のコントロールを判断する際に役立つため、あらゆるサイバーセキュリティのリスク管理戦略において重要な要素です。データ分類は、 AWS Well-Architected フレームワークのセキュリティの柱のコンポーネントです。詳細については、データ分類を参照してください。

データドリフト

実稼働データと ML モデルのトレーニングに使用されたデータとの間に有意な差異が生じたり、入力データが時間の経過と共に有意に変化したりすることです。データドリフトは、ML モデル予測の全体的な品質、精度、公平性を低下させる可能性があります。

転送中のデータ

ネットワーク内 (ネットワークリソース間など) を活発に移動するデータ。

データメッシュ

一元化された管理とガバナンスにより、分散され分散されたデータ所有権を提供するアーキテクチャフレームワーク。

データ最小化

厳密に必要なデータのみを収集し、処理するという原則。でデータ最小化を実践 AWS クラウドすることで、プライバシーリスク、コスト、分析のカーボンフットプリントを削減できます。

データ境界

AWS 環境内の一連の予防ガードレール。信頼できる ID のみが、期待されるネットワークから信頼できるリソースにアクセスしていることを確認できます。詳細については、「 でのデータ境界の構築 AWS」を参照してください。

データの前処理

raw データをお客様の機械学習モデルで簡単に解析できる形式に変換すること。データの前処理とは、特定の列または行を削除して、欠落している、矛盾している、または重複する値に対処することを意味します。

データ出所

データの生成、送信、保存の方法など、データのライフサイクル全体を通じてデータの出所と履歴を追跡するプロセス。

データ件名

データを収集、処理している個人。

データウェアハウス

分析などのビジネスインテリジェンスをサポートするデータ管理システム。データウェアハウスには、通常、大量の履歴データが含まれており、クエリや分析に使用されます。

データベース定義言語 (DDL)

データベース内のテーブルやオブジェクトの構造を作成または変更するためのステートメントまたはコマンド。

データベース操作言語 (DML)

データベース内の情報を変更 (挿入、更新、削除) するためのステートメントまたはコマンド。

DDL

「データベース定義言語」を参照してください。

ディープアンサンブル

予測のために複数の深層学習モデルを組み合わせる。ディープアンサンブルを使用して、より正確な予測を取得したり、予測の不確実性を推定したりできます。

ディープラーニング

人工ニューラルネットワークの複数層を使用して、入力データと対象のターゲット変数の間のマッピングを識別する機械学習サブフィールド。

defense-in-depth

一連のセキュリティメカニズムとコントロールをコンピュータネットワーク全体に層状に重ねて、ネットワークとその内部にあるデータの機密性、整合性、可用性を保護する情報セキュリティの手法。この戦略を採用するときは AWS、 AWS Organizations 構造の異なるレイヤーに複数のコントロールを追加して、リソースの安全性を確保します。例えば、アプローチでは defense-in-depth、多要素認証、ネットワークセグメンテーション、暗号化を組み合わせることができます。

委任管理者

では AWS Organizations、互換性のあるサービスが AWS メンバーアカウントを登録して組織のアカウントを管理し、そのサービスのアクセス許可を管理できます。このアカウントを、そのサービスの委任管理者と呼びます。詳細、および互換性のあるサービスの一覧は、 AWS Organizations ドキュメントのAWS Organizationsで使用できるサービスを参照してください。

デプロイ

アプリケーション、新機能、コードの修正をターゲットの環境で利用できるようにするプロセス。デプロイでは、コードベースに変更を施した後、アプリケーションの環境でそのコードベースを構築して実行します。

開発環境

「 環境」を参照してください。

検出管理

イベントが発生したときに、検出、ログ記録、警告を行うように設計されたセキュリティコントロール。これらのコントロールは副次的な防衛手段であり、実行中の予防的コントロールをすり抜けたセキュリティイベントをユーザーに警告します。詳細については、Implementing security controls on AWSのDetective controlsを参照してください。

開発値ストリームマッピング (DVSM)

ソフトウェア開発ライフサイクルの速度と品質に悪影響を及ぼす制約を特定して優先順位を付けるために使用されるプロセス。 は、もともと効率的な製造プラクティスのために設計されたバリューストリームマッピングプロセスDVSMを拡張します。ソフトウェア開発プロセスを通じて価値を創造し、動かすために必要なステップとチームに焦点を当てています。

デジタルツイン

建物、工場、産業機器、生産ラインなど、現実世界のシステムを仮想的に表現したものです。デジタルツインは、予知保全、リモートモニタリング、生産最適化をサポートします。

ディメンションテーブル

スタースキーマでは、ファクトテーブル内の量的データに関するデータ属性を含む小さなテーブル。ディメンションテーブル属性は通常、テキストフィールドまたはテキストのように動作する離散数値です。これらの属性は、クエリの制約、フィルタリング、結果セットのラベル付けによく使用されます。

ディザスタ

ワークロードまたはシステムが、導入されている主要な場所でのビジネス目標の達成を妨げるイベント。これらのイベントは、自然災害、技術的障害、または意図しない設定ミスやマルウェア攻撃などの人間の行動の結果である場合があります。

ディザスタリカバリ (DR)

災害によるダウンタイムとデータ損失を最小限に抑えるための戦略とプロセス。詳細については、 AWS Well-Architected フレームワークの「 でのワークロードのディザスタリカバリ AWS: クラウドでのリカバリ」を参照してください。

DML

「データベース操作言語」を参照してください。

ドメイン駆動型設計

各コンポーネントが提供している変化を続けるドメイン、またはコアビジネス目標にコンポーネントを接続して、複雑なソフトウェアシステムを開発するアプローチ。この概念は、エリック・エヴァンスの著書、Domain-Driven Design: Tackling Complexity in the Heart of Software (ドメイン駆動設計:ソフトウェアの中心における複雑さへの取り組み) で紹介されています (ボストン: Addison-Wesley Professional、2003)。strangler fig パターンでドメイン駆動型設計を使用する方法については、「従来の Microsoft ASP.NET (ASMX) ウェブサービスをコンテナと Amazon API Gateway を使用して段階的にモダナイズする」を参照してください。

DR

「ディザスタリカバリ」を参照してください。

ドリフト検出

ベースライン設定からの偏差の追跡。例えば、 AWS CloudFormation を使用してシステムリソースのドリフトを検出したり、 を使用して AWS Control Tower ガバナンス要件への準拠に影響を与える可能性のあるランディングゾーンの変更を検出したりできます。

DVSM

「開発値ストリームマッピング」を参照してください。

E

EDA

「探索的なデータ分析」を参照してください。

EDI

「電子データ交換」を参照してください。

エッジコンピューティング

IoT ネットワークのエッジにあるスマートデバイスの計算能力を高めるテクノロジー。クラウドコンピューティングと比較すると、エッジコンピューティングは通信レイテンシーを短縮し、応答時間を短縮できます。

電子データ交換 (EDI)

組織間のビジネスドキュメントの自動交換。詳細については、「電子データ交換とは」を参照してください。

暗号化

人間が読み取り可能なプレーンテキストデータを暗号文に変換するコンピューティングプロセス。

暗号化キー

暗号化アルゴリズムが生成した、ランダム化されたビットからなる暗号文字列。キーの長さは決まっておらず、各キーは予測できないように、一意になるように設計されています。

エンディアン

コンピュータメモリにバイトが格納される順序。ビッグエンディアンシステムでは、最上位バイトが最初に格納されます。リトルエンディアンシステムでは、最下位バイトが最初に格納されます。

エンドポイント

「 サービスエンドポイント」を参照してください。

エンドポイントサービス

仮想プライベートクラウド (VPC) でホストして他のユーザーと共有できるサービス。を使用してエンドポイントサービスを作成し AWS PrivateLink 、他の AWS アカウント または AWS Identity and Access Management （ IAM) プリンシパルにアクセス許可を付与できます。これらのアカウントまたはプリンシパルは、インターフェイスエンドポイントを作成することでVPC、エンドポイントサービスにプライベートに接続できます。詳細については、Amazon Virtual Private Cloud (Amazon VPC) ドキュメントの「エンドポイントサービスの作成」を参照してください。

エンタープライズリソース計画 (ERP)

エンタープライズの主要なビジネスプロセス (会計、MES、プロジェクト管理など) を自動化および管理するシステム。

エンベロープ暗号化

暗号化キーを、別の暗号化キーを使用して暗号化するプロセス。詳細については、 AWS Key Management Service （ AWS KMS) ドキュメントの「エンベロープ暗号化」を参照してください。

環境

実行中のアプリケーションのインスタンス。クラウドコンピューティングにおける一般的な環境の種類は以下のとおりです。

- 開発環境 — アプリケーションのメンテナンスを担当するコアチームのみが使用できる、実行中のアプリケーションのインスタンス。開発環境は、上位の環境に昇格させる変更をテストするときに使用します。このタイプの環境は、テスト環境と呼ばれることもあります。
- 下位環境 — 初期ビルドやテストに使用される環境など、アプリケーションのすべての開発環境。
- 本番環境 — エンドユーザーがアクセスできる、実行中のアプリケーションのインスタンス。CI/CD パイプラインでは、本番環境が最後のデプロイ環境になります。
- 上位環境 — コア開発チーム以外のユーザーがアクセスできるすべての環境。これには、本番環境、本番前環境、ユーザー承認テスト環境などが含まれます。

エピック

アジャイル方法論で、お客様の作業の整理と優先順位付けに役立つ機能カテゴリ。エピックでは、要件と実装タスクの概要についてハイレベルな説明を提供します。例えば、 AWS CAFセキュリティエピックには、アイデンティティとアクセスの管理、検出コントロール、インフラストラクチャセキュリティ、データ保護、インシデント対応が含まれます。 AWS 移行戦略のエピックの詳細については、プログラム実装ガイド を参照してください。

ERP

「エンタープライズリソース計画」を参照してください。

探索的なデータ分析 (EDA)

データセットを分析してその主な特性を理解するプロセス。データを収集または集計し、初期調査を実行してパターンを検索し、異常を検出して、仮定を確認します。 EDAは、サマリー統計を計算し、データの視覚化を作成することによって実行されます。

F

ファクトテーブル

星スキーマの中央テーブル。事業運営に関する量的データを保存します。通常、ファクトテーブルには、メジャーを含む列とディメンションテーブルへの外部キーを含む列の 2 つのタイプの列が含まれます。

フェイルファスト

開発ライフサイクルを短縮するために頻繁で段階的なテストを使用する哲学。これはアジャイルアプローチの重要な部分です。

障害分離の境界

では AWS クラウド、障害の影響を制限し、ワークロードの耐障害性を向上させるアベイラビリティーゾーン AWS リージョン、コントロールプレーン、データプレーンなどの境界です。詳細については、AWS 「障害分離境界」を参照してください。

機能ブランチ

「ブランチ」を参照してください。

特徴量

お客様が予測に使用する入力データ。例えば、製造コンテキストでは、特徴量は製造ラインから定期的にキャプチャされるイメージの可能性もあります。

特徴量重要度

モデルの予測に対する特徴量の重要性。これは通常、Shapley Additive Descriptions (SHAP) や積分勾配など、さまざまな手法で計算できる数値スコアとして表されます。詳細については、「 を使用した機械学習モデルの解釈可能性 AWS」を参照してください。

機能変換

追加のソースによるデータのエンリッチ化、値のスケーリング、単一のデータフィールドからの複数の情報セットの抽出など、機械学習プロセスのデータを最適化すること。これにより、機械学習モデルはデータの恩恵を受けることができます。例えば、「2021-05-27 00:15:37」の日付を「2021 年」、「5 月」、「木」、「15」に分解すると、学習アルゴリズムがさまざまなデータコンポーネントに関連する微妙に異なるパターンを学習するのに役立ちます。

数ショットプロンプト

同様のタスクを実行するように依頼する前に、タスクと必要な出力を示すLLM少数の例を に提供します。この手法はコンテキスト内学習のアプリケーションであり、モデルはプロンプトに埋め

込まれた例 (ショット) から学習します。特定のフォーマット、推論、またはドメインの知識を必要とするタスクには、数ショットプロンプトが効果的です。「ゼロショットプロンプト」も参照してください。

FGAC

「きめ細かなアクセスコントロール」を参照してください。

きめ細かなアクセスコントロール (FGAC)

複数の条件を使用してアクセス要求を許可または拒否すること。

フラッシュカット移行

段階的なアプローチを使用するのではなく、変更データキャプチャによる継続的なデータレプリケーションを使用して、可能な限り短時間でデータを移行するデータベース移行方法。目的はダウンタイムを最小限に抑えることです。

FM

「基盤モデル」を参照してください。

基盤モデル (FM)

一般化されたデータやラベル付けされていないデータの大規模なデータセットでトレーニングされている大規模な深層学習ニューラルネットワーク。 FMsは、言語の理解、テキストや画像の生成、自然言語での会話など、さまざまな一般的なタスクを実行できます。詳細については、「基礎モデルとは」を参照してください。

G

生成 AI

大量のデータでトレーニングされ、シンプルなテキストプロンプトを使用してイメージ、動画、テキスト、オーディオなどの新しいコンテンツやアーティファクトを作成できる AI モデルのサブセット。詳細については、「生成 AI とは」を参照してください。

ジオブロッキング

地理的制限を参照してください。

地理的制限 (ジオブロッキング)

Amazon では CloudFront、特定の国のユーザーがコンテンツディストリビューションにアクセスできないようにするオプションです。アクセスを許可する国と禁止する国は、許可リストまたは

禁止リストを使って指定します。詳細については、 CloudFront ドキュメントの「コンテンツの地理的ディストリビューションの制限」を参照してください。

Gitflow ワークフロー

下位環境と上位環境が、ソースコードリポジトリでそれぞれ異なるブランチを使用する方法。Gitflow ワークフローはレガシーと見なされ、トランクベースのワークフローは最新の推奨アプローチです。

ゴールデンイメージ

システムまたはソフトウェアの新しいインスタンスをデプロイするためのテンプレートとして使用されるシステムまたはソフトウェアのスナップショット。例えば、製造では、ゴールデンイメージを使用して複数のデバイスにソフトウェアをプロビジョニングし、デバイスの製造オペレーションの速度、スケーラビリティ、生産性を向上させることができます。

グリーンフィールド戦略

新しい環境に既存のインフラストラクチャが存在しないこと。システムアーキテクチャにグリーンフィールド戦略を導入する場合、既存のインフラストラクチャ (別名ブラウンフィールド) との互換性の制約を受けることなく、あらゆる新しいテクノロジーを選択できます。既存のインフラストラクチャを拡張している場合は、ブラウンフィールド戦略とグリーンフィールド戦略を融合させることもできます。

ガードレール

組織単位 () 全体のリソース、ポリシー、コンプライアンスを管理するのに役立つ高レベルのルールOUs。予防ガードレールは、コンプライアンス基準に一致するようにポリシーを実施します。これらは、サービスコントロールポリシーとIAMアクセス許可の境界を使用して実装されます。検出ガードレールは、ポリシー違反やコンプライアンス上の問題を検出し、修復のためのアラートを発信します。これらは AWS Config、、Amazon AWS Security Hub、 GuardDuty、 AWS Trusted Advisor Amazon Inspector、およびカスタム AWS Lambda チェックを使用して実装されます。

H

HA

「高可用性」を参照してください。

異種混在データベースの移行

別のデータベースエンジンを使用するターゲットデータベースへお客様の出典データベースの移行 (例えば、Oracle から Amazon Aurora)。異種間移行は通常、アーキテクチャの再設計作業の一部であり、スキーマの変換は複雑なタスクになる可能性があります。AWS は、スキーマの変換に役立つ AWS SCTを提供します。

ハイアベイラビリティ (HA)

課題や災害が発生した場合に、介入なしにワークロードを継続的に運用できること。HA システムは、自動的にフェイルオーバーし、一貫して高品質のパフォーマンスを提供し、パフォーマンスへの影響を最小限に抑えながらさまざまな負荷や障害を処理するように設計されています。

ヒストリアンのモダナイゼーション

製造業のニーズによりよく応えるために、オペレーションテクノロジー (OT) システムをモダナイズし、アップグレードするためのアプローチ。ヒストリアンは、工場内のさまざまなソースからデータを収集して保存するために使用されるデータベースの一種です。

ホールドアウトデータ

機械学習モデルのトレーニングに使用されるデータセットから保留される、ラベル付きの履歴データの一部。ホールドアウトデータを使用してモデル予測をホールドアウトデータと比較することで、モデルのパフォーマンスを評価できます。

同種データベースの移行

ソースデータベースを、同じデータベースエンジンを共有するターゲットデータベース (Microsoft SQL Server から Amazon RDS for SQL Server など) に移行します。同種間移行は、通常、リホストまたはリプラットフォーム化の作業の一部です。ネイティブデータベースユーティリティを使用して、スキーマを移行できます。

ホットデータ

リアルタイムデータや最近の翻訳データなど、頻繁にアクセスされるデータ。通常、このデータには高速なクエリ応答を提供する高性能なストレージ階層またはクラスが必要です。

ホットフィックス

本番環境の重大な問題を修正するために緊急で配布されるプログラム。緊急性のため、通常、修正は一般的な DevOps リリースワークフローの外部で行われます。

ハイパーケア期間

カットオーバー直後、移行したアプリケーションを移行チームがクラウドで管理、監視して問題に対処する期間。通常、この期間は 1～4 日です。ハイパーケア期間が終了すると、アプリケーションに対する責任は一般的に移行チームからクラウドオペレーションチームに移ります。

I

IaC

Infrastructure as Code を参照してください。

ID ベースのポリシー

AWS クラウド 環境内のアクセス許可を定義する 1 つ以上のIAMプリンシパルにアタッチされたポリシー。

アイドル状態のアプリケーション

90 日間の平均CPUおよびメモリ使用量が 5～20% のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するか、オンプレミスに保持するのが一般的です。

IIoT

「産業モノのインターネット」を参照してください。

イミュータブルインフラストラクチャ

既存のインフラストラクチャを更新、パッチ適用、または変更するのではなく、本番ワークロードに新しいインフラストラクチャをデプロイするモデル。イミュータブルなインフラストラクチャは、本質的にミュータブルなインフラストラクチャよりも一貫性、信頼性、予測性が高くなります。詳細については、 AWS Well-Architected フレームワークの「イミュータブルインフラストラクチャを使用したデプロイ」のベストプラクティスを参照してください。

インバウンド (イングレス) VPC

AWS マルチアカウントアーキテクチャでは、 がアプリケーション外からのネットワーク接続を受け入れ、検査し、ルーティングVPCする です。AWS セキュリティリファレンスアーキテクチャでは、アプリケーションとより広範なインターネット間の双方向インターフェイスVPCsを保護するために、インバウンド、アウトバウンド、検査を使用してネットワークアカウントを設定することをお勧めします。

増分移行

アプリケーションを 1 回ですべてカットオーバーするのではなく、小さい要素に分けて移行するカットオーバー戦略。例えば、最初は少数のマイクロサービスまたはユーザーのみを新しいシステムに移行する場合があります。すべてが正常に機能することを確認できたら、残りのマイクロサービスやユーザーを段階的に移行し、レガシーシステムを廃止できるようにします。この戦略により、大規模な移行に伴うリスクが軽減されます。

インダストリー 4.0

2016 年に Klaus Schwab によって導入された用語で、接続、リアルタイムデータ、自動化、分析、AI/ML の進歩によるビジネスプロセスのモダナイゼーションを指します。

インフラストラクチャ

アプリケーションの環境に含まれるすべてのリソースとアセット。

Infrastructure as Code (IaC)

アプリケーションのインフラストラクチャを一連の設定ファイルを使用してプロビジョニングし、管理するプロセス。IaC は、新しい環境を再現可能で信頼性が高く、一貫性のあるものにするため、インフラストラクチャを一元的に管理し、リソースを標準化し、スケールを迅速に行えるように設計されています。

産業 IoT (IIoT)

製造、エネルギー、自動車、ヘルスケア、ライフサイエンス、農業などの産業部門におけるインターネットに接続されたセンサーやデバイスの使用。詳細については、「産業モノのインターネット (IIoT) デジタルトランスフォーメーション戦略の構築」を参照してください。

検査 VPC

AWS マルチアカウントアーキテクチャでは、 VPCs (同じまたは異なる 内 AWS リージョン)、インターネット、オンプレミスネットワーク間のネットワークトラフィックの検査VPCを管理する一元化された です。AWS セキュリティリファレンスアーキテクチャでは、アプリケーションとより広範なインターネット間の双方向インターフェイスVPCsを保護するために、インバウンド、アウトバウンド、検査を使用してネットワークアカウントを設定することをお勧めします。

IoT

インターネットまたはローカル通信ネットワークを介して他のデバイスやシステムと通信する、センサーまたはプロセッサが組み込まれた接続済み物理オブジェクトのネットワーク。詳細については、「IoT とは」を参照してください。

解釈可能性

機械学習モデルの特性で、モデルの予測がその入力にどのように依存するかを人間が理解できる度合いを表します。詳細については、「 を使用した機械学習モデルの解釈可能性AWS」を参照してください。

IoT

「モノのインターネット」を参照してください。

IT 情報ライブラリ (ITIL)

IT サービスを提供し、これらのサービスをビジネス要件に合わせるための一連のベストプラクティス。 ITILは、 の基盤を提供しますITSM。

IT サービス管理 (ITSM)

組織の IT サービスの設計、実装、管理、およびサポートに関連する活動。クラウドオペレーションと ITSM ツールの統合については、「 オペレーション統合ガイド」を参照してください。

ITIL

「IT 情報ライブラリ」を参照してください。

ITSM

「IT サービス管理」を参照してください。

L

ラベルベースのアクセスコントロール (LBAC)

ユーザーとデータ自体にそれぞれセキュリティラベル値が明示的に割り当てられている、必須のアクセスコントロール (MAC) の実装。ユーザーセキュリティラベルとデータセキュリティラベルが交差する部分によって、ユーザーに表示される行と列が決まります。

ランディングゾーン

ランディングゾーンは、スケーラブルで安全な、適切に設計されたマルチアカウント AWS 環境です。これは、組織がセキュリティおよびインフラストラクチャ環境に自信を持ってワークロードとアプリケーションを迅速に起動してデプロイできる出発点です。ランディングゾーンの詳細については、安全でスケーラブルなマルチアカウント AWS 環境のセットアップ を参照してください。

大規模言語モデル (LLM)

大量のデータに基づいて事前トレーニングされた深層学習 AI モデル。LLM は、質問への回答、ドキュメントの要約、テキストの他の言語への翻訳、文の完了など、複数のタスクを実行できます。詳細については、「 とはLLMs」を参照してください。

大規模な移行

300 台以上のサーバの移行。

LBAC

「ラベルベースのアクセスコントロール」を参照してください。

最小特権

タスクの実行には必要最低限の権限を付与するという、セキュリティのベストプラクティス。詳細については、 IAMドキュメントの「最小特権のアクセス許可を適用する」を参照してください。

リフトアンドシフト

「7 R」を参照してください。

リトルエンディアンシステム

最下位バイトを最初に格納するシステム。エンディアンネスも参照してください。

LLM

「大規模言語モデル」を参照してください。

下位環境

環境を参照してください。

M

機械学習 (ML)

パターン認識と学習にアルゴリズムと手法を使用する人工知能の一種。ML は、モノのインターネット (IoT) データなどの記録されたデータを分析して学習し、パターンに基づく統計モデルを生成します。詳細については、「機械学習」を参照してください。

メインブランチ

「ブランチ」を参照してください。

マルウェア

コンピュータのセキュリティまたはプライバシーを侵害するように設計されているソフトウェア。マルウェアは、コンピュータシステムの中断、機密情報の漏洩、不正アクセスにつながる可能性があります。マルウェアの例としては、ウイルス、ワーム、ランサムウェア、トロイの木馬、スパイウェア、キーロガーなどがあります。

マネージドサービス

AWS のサービス はインフラストラクチャレイヤー、オペレーティングシステム、プラットフォームを AWS 運用し、ユーザーはエンドポイントにアクセスしてデータを保存および取得します。Amazon Simple Storage Service (Amazon S3) と Amazon DynamoDB は、 マネージドサービスの例です。これらは抽象化されたサービスとも呼ばれます。

製造実行システム (MES)

生産プロセスを追跡、モニタリング、文書化、制御するためのソフトウェアシステム。このソフトウェアシステムは、生産プロセスで、生産品目を工場の完成製品に変換します。

MAP

「移行促進プログラム」を参照してください。

メカニズム

ツールを作成し、ツールの導入を推進し、調整のために結果を検査する完全なプロセス。メカニズムは、動作中にそれ自体を強化および改善するサイクルです。詳細については、 AWS Well-Architected フレームワークの「メカニズムの構築」を参照してください。

メンバーアカウント

組織の一部である管理アカウント AWS アカウント 以外のすべて AWS Organizations。アカウントが組織のメンバーになることができるのは、一度に 1 つのみです。

MES

「製造実行システム」を参照してください。

メッセージキューイングテレメトリトランスポート (MQTT)

リソースに制約のある IoT デバイス用の、 machine-to-machineパブリッシュ/サブスクライブパターンに基づく軽量 (M2M) 通信プロトコル。

マイクロサービス

明確に定義された上で通信APIsし、通常は小規模で自己完結型のチームが所有する、小規模で独立したサービス。例えば、保険システムには、販売やマーケティングなどのビジネス機能、また

は購買、請求、分析などのサブドメインにマッピングするマイクロサービスが含まれる場合があります。マイクロサービスの利点には、俊敏性、柔軟なスケーリング、容易なデプロイ、再利用可能なコード、回復力などがあります。詳細については、AWS「サーバーレスサービスを使用したマイクロサービスの統合」を参照してください。

マイクロサービスアーキテクチャ

各アプリケーションプロセスをマイクロサービスとして実行する独立したコンポーネントを使用してアプリケーションを構築するアプローチ。これらのマイクロサービスは、軽量な を使用して明確に定義されたインターフェイスを介して通信しますAPIs。このアーキテクチャの各マイクロサービスは、アプリケーションの特定の機能に対する需要を満たすように更新、デプロイ、およびスケーリングできます。詳細については、「 でのマイクロサービスの実装 AWS」を参照してください。

Migration Acceleration Program (MAP)

組織がクラウドへの移行のための強固な運用基盤を構築し、移行の初期コストを相殺するのに役立つコンサルティングサポート、トレーニング、サービスを提供する AWS プログラム。 MAPには、従来の移行を体系的に実行するための移行方法論と、一般的な移行シナリオを自動化して高速化するための一連のツールが含まれています。

大規模な移行

アプリケーションポートフォリオの大部分を次々にクラウドに移行し、各ウェーブでより多くのアプリケーションを高速に移動させるプロセス。この段階では、以前の段階から学んだベストプラクティスと教訓を使用して、移行ファクトリー チーム、ツール、プロセスのうち、オートメーションとアジャイルデリバリーによってワークロードの移行を合理化します。これは、AWS 移行戦略 の第 3 段階です。

移行ファクトリー

自動化された俊敏性のあるアプローチにより、ワークロードの移行を合理化する部門横断的なチーム。移行ファクトリーチームには、通常、運用、ビジネスアナリストと所有者、移行エンジニア、デベロッパー、スプリントに取り組む DevOps プロフェッショナルが含まれます。エンタープライズアプリケーションポートフォリオの 20～50% は、ファクトリーのアプローチによって最適化できる反復パターンで構成されています。詳細については、このコンテンツセットの移行ファクトリーに関する解説とCloud Migration Factory ガイドを参照してください。

移行メタデータ

移行を完了するために必要なアプリケーションおよびサーバーに関する情報。移行パターンごとに、異なる一連の移行メタデータが必要です。移行メタデータの例には、ターゲットサブネット、セキュリティグループ、 AWS アカウントなどがあります。

移行パターン

移行戦略、移行先、および使用する移行アプリケーションまたはサービスを詳述する、反復可能な移行タスク。例: AWS Application Migration Service EC2を使用して Amazon への移行をリホストします。

移行ポートフォリオ評価 (MPA)

に移行するためのビジネスケースを検証するための情報を提供するオンラインツール AWS クラウド。 MPAは、詳細なポートフォリオ評価 (サーバーの適切なサイズ設定、料金設定、TCO比較、移行コスト分析) と移行計画 (アプリケーションデータ分析とデータ収集、アプリケーショングループ化、移行の優先順位付け、ウェーブプランニング) を提供します。MPA ツール (ログインが必要) は、すべての AWS コンサルタントとAPNパートナーコンサルタントが無料で利用できます。

移行準備状況評価 (MRA)

を使用して、組織のクラウド準備状況に関するインサイトを取得し、長所と短所を特定し、特定されたギャップを埋めるためのアクションプランを構築するプロセス AWS CAF。詳細については、「移行準備ガイド」を参照してください。 MRAはAWS 移行戦略の最初のフェーズです。

移行戦略

ワークロードを に移行するために使用されるアプローチ AWS クラウド。詳細については、この用語集の「7 Rs エントリ」と「組織を動員して大規模な移行を加速する」を参照してください。

ML

???「機械学習」を参照してください。

モダナイゼーション

古い (レガシーまたはモノリシック) アプリケーションとそのインフラストラクチャをクラウド内の俊敏で弾力性のある高可用性システムに変換して、コストを削減し、効率を高め、イノベーションを活用します。詳細については、「」の「アプリケーションをモダナイズするための戦略 AWS クラウド」を参照してください。

モダナイゼーション準備状況評価

組織のアプリケーションのモダナイゼーションの準備状況を判断し、利点、リスク、依存関係を特定し、組織がこれらのアプリケーションの将来の状態をどの程度適切にサポートできるかを決定するのに役立つ評価。評価の結果として、ターゲットアーキテクチャのブループリント、モダナイゼーションプロセスの開発段階とマイルストーンを詳述したロードマップ、特定され

たギャップに対処するためのアクションプランが得られます。詳細については、「」の「アプリケーションのモダナイゼーション準備状況の評価 AWS クラウド」を参照してください。

モノリシックアプリケーション (モノリス)

緊密に結合されたプロセスを持つ単一のサービスとして実行されるアプリケーション。モノリシックアプリケーションにはいくつかの欠点があります。1 つのアプリケーション機能エクスペリエンスの需要が急増する場合は、アーキテクチャ全体をスケーリングする必要があります。モノリシックアプリケーションの特徴を追加または改善することは、コードベースが大きくなると複雑になります。これらの問題に対処するには、マイクロサービスアーキテクチャを使用できます。詳細については、モノリスをマイクロサービスに分解する を参照してください。

MPA

「移行ポートフォリオ評価」を参照してください。

MQTT

「Message Queuing Telemetry Transport」を参照してください。

多クラス分類

複数のクラスの予測を生成するプロセス (2 つ以上の結果の 1 つを予測します)。例えば、機械学習モデルが、「この製品は書籍、自動車、電話のいずれですか?」 または、「このお客様にとって最も関心のある商品のカテゴリはどれですか?」と聞くかもしれません。

ミュータブルインフラストラクチャ

本番ワークロードの既存のインフラストラクチャを更新および変更するモデル。Well-Architected AWS フレームワークでは、一貫性、信頼性、予測可能性を向上させるために、イミュータブルインフラストラクチャをベストプラクティスとして使用することを推奨しています。

O

OAC

「オリジンアクセスコントロール」を参照してください。

OAI

「オリジンアクセスアイデンティティ」を参照してください。

OCM

「組織の変更管理」を参照してください。

オフライン移行

移行プロセス中にソースワークロードを停止させる移行方法。この方法はダウンタイムが長くなるため、通常は重要ではない小規模なワークロードに使用されます。

OI

「 オペレーションの統合」を参照してください。

OLA

「運用レベルの契約」を参照してください。

オンライン移行

ソースワークロードをオフラインにせずにターゲットシステムにコピーする移行方法。ワークロードに接続されているアプリケーションは、移行中も動作し続けることができます。この方法はダウンタイムがゼロから最小限で済むため、通常は重要な本番稼働環境のワークロードに使用されます。

OPC-UA

「オープンプロセス通信 - 統合アーキテクチャ」を参照してください。

オープンプロセス通信 - 統合アーキテクチャ (OPC-UA)

産業オートメーション用の machine-to-machine (M2M) 通信プロトコル。OPC-UA は、データの暗号化、認証、認可スキームを備えた相互運用性標準を提供します。

運用レベルの契約 (OLA)

サービスレベルアグリーメント () をサポートするために、 が相互に提供することを約束する機能的な IT グループを明確にする契約SLA。

運用準備状況レビュー (ORR)

インシデントや潜在的な障害の理解、評価、防止、または範囲の縮小に役立つ質問のチェックリストおよび関連するベストプラクティス。詳細については、 AWS Well-Architected フレームワークの「運用準備状況レビュー (ORR)」を参照してください。

運用テクノロジー (OT)

産業オペレーション、機器、インフラストラクチャを制御するために物理環境と連携するハードウェアおよびソフトウェアシステム。製造では、OT と情報技術 (IT) システムの統合が、Industry 4.0 トランスフォーメーションの主な焦点です。

オペレーション統合 (OI)

クラウドでオペレーションをモダナイズするプロセスには、準備計画、オートメーション、統合が含まれます。詳細については、オペレーション統合ガイド を参照してください。

組織の証跡

組織 AWS アカウント 内のすべての のすべてのイベント AWS CloudTrail をログに記録する、 によって作成された証跡 AWS Organizations。証跡は、組織に含まれている各 AWS アカウント に作成され、各アカウントのアクティビティを追跡します。詳細については、 ドキュメントの「組織の証跡の作成」を参照してください。 CloudTrail

組織の変更管理 (OCM)

人材、文化、リーダーシップの観点から、破壊的で重要なビジネス変革を管理するためのフレームワーク。 OCMは、変化の導入を加速し、移行問題に対処し、文化的および組織的な変化を促進することで、組織が新しいシステムや戦略の準備と移行を支援します。 AWS 移行戦略では、クラウド導入プロジェクトに必要な変化のスピードから、このフレームワークは人材アクセラレーションと呼ばれます。詳細については、 OCMガイドを参照してください。

オリジンアクセスコントロール (OAC)

では CloudFront、Amazon Simple Storage Service (Amazon S3) コンテンツを保護するためのアクセスを制限するための拡張オプションです。 OACは、すべての S3 バケット、 AWS KMS (-SSEKMS) によるサーバー側の暗号化 AWS リージョン、および S3 バケットへの動的 PUT および DELETEリクエストをサポートします。

オリジンアクセスアイデンティティ (OAI)

では CloudFront、Amazon S3 コンテンツを保護するためのアクセスを制限するオプションです。を使用するとOAI、 は Amazon S3 が認証できるプリンシパル CloudFront を作成します。認証されたプリンシパルは、特定の CloudFront ディストリビューションを介してのみ S3 バケット内のコンテンツにアクセスできます。「」も参照してください。より詳細で拡張OACされたアクセスコントロールを提供します。

ORR

「運用準備状況レビュー」を参照してください。

OT

「運用テクノロジー」を参照してください。

アウトバウンド (出力) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーション内から開始されたネットワーク接続VPCを処理する です。AWS セキュリティリファレンスアーキテクチャでは、アプリケーションとより広範なインターネット間の双方向インターフェイスVPCsを保護するために、インバウンド、アウトバウンド、検査を使用してネットワークアカウントを設定することをお勧めします。

P

アクセス許可の境界

ユーザーまたはロールが持つことができるアクセス許可の上限を設定するためにIAMプリンシパルにアタッチされるIAM管理ポリシー。詳細については、 IAMドキュメントの「アクセス許可の境界」を参照してください。

個人を特定できる情報 (PII)

直接閲覧した場合、または他の関連データと組み合わせた場合に、個人の身元を合理的に推測するために使用できる情報。例としては、名前、住所、連絡先情報PIIなどがあります。

PII

個人を特定できる情報を参照してください。

プレイブック

クラウドでのコアオペレーション機能の提供など、移行に関連する作業を取り込む、事前定義された一連のステップ。プレイブックは、スクリプト、自動ランブック、またはお客様のモダナイズされた環境を運用するために必要なプロセスや手順の要約などの形式をとることができます。

PLC

「プログラム可能なロジックコントローラー」を参照してください。

PLM

「製品ライフサイクル管理」を参照してください。

ポリシー

アクセス許可の定義 (アイデンティティベースのポリシーを参照)、アクセス条件の指定 (リソースベースのポリシーを参照)、または の組織内のすべてのアカウントに対する最大アクセス許可の定義 AWS Organizations （ サービスコントロールポリシーを参照) が可能なオブジェクト。

多言語の永続性

データアクセスパターンやその他の要件に基づいて、マイクロサービスのデータストレージテクノロジーを個別に選択します。マイクロサービスが同じデータストレージテクノロジーを使用している場合、実装上の問題が発生したり、パフォーマンスが低下する可能性があります。マイクロサービスは、要件に最も適合したデータストアを使用すると、より簡単に実装でき、パフォーマンスとスケーラビリティが向上します。詳細については、マイクロサービスでのデータ永続性の有効化 を参照してください。

ポートフォリオ評価

移行を計画するために、アプリケーションポートフォリオの検出、分析、優先順位付けを行うプロセス。詳細については、「移行準備状況ガイド」を参照してください。

述語

`true` または を返すクエリ条件。通常は `false`WHERE句にあります。

述語のプッシュダウン

転送前にクエリ内のデータをフィルタリングするデータベースクエリ最適化手法。これにより、リレーショナルデータベースから取得して処理する必要があるデータの量が減少し、クエリのパフォーマンスが向上します。

予防的コントロール

イベントの発生を防ぐように設計されたセキュリティコントロール。このコントロールは、ネットワークへの不正アクセスや好ましくない変更を防ぐ最前線の防御です。詳細については、Implementing security controls on AWSのPreventative controlsを参照してください。

プリンシパル

アクションを実行し AWS 、リソースにアクセスできる のエンティティ。このエンティティは通常、、 AWS アカウント IAMロール、または ユーザーのルートユーザーです。詳細については、IAMドキュメントの「ロールの用語と概念」の「プリンシパル」を参照してください。

プライバシーバイデザイン

開発プロセス全体でプライバシーを考慮するシステムエンジニアリングアプローチ。

プライベートホストゾーン

Amazon Route 53 が 1 つ以上の 内のドメインとそのサブドメインのDNSクエリにどのように応答するかに関する情報を保持するコンテナVPCs。詳細については、Route 53 ドキュメントの「プライベートホストゾーンの使用」を参照してください。

プロアクティブコントロール

非準拠リソースのデプロイを防ぐように設計されたセキュリティコントロール。これらのコントロールは、プロビジョニングされる前にリソースをスキャンします。リソースがコントロールに準拠していない場合、プロビジョニングされません。詳細については、 AWS Control Tower ドキュメントの「コントロールリファレンスガイド」および「セキュリティコントロールの実装」の「プロアクティブコントロール」を参照してください。 AWS

製品ライフサイクル管理 (PLM)

設計、開発、発売から成長と成熟、辞退と削除まで、ライフサイクル全体にわたる製品のデータとプロセスの管理。

本番環境

???「環境」を参照してください。

プログラム可能なロジックコントローラー (PLC)

製造では、マシンをモニタリングし、承認プロセスを自動化する、信頼性が高く、適応性の高いコンピュータです。

プロンプトの連鎖

1 つのLLMプロンプトの出力を次のプロンプトの入力として使用して、より良いレスポンスを生成します。この手法は、複雑なタスクをサブタスクに分割したり、事前レスポンスを繰り返し調整または拡張したりするために使用されます。これにより、モデルのレスポンスの精度と関連性が向上し、より詳細でパーソナライズされた結果が得られます。

仮名化

データセット内の個人識別子をプレースホルダー値に置き換えるプロセス。仮名化は個人のプライバシー保護に役立ちます。仮名化されたデータは、依然として個人データとみなされます。

publish/subscribe (pub/sub)

マイクロサービス間の非同期通信を可能にするパターン。スケーラビリティと応答性を向上させます。たとえば、マイクロサービスベースの ではMES、マイクロサービスは他のマイクロサービスがサブスクライブできるチャネルにイベントメッセージを発行できます。システムは、公開サービスを変更せずに新しいマイクロサービスを追加できます。

Q

クエリプラン

SQL リレーショナルデータベースシステム内のデータにアクセスするために使用される手順などの一連のステップ。

クエリプランのリグレッション

データベースサービスのオプティマイザーが、データベース環境に特定の変更が加えられる前に選択されたプランよりも最適性の低いプランを選択すること。これは、統計、制限事項、環境設定、クエリパラメータのバインディングの変更、およびデータベースエンジンの更新などが原因である可能性があります。

R

RACI マトリックス

責任、説明責任、相談、通知 (RACI) を参照してください。

RAG

「取得拡張生成」を参照してください。

ランサムウェア

決済が完了するまでコンピュータシステムまたはデータへのアクセスをブロックするように設計された、悪意のあるソフトウェア。

RASCI マトリックス

責任、説明責任、相談、通知 (RACI) を参照してください。

RCAC

「行と列のアクセスコントロール」を参照してください。

リードレプリカ

読み取り専用に使用されるデータベースのコピー。クエリをリードレプリカにルーティングして、プライマリデータベースへの負荷を軽減できます。

再構築

「7 Rs」を参照してください。

目標復旧時点 (RPO)

最後のデータリカバリポイントからの最大許容時間です。これにより、最後の回復時点からサービスが中断されるまでの間に許容できるデータ損失の程度が決まります。

目標復旧時間 (RTO)

サービスの中断から復旧までの最大許容遅延時間。

リファクタリング

「7 Rs」を参照してください。

リージョン

地理的エリア内の AWS リソースのコレクション。各 AWS リージョン は、耐障害性、安定性、耐障害性を提供するために、他の とは分離され、独立しています。詳細については、AWS リージョン 「アカウントで使用できる を指定する」を参照してください。

回帰

数値を予測する機械学習手法。例えば、「この家はどれくらいの値段で売れるでしょうか?」という問題を解決するために、機械学習モデルは、線形回帰モデルを使用して、この家に関する既知の事実 (平方フィートなど) に基づいて家の販売価格を予測できます。

リホスト

「7 R」を参照してください。

リリース

デプロイプロセスで、変更を本番環境に昇格させること。

再配置

「7 R」を参照してください。

プラットフォーム変更

「7 R」を参照してください。

再購入

「7 R」を参照してください。

回復性

中断に耐えたり、中断から回復したりするアプリケーションの機能。で障害耐性を計画するときは、高可用性とディザスタリカバリがよく考慮されます AWS クラウド。詳細については、AWS クラウド 「耐障害性」を参照してください。

リソースベースのポリシー

Amazon S3 バケット、エンドポイント、暗号化キーなどのリソースにアタッチされたポリシー。このタイプのポリシーは、アクセスが許可されているプリンシパル、サポートされているアクション、その他の満たすべき条件を指定します。

責任、説明責任、相談、情報 (RACI) マトリックス

移行活動とクラウド運用に関わるすべての関係者の役割と責任を定義したマトリックス。マトリックスの名前は、マトリックスで定義されている責任の種類、すなわち責任 (R) 、説明責任 (A) 、協議 (C) 、情報提供 (I) に由来します。サポート (S) タイプはオプションです。サポートを含めると、マトリックスはRASCIマトリックスと呼ばれ、除外するとRACIマトリックスと呼ばれます。

レスポンシブコントロール

有害事象やセキュリティベースラインからの逸脱について、修復を促すように設計されたセキュリティコントロール。詳細については、Implementing security controls on AWSのResponsive controlsを参照してください。

保持

「7 Rs」を参照してください。

廃止

「7 Rs」を参照してください。

取得拡張生成 (RAG)

レスポンスを生成する前に、 がトレーニングデータソースの外部にある信頼できるデータソースLLMを参照する生成 AI テクノロジー。例えば、RAGモデルは組織のナレッジベースやカスタムデータのセマンティック検索を実行する場合があります。詳細については、「RAG とは」を参照してください。

ローテーション

定期的にシークレットを更新して、攻撃者が認証情報にアクセスするのをより困難にするプロセス。

行と列のアクセスコントロール (RCAC)

アクセスルールが定義されている基本的で柔軟なSQL式を使用します。 RCACは、行のアクセス許可と列マスクで構成されます。

RPO

「目標復旧時点」を参照してください。

RTO

目標復旧時間を参照してください。

ランブック

特定のタスクを実行するために必要な手動または自動化された一連の手順。これらは通常、エラー率の高い反復操作や手順を合理化するために構築されています。

S

SAML 2.0

多くの ID プロバイダー (IdPs) が使用するオープンスタンダード。この機能により、フェデレーティッドシングルサインオン (SSO) が有効になるため、ユーザーは にログイン AWS Management Console したり、 AWS API オペレーションを呼び出したりできます。組織内のすべてのユーザーIAMに対して でユーザーを作成する必要はありません。SAML 2.0 ベースのフェデレーションの詳細については、 IAMドキュメントの「About SAML 2.0-based federation」を参照してください。

SCADA

「監視コントロールとデータ取得」を参照してください。

SCP

「サービスコントロールポリシー」を参照してください。

シークレット

暗号化された形式で保存するパスワードやユーザー認証情報などの AWS Secrets Manager機密情報または制限付き情報。シークレット値とそのメタデータで構成されます。シークレット値は、バイナリ、1 つの文字列、または複数の文字列にすることができます。詳細については、Secrets Manager ドキュメントの「Secrets Manager シークレットの内容」を参照してください。

設計によるセキュリティ

開発プロセス全体でセキュリティを考慮するシステムエンジニアリングアプローチ。

セキュリティコントロール

脅威アクターによるセキュリティ脆弱性の悪用を防止、検出、軽減するための、技術上または管理上のガードレール。セキュリティコントロールには、予防的、検出的、応答的、プロアクティブの 4 つの主なタイプがあります。

セキュリティ強化

アタックサーフェスを狭めて攻撃への耐性を高めるプロセス。このプロセスには、不要になったリソースの削除、最小特権を付与するセキュリティのベストプラクティスの実装、設定ファイル内の不要な機能の無効化、といったアクションが含まれています。

セキュリティ情報とイベント管理 (SIEM) システム

セキュリティ情報管理 (SIM) システムとセキュリティイベント管理 (SEM) システムを組み合わせたツールとサービス。SIEM システムは、サーバー、ネットワーク、デバイス、その他のソースからデータを収集、モニタリング、分析して、脅威やセキュリティ違反を検出し、アラートを生成します。

セキュリティレスポンスの自動化

セキュリティイベントに自動的に応答または修正するように設計された、事前定義されたプログラムされたアクション。これらの自動化は、セキュリティのベストプラクティスを実装するのに役立つ検出的または応答的な AWS セキュリティコントロールとして機能します。自動レスポンスアクションの例には、VPCセキュリティグループの変更、Amazon EC2インスタンスへのパッチ適用、認証情報のローテーションなどがあります。

サーバー側の暗号化

送信先で、それを受け取る AWS のサービス によるデータの暗号化。

サービスコントロールポリシー (SCP)

の組織内のすべてのアカウントのアクセス許可を一元的に制御するポリシー AWS Organizations。 は、ガードレールSCPsを定義するか、管理者がユーザーまたはロールに委任できるアクションの制限を設定します。を許可リストまたは拒否リストSCPsとして使用して、許可または禁止するサービスまたはアクションを指定できます。詳細については、 AWS Organizations ドキュメントの「サービスコントロールポリシー」を参照してください。

サービスエンドポイント

のエントリポイントURLの AWS のサービス。ターゲットサービスにプログラムで接続するには、エンドポイントを使用します。詳細については、AWS 全般のリファレンスの「AWS のサービス エンドポイント」を参照してください。

サービスレベルアグリーメント (SLA)

サービスのアップタイムやパフォーマンスなど、IT チームがお客様に提供すると約束したものを明示した合意書。

サービスレベルインジケータ (SLI)

エラー率、可用性、スループットなど、サービスのパフォーマンス側面の測定。

サービスレベルの目標 (SLO)

サービスレベルのインジケータによって測定される、サービスの正常性を表すターゲットメトリクス。

責任共有モデル

クラウドのセキュリティとコンプライアンス AWS について と共有する責任を説明するモデル。AWS はクラウドのセキュリティを担当しますが、お客様はクラウド内のセキュリティを担当します。詳細については、責任共有モデルを参照してください。

SIEM

セキュリティ情報とイベント管理システムを参照してください。

単一障害点 (SPOF)

システムを中断する可能性のある、アプリケーションの 1 つの重要なコンポーネントの障害。

SLA

「サービスレベルアグリーメント」を参照してください。

SLI

「サービスレベルインジケータ」を参照してください。

SLO

「サービスレベルの目標」を参照してください。

split-and-seed モデル

モダナイゼーションプロジェクトのスケーリングと加速のためのパターン。新機能と製品リリースが定義されると、コアチームは解放されて新しい製品チームを作成します。これにより、お客様の組織の能力とサービスの拡張、デベロッパーの生産性の向上、迅速なイノベーションのサポートに役立ちます。詳細については、「」の「アプリケーションをモダナイズするための段階的アプローチ AWS クラウド」を参照してください。

SPOF

単一障害点を参照してください。

star スキーマ

トランザクションデータまたは測定データを保存するために 1 つの大きなファクトテーブルを使用し、データ属性を保存するために 1 つ以上の小さなディメンションテーブルを使用するデータベース組織構造。この構造は、データウェアハウスまたはビジネスインテリジェンスの目的で使用するために設計されています。

strangler fig パターン

レガシーシステムが廃止されるまで、システム機能を段階的に書き換えて置き換えることにより、モノリシックシステムをモダナイズするアプローチ。このパターンは、宿主の樹木から根を成長させ、最終的にその宿主を包み込み、宿主に取って代わるイチジクのつるを例えています。そのパターンは、モノリシックシステムを書き換えるときのリスクを管理する方法として Martin Fowler により提唱されました。このパターンを適用する方法の例については、「従来の Microsoft ASP.NET (ASMX) ウェブサービスをコンテナと Amazon API Gateway を使用して段階的にモダナイズする」を参照してください。

サブネット

の IP アドレスの範囲VPC。サブネットは、1 つのアベイラビリティーゾーンに存在する必要があります。

監視コントロールとデータ収集 (SCADA)

製造では、ハードウェアとソフトウェアを使用して物理アセットと生産オペレーションをモニタリングするシステム。

対称暗号化

データの暗号化と復号に同じキーを使用する暗号化のアルゴリズム。

合成テスト

ユーザーとのやり取りをシミュレートして潜在的な問題を検出したり、パフォーマンスをモニタリングしたりする方法でシステムをテストします。Amazon Synthetics CloudWatch を使用してこれらのテストを作成できます。

システムプロンプト

動作を指示LLMするために、コンテキスト、指示、またはガイドラインを に提供するための手法。システムプロンプトは、コンテキストを設定し、ユーザーとのやり取りのルールを確立するのに役立ちます。

T

tags

AWS リソースを整理するためのメタデータとして機能するキーと値のペア。タグは、リソースの管理、識別、整理、検索、フィルタリングに役立ちます。詳細については、「AWS リソースのタグ付け」を参照してください。

ターゲット変数

監督された機械学習でお客様が予測しようとしている値。これは、結果変数 のことも指します。例えば、製造設定では、ターゲット変数が製品の欠陥である可能性があります。

タスクリスト

ランブックの進行状況を追跡するために使用されるツール。タスクリストには、ランブックの概要と完了する必要のある一般的なタスクのリストが含まれています。各一般的なタスクには、推定所要時間、所有者、進捗状況が含まれています。

テスト環境

???「環境」を参照してください。

トレーニング

お客様の機械学習モデルに学習するデータを提供すること。トレーニングデータには正しい答えが含まれている必要があります。学習アルゴリズムは入力データ属性をターゲット (お客様が予測したい答え) にマッピングするトレーニングデータのパターンを検出します。これらのパターンをキャプチャする機械学習モデルを出力します。そして、お客様が機械学習モデルを使用して、ターゲットがわからない新しいデータでターゲットを予測できます。

トランジットゲートウェイ

VPCs とオンプレミスのネットワークを相互接続するために使用できるネットワークトランジットハブ。詳細については、 AWS Transit Gateway ドキュメントの「トランジットゲートウェイとは」を参照してください。

トランクベースのワークフロー

デベロッパーが機能ブランチで機能をローカルにビルドしてテストし、その変更をメインブランチにマージするアプローチ。メインブランチはその後、開発環境、本番前環境、本番環境に合わせて順次構築されます。

信頼されたアクセス

ユーザーに代わって AWS Organizations およびそのアカウントで組織内のタスクを実行するために指定したサービスにアクセス許可を付与します。信頼されたサービスは、サービスにリンクされたロールを必要なときに各アカウントに作成し、ユーザーに代わって管理タスクを実行します。詳細については、 ドキュメントの「 AWS Organizations を他の AWS サービスで使用する AWS Organizations 」を参照してください。

チューニング

機械学習モデルの精度を向上させるために、お客様のトレーニングプロセスの側面を変更する。例えば、お客様が機械学習モデルをトレーニングするには、ラベル付けセットを生成し、ラベルを追加します。これらのステップを、異なる設定で複数回繰り返して、モデルを最適化します。

ツーピザチーム

ピザを 2 つ用意できる小さな DevOps チーム。ツーピザチームの規模では、ソフトウェア開発におけるコラボレーションに最適な機会が確保されます。

U

不確実性

予測機械学習モデルの信頼性を損なう可能性がある、不正確、不完全、または未知の情報を指す概念。不確実性には、次の 2 つのタイプがあります。認識論的不確実性は、限られた、不完全なデータによって引き起こされ、弁論的不確実性 は、データに固有のノイズとランダム性によって引き起こされます。詳細については、深層学習システムにおける不確実性の定量化 ガイドを参照してください。

未分化なタスク

ヘビーリフティングとも呼ばれ、アプリケーションの作成と運用には必要だが、エンドユーザーに直接的な価値をもたらさなかったり、競争上の優位性をもたらしたりしない作業です。未分化なタスクの例としては、調達、メンテナンス、キャパシティプランニングなどがあります。

上位環境

環境を参照してください。

V

バキューミング

ストレージを再利用してパフォーマンスを向上させるために、増分更新後にクリーンアップを行うデータベースのメンテナンス操作。

バージョンコントロール

リポジトリ内のソースコードへの変更など、変更を追跡するプロセスとツール。

VPC ピアリング

プライベート IP アドレスを使用してトラフィックをルーティングVPCsできる 2 つの 間の接続。詳細については、Amazon VPCドキュメントのVPC「What is peering」を参照してください。

脆弱性

システムのセキュリティを脅かすソフトウェアまたはハードウェアの欠陥。

W

ウォームキャッシュ

頻繁にアクセスされる最新の関連データを含むバッファキャッシュ。データベースインスタンスはバッファキャッシュから、メインメモリまたはディスクからよりも短い時間で読み取りを行うことができます。

ウォームデータ

アクセス頻度の低いデータ。この種類のデータをクエリする場合、通常は適度に遅いクエリでも問題ありません。

ウィンドウ関数

現在のレコードに関連する行のグループに対して計算を実行するSQL関数。ウィンドウ関数は、移動平均の計算や、現在の行の相対位置に基づく行の値へのアクセスなどのタスクの処理に役立ちます。

ワークロード

ビジネス価値をもたらすリソースとコード (顧客向けアプリケーションやバックエンドプロセスなど) の総称。

ワークストリーム

特定のタスクセットを担当する移行プロジェクト内の機能グループ。各ワークストリームは独立していますが、プロジェクト内の他のワークストリームをサポートしています。たとえば、ポートフォリオワークストリームは、アプリケーションの優先順位付け、ウェーブ計画、および移行メタデータの収集を担当します。ポートフォリオワークストリームは、これらの設備を移行ワークストリームで実現し、サーバーとアプリケーションを移行します。

WORM

「書き込み 1 回」、「読み取り数」を参照してください。

WQF

AWS 「ワークロード認定フレームワーク」を参照してください。

Write Once、Read Many (WORM)

データを 1 回書き込み、データの削除や変更を防ぐストレージモデル。許可されたユーザーは、必要な回数だけデータを読み取ることができますが、変更することはできません。このデータストレージインフラストラクチャはイミュータブルと見なされます。

Z

ゼロデイエクスプロイト

ゼロデイ脆弱性を利用する攻撃、通常はマルウェア。

ゼロデイ脆弱性

実稼働システムにおける未解決の欠陥または脆弱性。脅威アクターは、このような脆弱性を利用してシステムを攻撃する可能性があります。開発者は、よく攻撃の結果で脆弱性に気付きます。

ゼロショットプロンプト

タスクを実行するための指示を LLMに提供しますが、そのガイドに役立つ例 (ショット) はありません。は、事前トレーニング済みの知識を使用してタスクを処理するLLM必要があります。ゼロショットプロンプトの有効性は、タスクの複雑さとプロンプトの品質によって異なります。「数ショットプロンプト」も参照してください。

ゾンビアプリケーション

平均使用量CPUとメモリ使用量が 5% 未満のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するのが一般的です。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。