AWS CDK レイヤーガイド

AWS 規範ガイダンス

AWS 規範ガイダンス: AWS CDK レイヤーガイド

Table of Contents

AWS CDK レイヤーガイド

Amazon Web Services (AWS)、Steven Guggenheimer

2023 年 12 月 (ドキュメント履歴)

AWS Cloud Development Kit (AWS CDK) を支える主な概念の 1 つは、寒い日に暖かく過ごすための考え方によく似ています。この考え方をレイヤリングといいます。寒い日には、シャツの上にジャケットを着、寒さによってはさらに厚めのジャケットを羽織ることもあるでしょう。その状態で室内に入ったときに、暖房が効いている場合は、暑くならないように、一方または両方のレイヤー、つまりジャケットを脱ぐことができます。AWS CDK では、レイヤリングによって、クラウドコンポーネントを使用するためのさまざまな抽象化レベルを実現しています。レイヤリングの仕組みがあれば、Infrastructure as Code (IAC) スタックのデプロイ時に、必要以上に多くのコードを記述したり、アクセス権が少なすぎてリソースプロパティにアクセスできなかったりすることがなくなります。

AWS CDK を使用しない場合は、手動で AWS CloudFormation テンプレートを記述する必要があります。つまり、1 つのレイヤーしか活用できず、通常必要な数よりもはるかに多くのコードを記述せざるを得なくなります。一方、CloudFormation では通常、記述が不要なものの、それらが AWS CDK によってすべて抽象化されると、エッジケースに対応できなくなります。

この問題に対処するために、AWS CDK では、リソースのプロビジョニングを 3 つの異なるレイヤーに分割します。

- レイヤー 1 – CloudFormation レイヤー: 最も基本的なレイヤーであり、CloudFormation と AWS CDK のリソースは、ほぼ同じです。
- レイヤー 2 – キュレートされたレイヤー: CloudFormation リソースがプログラムクラスに抽象化されるとともに、内部にある CloudFormation ボイラープレート構文の多くが合理化されるレイヤーであり、AWS CDK のほぼ全体を占めています。
- レイヤー 3 – パターンレイヤー: 最も抽象化されたレイヤーであり、ここでは、レイヤー 1 と 2 が備える構成要素を使用して、特定のユースケースに合うようにコードをカスタマイズできます。

各レイヤーの各項目は、特別な AWS CDK クラスのインスタンスであり、これを `Construct` と呼びます。AWS ドキュメント では、こう説明しています。「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、クラウドコンポーネントを表すものです。これによって、AWS CloudFormation でのコンポーネント作成に必要なあらゆる要素をカプセル化します」。こうしたレイヤー内のコンストラクトを、それらが属するレイヤーに応じて L1、L2、L3 コンストラクトと呼び

ます。このガイドでは、各 AWS CDK レイヤーを解説しながら、その用途や、レイヤーが重要である理由を明らかにしていきます。

このガイドは、AWS CDK が動作するための主な概念を深く掘り下げたい技術マネージャー、リーダー、開発者を対象にしています。AWS CDK は、広く利用されているツールですが、それが持つ機能の大部分が見落とされがちです。このガイドが示す概念を理解し始めると、新たな可能性の世界を導き出し、チームのリソースプロビジョニングプロセスを最適化できるでしょう。

このガイドの内容

レイヤー 1 コンストラクト

L1 コンストラクトは AWS CDK の構成要素であり、Cfn というプレフィックスが付くため、他のコンストラクトと簡単に区別できます。例えば、AWS CDK の Amazon DynamoDB パッケージには、L2 コンストラクトである Table コンストラクトが含まれています。対応する L1 コンストラクトを CfnTable と呼び、これによって、CloudFormation DynamoDB Table を直接表します。通常、AWS CDK アプリケーションで L1 コンストラクトを直接使用することはありませんが、この最初のレイヤーにアクセスせずに AWS CDK を使用することは不可能です。ただし、ほとんどの場合、開発者が使い慣れている L2 および L3 コンストラクトは、L1 コンストラクトに大きく依存しているため、L1 コンストラクトは CloudFormation と AWS CDK 間の橋渡しをするものと考えると良いでしょう。

AWS CDK は、標準のコーディング言語を使用して CloudFormation テンプレートを生成することに特化しています。つまり、cdk synth CLI コマンドを実行し、CloudFormation テンプレートが生成されると、AWS CDK は仕事を終えます。利便性に配慮して cdk deploy コマンドが用意されていますが、これを実行すると、その処理は CloudFormation 内ですべて実行されます。CloudFormation で理解可能な形式に AWS CDK コードを変換する際に重要な役割を果たすのが、L1 コンストラクトなのです。

AWS CDK と CloudFormation 間における、L1 コンストラクトのライフサイクル

L1 コンストラクトを作成して使用するプロセスは、以下のステップで構成されます。

1. AWS CDK のビルドプロセスにより、CloudFormation の仕様が L1 コンストラクトの形式でプログラムコードに変換されます。
2. 開発者が、AWS CDK アプリケーションの一部として L1 コンストラクトを直接または間接的に参照するコードを記述します。
3. 開発者が、cdk synth コマンドを実行して、プログラムコードを CloudFormation 仕様 (テンプレート) で指定された形式に変換します。
4. 開発者が、cdk deploy コマンドを実行して、こうしたテンプレート内の CloudFormation スタックを AWS アカウント環境にデプロイします。

簡単な演習を行いましょう。GitHub の AWS CDK オープンソースリポジトリにアクセスして、AWS サービスの 1 つをランダムに選択し、そのサービスの AWS CDK パッケージに移動します (次のよ

うにフォルダをたどります: packages、aws-cdk-lib、aws-<servicename>、lib)。この例では Amazon S3 を選択しますが、どのサービスを使用しても動作します。そのパッケージのメイン index.ts ファイルを確認すると、次の行があります。

```
export * from './s3.generated';
```

ただし、対応するディレクトリのどこにも、s3.generated ファイルはありません。なぜなら、L1 コンストラクトは、AWS CDK ビルドプロセス中に CloudFormation リソース仕様に基づいて自動生成されるからです。そのため、対象のパッケージに AWS CDK ビルドコマンドを実行した後にのみ、パッケージ内に s3.generated が存在することになります。

AWS CloudFormation リソース仕様

AWS CloudFormation リソース仕様は、AWS の Infrastructure as Code (IAC) を定義するとともに、この仕様によって、CloudFormation テンプレート内のコードをどのように AWS アカウントのリソースに変換するかを決定するものです。この仕様には、リージョンごとのレベルで、JSON 形式の AWS リソースが定義されています。各リソースには、provider::service::resource の形式に従い、一意のリソースタイプ名が付与されます。例えば、Amazon S3 バケットのリソースタイプには AWS::S3::Bucket という名前付き、Amazon S3 アクセスポイントのリソースタイプには AWS::S3::AccessPoint という名前が付きます。こうしたリソースタイプは、AWS CloudFormation リソース仕様で定義されている構文を使用して CloudFormation テンプレート内にレンダリングすることもできます。AWS CDK ビルドプロセスを実行すると、各リソースタイプも L1 コンストラクトに変換されます。

したがって、各 L1 コンストラクトは、対応する CloudFormation リソースがプログラムによってそのまま反映されたものです。CloudFormation テンプレートに適用するどのプロパティも、L1 コンストラクトのインスタンス化に使用でき、対応する L1 コンストラクトをインスタンス化する際にも、必要なすべての CloudFormation プロパティを引数に指定しなければなりません。次の表は、CloudFormation テンプレートに表された S3 バケットと、AWS CDK L1 コンストラクトとして定義した同じ S3 バケットを比較したものです。

CloudFormation テンプレート

```
"amzns3demobucket": {
    "Type": "AWS::S3::Bucket",
    "Properties": {
      "BucketName": "amzn-s3-demo-
bucket",
```

L1 コンストラクト

```
new CfnBucket(this, "amzns3de
mobucket", {
  bucketName: "amzn-s3-demo-bucket",
  bucketEncryption: {
```

```
      "BucketEncryption": {
        "ServerSideEncryptionConfig
uration": [
          {
            "ServerSideEncrypt
ionByDefault": {
              "SSEAlgorithm": "AES256"
            }
          }
        ]
      },
      "MetricsConfigurations": [
        {
          "Id": "myConfig"
        }
      ],
      "OwnershipControls": {
        "Rules": [
          {
            "ObjectOwnership":
 "BucketOwnerPreferred"
          }
        ]
      },
      "PublicAccessBlockConfigura
tion": {
        "BlockPublicAcls": true,
        "BlockPublicPolicy": true,
        "IgnorePublicAcls": true,
        "RestrictPublicBuckets": true
      },
      "VersioningConfiguration": {
        "Status": "Enabled"
      }
    }
  }
```

```
    serverSideEncryptionConfigu
ration: [
      {
        serverSideEncryptionByDefau
lt: {
          sseAlgorithm: "AES256"
        }
      }
    ]
  },
  metricsConfigurations: [
    {
      id: "myConfig"
    }
  ],
  ownershipControls: {
    rules: [
      {
        objectOwnership: "BucketOw
nerPreferred"
      }
    ]
  },
  publicAccessBlockConfiguration: {
    blockPublicAcls: true,
    blockPublicPolicy: true,
    ignorePublicAcls: true,
    restrictPublicBuckets: true
  },
  versioningConfiguration: {
    status: "Enabled"
  }
});
```

ご覧のとおり、L1 コンストラクトは、CloudFormation リソースのコードを正確に示すマニフェストであり、ショートカットや単純化が行われないため、記述が必要なボイラープレートテキストの量はほぼ同じです。ただし、AWS CDK を使用する大きな利点の 1 つは、こうした CloudFormation 構文

ボイラープレートの多くを排除しやすくなることです。これを可能にするために、有用なのが、L2 コンストラクトです。

レイヤー 2 コンストラクト

AWS CDK オープンソースリポジトリ内のコードは、主に TypeScript プログラミング言語で記述され、多数のパッケージとモジュールで構成されています。メインパッケージライブラリは、aws-cdk-lib と呼ばれ、AWS サービスごとに 1 つのパッケージに大まかに分割されていますが、常にその状態であるとは限りません。前述したように、L1 コンストラクトは、ビルドプロセス中に自動生成されます。つまり、リポジトリ内に存在するあらゆるコードは、L2 コンストラクトであり、これらは、L1 コンストラクトが抽象化されたものです。

パッケージには、TypeScript タイプ、列挙型、インターフェイスのコレクションに加え、機能を追加するヘルパークラスも含まれていますが、こうした要素はいずれも、L2 コンストラクトに利用します。どの L2 コンストラクトも、インスタンス化された際に、自身のコンストラクタ内にある対応する L1 コンストラクトを呼び出します。作成された L1 コンストラクトには、レイヤー 2 から次のようにアクセスできます。

```
const role = new Bucket(this, "amzn-s3-demo-bucket", {/*...BucketProps*/});
const cfnBucket = role.node.defaultChild;
```

L2 コンストラクトでは、デフォルトのプロパティ、便利なメソッド、その他のシンタックスシュガーを取り込み、L1 コンストラクトに適用します。これにより、CloudFormation でリソースを直接プロビジョニングするために必要な反復と冗長性の大部分を取り除くことができます。

どの L2 コンストラクトでも、対応する L1 コンストラクトが内部に構築されます。しかし、L2 コンストラクトは、L1 コンストラクトを継承することはありません。L1 コンストラクトも L2 コンストラクトも、Construct という特別なクラスを継承します。AWS CDK バージョン 1 の Construct クラスは、開発キットに組み込まれていましたが、バージョン 2 では、独立したスタンドアロンパッケージとして提供されています。そうすることで、Cloud Development Kit for Terraform (CDKTF) などの他のパッケージ内に、依存関係として追加できるからです。Construct クラスを継承するクラスはいずれも、L1、L2、L3 コンストラクトのいずれかです。次の表に示すように、L2 コンストラクトは、このクラスを直接継承しますが、L1 コンストラクトは CfnResource というクラスを継承します。

L1 継承ツリー	L2 継承ツリー
L1 コンストラクト	L2 コンストラクト
→ クラス CfnResource	→ クラス Construct

→ → 抽象クラス CfnRefElement

→ → → 抽象クラス CfnElement

→ → → → クラス Construct

L1 コンストラクトも L2 コンストラクトも Construct クラスを継承しているのに、なぜ L2 コンストラクトは L1 を継承しないのでしょうか。Construct クラスとレイヤー 1 間にあるクラスでは、CloudFormation リソースをそのまま反映したものとして、L1 コンストラクトが特定の位置に固定されるからです。こうしたクラスには、ダウンストリームクラスに含まれる必要のあるメソッドが含まれています。_toCloudFormation がその例ですが、このメソッドは、CloudFormation 構文が直接出力されるよう強制するものです。L2 コンストラクトは、こうしたクラスをスキップし、Construct クラスを直接継承します。そのように、L2 コンストラクタ内で L1 を個別に構築することで、L1 コンストラクトに必要なコードの多くを柔軟に抽象化できるのです。

前のセクションでは、CloudFormation テンプレートの S3 バケットと、L1 コンストラクトとしてレンダリングした同じ S3 バケットを並べて比較しました。この比較では、プロパティと構文がほぼ同じであることがわかり、L1 コンストラクトのコードは、CloudFormation コンストラクトよりも、3 ~ 4 行だけ短くなっていました。今度は、同じ S3 バケットをそれぞれ記述した L1 コンストラクトと L2 コンストラクトを比較してみましょう。

S3 バケットの L1 コンストラクト

```
new CfnBucket(this, "amzns3de
mobucket", {
  bucketName: "amzn-s3-demo-bucket",
  bucketEncryption: {
    serverSideEncryptionConfigu
ration: [
      {
        serverSideEncryptionByDefau
lt: {
          sseAlgorithm: "AES256"
        }
      }
    ]
  },
  metricsConfigurations: [
    {
```

S3 バケットの L2 コンストラクト

```
new Bucket(this, "amzns3demobucket",
 {
  bucketName: "amzn-s3-demo-bucket",
  encryption: BucketEncryption.S
3_MANAGED,
  metrics: [
    {
      id: "myConfig"
    },
  ],
  objectOwnership: ObjectOwnership.BU
CKET_OWNER_PREFERRED,
  blockPublicAccess: BlockPubl
icAccess.BLOCK_ALL,
  versioned: true
});
```

```
      id: "myConfig"
    }
  ],
  ownershipControls: {
    rules: [
      {
        objectOwnership: "BucketOw
nerPreferred"
      }
    ]
  },
  publicAccessBlockConfiguration: {
    blockPublicAcls: true,
    blockPublicPolicy: true,
    ignorePublicAcls: true,
    restrictPublicBuckets: true
  },
  versioningConfiguration: {
    status: "Enabled"
  }
});
```

ご覧のとおり、L2 コンストラクトのサイズは、L1 コンストラクトの半分未満です。L2 コンストラクトでは、このような統合を実現するために多くの手法を使用します。そうした手法の一部は 1 つの L2 コンストラクトに適用しますが、それ以外は、複数のコンストラクトで再利用可能なため、クラスとして独立させ再利用します。L2 コンストラクトでは、複数の方法で CloudFormation 構文を統合します。これについては、以下のセクションで説明します。

デフォルトのプロパティ

リソースプロビジョニングのコードを最も簡単に統合する方法とは、最も一般的なプロパティ設定をデフォルトとして扱うことです。AWS CDK では、強力なプログラミング言語にアクセスできますが、CloudFormation ではそれが行えないため、多くの場合、こうしたデフォルト設定には、その性質上、条件が設定されます。これにより、AWS CDK コードから、数行を削除できる場合があります。CloudFormation の設定は、コンストラクトに渡される他のプロパティ値から推測できるからです。

構造体、型、インターフェイス

AWS CDK は複数のプログラミング言語で使用できますが、ネイティブ言語は TypeScript であるため、L2 コンストラクトを構成する型の定義には、言語の型システムを使用します。そうした型システムの掘り下げは、このガイドの範囲外です。詳細については、TypeScript ドキュメントを参照してください。要約すると、TypeScript の `type` とは、特定の変数が保持するデータの種類を表すもので、こうしたデータは、`string` などの基本的なものから、`object` などの複雑なものまで多岐にわたります。TypeScript オブジェクトの型は、TypeScript `interface` によっても表すことができ、`struct` は、インターフェイスの別名とも言えます。

TypeScript では、構造体という用語を使用しませんが、AWS CDK API リファレンスを見ると、構造体がコード内で TypeScript インターフェイスと同じ概念で使用されていることがわかります。同時に、API リファレンスでは、特定のインターフェイスをインターフェイスと呼んでいます。構造体とインターフェイスは同じ概念を持つにもかかわらず、AWS CDK ドキュメントでは、なぜそれらを区別しているのでしょうか。

AWS CDK では、L2 コンストラクトで使用するオブジェクトを表すインターフェイスを構造体と呼んでいます。これには、インスタンス化中に L2 コンストラクトに渡すプロパティ引数のオブジェクト型も含まれます。例えば、S3 バケットコンストラクトの `BucketProps` や DynamoDB Table コンストラクトの `TableProps`、AWS CDK 内で使用するその他の TypeScript インターフェイスなどです。つまり、それが AWS CDK 内の TypeScript インターフェイスであり、名前の先頭に `I` の文字が付いていないものを AWS CDK では構造体と呼びます。

逆に、AWS CDK でインターフェイスという用語が使用されるのは、プレーンオブジェクトによって特定のコンストラクトまたはヘルパークラスを適切に表現するのに必要な基本要素を表す場合です。つまり、インターフェイスによって、L2 コンストラクトのパブリックプロパティを記述します。AWS CDK のインターフェイス名はいずれも、既存のコンストラクトまたはヘルパークラスの名前にプレフィックス `I` を付けて表します。すべての L2 コンストラクトに、`Construct` クラスが継承され、対応するインターフェイスも実装されます。つまり、L2 コンストラクトの `Bucket` には、`IBucket` インターフェイスが実装されます。

静的メソッド

L2 コンストラクトのすべてのインスタンスは、対応するインターフェイスのインスタンスでもありますが、その逆が成り立つことはありません。この点は、構造体を調べ、必要なデータ型を確認する際に重要となります。構造体に `bucket` というプロパティがあり、そのプロパティにデータ型 `IBucket` が必要な場合、`IBucket` インターフェイスにリストしているプロパティが含まれるオブジェクト、または L2 `Bucket` のインスタンスのいずれかを渡すことができます。この場合は、どち

らでも機能しますが、その bucket プロパティに L2 Bucket が必要な場合、そのフィールドに渡すことができるのは Bucket インスタンスのみです。

こうした区別は、既存のリソースをスタックにインポートする際に非常に重要となります。そのスタックのネイティブリソースには L2 コンストラクトを作成できますが、スタック外で作成したリソースの参照が必要な場合は、その L2 コンストラクトのインターフェイスを使用しなければなりません。なぜなら、L2 コンストラクトの作成時にそのスタック内にリソースが存在しない場合、新しいリソースが作成されるからです。既存のリソースを参照するには、その L2 コンストラクトのインターフェイスに準拠するプレーンオブジェクトを使用する必要があります。

実践上、これを簡単にするために、ほとんどの L2 コンストラクトには、一連の静的メソッドが関連付けられており、こうしたメソッドによってインターフェイスが返ります。静的メソッドは通常、from という単語で始まります。これらのメソッドに渡す最初の 2 つの引数は、標準的な L2 コンストラクトに必要な scope および id の引数と同じものです。ただし、3 番目の引数には、props ではなく、インターフェイスを定義する小さいプロパティサブセット (場合によっては 1 つのプロパティのみ) を指定します。そのため、L2 コンストラクトを渡す際には、多くの場合、インターフェイスの要素のみが必要となります。これにより、可能な限りインポート済みのリソースを使用できるようになります。

```
// Example of referencing an external S3 bucket
const preExistingBucket = Bucket.fromBucketName(this, "external-bucket", "name-of-
bucket-that-already-exists");
```

ただし、インターフェイスに過度に依存してはなりません。リソースをインポートしてインターフェイスを直接使用するのは、本当に必要な場合のみにします。インターフェイスには、ヘルパーメソッドといった、L2 コンストラクトの効果を高める各種プロパティが用意されていないからです。

ヘルパーメソッド

L2 コンストラクトは単純なオブジェクトではなくプログラムクラスであるため、これによって、インスタンス化後にリソース設定を操作可能なクラスメソッドを公開できます。それがよくわかる例として、AWS Identity and Access Management (IAM) L2 Role コンストラクトが挙げられます。以下のスニペットに、2 つの方法を示します。ここでは、L2 Role コンストラクトを使用して、同じ IAM ロールを作成しています。

ヘルパーメソッドを使用しない場合:

```
const role = new Role(this, "my-iam-role", {
```

```
   assumedBy: new FederatedPrincipal('my-identity-provider.com'),
   managedPolicies: [
      ManagedPolicy.fromAwsManagedPolicyName("ReadOnlyAccess")
   ],
   inlinePolicies: {
      lambdaPolicy: new PolicyDocument({
         statements: [
            new PolicyStatement({
               effect: Effect.ALLOW,
               actions: [ 'lambda:UpdateFunctionCode' ],
               resources: [ 'arn:aws:lambda:us-east-1:123456789012:function:my-
function' ]
            })
         ]
      })
   }
});
```

ヘルパーメソッドを使用する場合:

```
const role = new Role(this, "my-iam-role", {
   assumedBy: new FederatedPrincipal('my-identity-provider.com')
});

role.addManagedPolicy(ManagedPolicy.fromAwsManagedPolicyName("ReadOnlyAccess"));
role.attachInlinePolicy(new Policy(this, "lambda-policy", {
   policyName: "lambdaPolicy",
   statements: [
      new PolicyStatement({
         effect: Effect.ALLOW,
         actions: [ 'lambda:UpdateFunctionCode' ],
         resources: [ 'arn:aws:lambda:us-east-1:123456789012:function:my-function' ]
      })
   ]
 }));
```

インスタンス化後にインスタンスメソッドを使用してリソース設定を操作できるため、L2 コンストラクトを使用すると、以前のレイヤーよりも柔軟性が大幅に向上します。L1 コンストラクトも一部のリソースメソッド (addPropertyOverride など) を継承しますが、そのリソースおよびプロパティ専用に設計されたメソッドは、レイヤー 2 で初めて利用できるようになりました。

列挙型

CloudFormation 構文では、多くの場合、リソースを適切にプロビジョニングするために、多数の詳細項目を指定しなければなりません。しかし、ほとんどのユースケースは、わずか数種類の設定を行うだけで済みます。こうした設定を一連の列挙値で表すと、必要なコード量を大幅に削減できます。

例えば、このセクションで前述した S3 バケットの L2 のコード例では、CloudFormation テンプレートの `bucketEncryption` プロパティを使用する必要があり、これによって、使用する暗号化アルゴリズムの名前などの詳細をすべて指定しました。AWS CDK では、そうした操作ではなく、`BucketEncryption` という列挙型を利用できます。これによって、バケット暗号化で最も一般的な 5 つの形式を受け取り、それぞれを単一の変数名で表現することが可能です。

次に、列挙型では対応できないエッジケースについて考えてみましょう。L2 コンストラクトの目標の 1 つは、レイヤー 1 リソースのプロビジョニングタスクを簡素化することです。そのため、あまり使用されない特定のエッジケースは、レイヤー 2 ではサポートされない可能性があります。これらのエッジケースに対応するために、AWS CDK では、addPropertyOverride メソッドを使用して、CloudFormation リソースの基盤となるプロパティを直接操作できるようになっています。プロパティのオーバーライドの詳細については、このガイドの「ベストプラクティス」セクションと、AWS CDK ドキュメントの「Abstractions and escape hatches」セクションを参照してください。

ヘルパークラス

列挙型では、特定のユースケースのリソース設定に必要なプログラムロジックを実現できないことがあります。こうした状況での AWS CDK では、多くの場合、列挙型ではなく、ヘルパークラスが利用できます。列挙型は単純なオブジェクトで、キーと値の一連のペアが使用可能な一方で、ヘルパークラスは TypeScript クラスの機能をすべて備えています。ヘルパークラスは、静的プロパティの公開によって列挙型のように動作しますが、こうしたプロパティ値は、ヘルパークラスのコンストラクタまたはヘルパーメソッド内の条件付きロジックを使用して内部的に設定できます。

前述のとおり、`BucketEncryption` 列挙型を使用すると、S3 バケットへの暗号化アルゴリズム設定に必要なコードの量を削減できますが、期間を設定する場合は、選択可能な値が多いという理由で、これと同じ戦略の効果は得られません。値ごとの列挙型の作成は、価値に見合わないかなりの労力を伴うからです。そのため、S3 バケットのデフォルト S3 Object Lock 設定には、ヘルパークラスを使用し、これを ObjectLockRetention クラスで表します。`ObjectLockRetention` は、2 つの静的メソッドで構成されます。1 つは、コンプライアンス維持を、もう 1 つはガバナンス維持を目的とするものです。どちらのメソッドも、ロック設定が必要な期間を表す Duration ヘルパークラスのインスタンスを引数として受け取ります。

もう 1 つの例として、AWS Lambda のヘルパークラスである Runtime が挙げられます。このクラスに関連付けられた静的プロパティは、一見すると、列挙型で処理できそうに思えます。しかし、内部的には、各プロパティ値は `Runtime` クラス自体のインスタンスを表しているため、クラスのコンストラクタで実行されるロジックは、列挙型では実現できません。

レイヤー 3 コンストラクト

L1 コンストラクトでは CloudFormation リソースをプログラムコードにそのまま変換し、L2 コンストラクトでは冗長な CloudFormation 構文の多くをヘルパーメソッドやカスタムロジックに置換しました。では、L3 コンストラクトではどのような処理を行うのでしょうか。それは、ユーザーの想像力によって決まります。レイヤー 3 は、どのような特定のユースケースにも合うよう作成できるのです。例えば、プロジェクトで特定のプロパティサブセットを持つリソースが必要な場合は、再利用可能な L3 コンストラクトを作成して、そうしたニーズを満たすことができます。

AWS CDK 内では、L3 コンストラクトをパターンと呼びます。パターンとは、AWS CDK の `Construct` クラスを継承する (または `Construct` クラスを継承するクラスを継承する) オブジェクトであり、レイヤー 2 を超える抽象化が行われたロジックを実行するものです。AWS CDK CLI で cdk init を実行して、新しい AWS CDK プロジェクトを開始する場合、AWS CDK アプリケーションタイプを `app`、`lib`、`sample-app` の 3 つから選択する必要があります。

```
Available templates:
* app: Template for a CDK Application
   └ cdk init app --language=[csharp|fsharp|go|java|javascript|python|typescript]
* lib: Template for a CDK Construct Library
   └ cdk init lib --language=typescript
* sample-app: Example CDK Application with some constructs
   └ cdk init sample-app --language=[csharp|fsharp|go|java|javascript|python|typescript]
```

`app` と `sample-app` はどちらも、従来の AWS CDK アプリケーションを表すもので、これによって、CloudFormation スタックを構築し AWS 環境にデプロイします。`lib` は、まったく新しい L3 コンストラクトを構築する場合に選択します。`app` と `sample-app` では、AWS CDK のサポート対象言語を任意に選択できますが、`lib` で選択できるのは TypeScript のみです。なぜなら、AWS CDK は、当初から TypeScript で開発されており、JSii というオープンソースシステムを使用して元のコードを他のサポート対象言語に翻訳するからです。AWS CDK の拡張機能を構築する場合は、`lib` を選択してプロジェクトを開始します。

`Construct` クラスを継承するクラスはすべて L3 コンストラクトとして定義できますが、レイヤー 3 の最も一般的なユースケースは、リソースインタラクション、リソース拡張、カスタムリソースです。ほとんどの L3 コンストラクトでは、AWS CDK の機能を拡張するために、これら 3 つのケースの 1 つ以上を使用します。

リソースインタラクション

ソリューションでは、通常、連携して動作する複数の AWS サービスを利用します。例えば、Amazon CloudFront ディストリビューションでは、多くの場合、そのオリジンに S3 バケットを使用し、一般的なエクスプロイトからの保護には AWS WAF を使用します。また、AWS AppSync と Amazon API Gateway では、API のデータソースに Amazon DynamoDB テーブルを使用することが多く、AWS CodePipeline のパイプラインでは、Amazon S3 をソースにし、ビルドステージに AWS CodeBuild を使用することがよくあります。こうした場合に有用なのは、単一の L3 コンストラクトを作成し、これによって、相互接続した 2 つ以上の L2 コンストラクトのプロビジョニングを処理することです。

以下に、L3 コンストラクトの例を示します。ここでは次の要素をプロビジョニングします: CloudFront ディストリビューション、その S3 オリジン、ディストリビューションの前に配置する AWS WAF、Amazon Route 53 レコード、AWS Certificate Manager (ACM) 証明書 (これにより、暗号化を備えたカスタムエンドポイントを追加する)。これらすべてが 1 つの再利用可能なコンストラクトにまとめられプロビジョニングされます。

```
// Define the properties passed to the L3 construct
export interface CloudFrontWebsiteProps {
     distributionProps: DistributionProps
     bucketProps: BucketProps
     wafProps: CfnWebAclProps
     zone: IHostedZone
}

// Define the L3 construct
export class CloudFrontWebsite extends Construct {
     public distribution: Distribution

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

         const certificate = new Certificate(this, "Certificate", {
            domainName: props.zone.zoneName,
            validation: CertificateValidation.fromDns(props.zone)
         });
         const defaultBehavior = {
```

```
            origin: new S3Origin(new Bucket(this, "bucket", props.bucketProps))
         }
         const waf = new CfnWebACL(this, "waf", props.wafProps);
         this.distribution = new Distribution(this, id, {
            ...props.distributionProps,
            defaultBehavior,
            certificate,
            domainNames: [this.domainName],
            webAclId: waf.attrArn,
          });
     }
}
```

CloudFront、Amazon S3、Route 53、ACM ではいずれも、L2 コンストラクトを使用していますが、Web ACL (Web リクエストの処理ルールを定義する) では L1 コンストラクトを使用していることに注意してください。これは、AWS CDK が進歩し続けるオープンソースパッケージであり、完成していないことと、`WebAcl` の L2 コンストラクトがまだ存在しないことによります。ただし、誰もが、新しい L2 コンストラクトを作成して AWS CDK の進歩に貢献できます。そのような理由で、AWS CDK に `WebAcl` の L2 コンストラクトが用意されるまでは、L1 コンストラクトを使用しなければなりません。`CloudFrontWebsite` L3 コンストラクトを使用して新しいウェブサイトを作成するには、次のコードを使用します。

```
const siteADotCom = new CloudFrontWebsite(stack, "siteA", siteAProps);
const siteBDotCom = new CloudFrontWebsite(stack, "siteB", siteBProps);
const siteCDotCom = new CloudFrontWebsite(stack, "siteC", siteCProps);
```

この例では、CloudFront `Distribution` の L2 コンストラクトを L3 コンストラクトのパブリックプロパティとして公開しています。このように、場合によっては、L3 プロパティの公開が必要なケースも依然として存在します。実際に、以降の「カスタムリソース」セクションでも、`Distribution` を再度取り上げます。

AWS CDK 内には、このようなリソース相互作用パターンの例が複数見られます。例えば、Amazon Elastic Container Service (Amazon ECS) の L2 コンストラクトが含まれる `aws-ecs` パッケージのほか、AWS CDK には、aws-ecs-patterns というパッケージもあります。このパッケージ内には、Amazon ECS を Application Load Balancer、Network Load Balancer、ターゲットグループと組み合わせる複数の L3 コンストラクトに加え、Amazon Elastic Compute Cloud (Amazon EC2) と AWS Fargate 向けにプリセットされた異なるバージョンが用意されています。多くのサーバーレスアプリケーションでは Amazon ECS を Fargate とのみ組み合わせて使用するため、こうした L3 コンストラクトを使用すると、開発側の時間と顧客側のコストをともに節約できます。

リソース拡張

一部のユースケースのリソースには、L2 コンストラクトでネイティブではない特定のデフォルト設定が必要です。スタックレベルであれば、アスペクトを使用してこれに対応できますが、L2 コンストラクトに新しいデフォルトを設定するもう 1 つの便利な方法は、レイヤー 2 を継承することです。どのコンストラクトも Construct クラスを継承したコンストラクトであり、L2 コンストラクトはそのクラスを継承するため、L2 コンストラクトを直接継承することでも L3 コンストラクトを作成できます。

これが特に有用なのは、顧客の特定のニーズに対応したカスタムビジネスロジックを実行するときです。AWS Lambda 関数のコードが src/lambda という単一のディレクトリにすべて保存されるリポジトリがあり、ほとんどの Lambda 関数で毎回同じランタイム名とハンドラー名が再利用されているとします。その場合は、Lambda 関数の新規設定のたびにコードパスを設定せず、新しい L3 コンストラクトを作成すると良いでしょう。

```
export class MyCompanyLambdaFunction extends Function {
      constructor(
        scope: Construct,
        id: string,
        props: Partial<FunctionProps> = {}
     ) {
         super(scope, id, {
            handler: 'index.handler',
            runtime: Runtime.NODEJS_LATEST,
            code: Code.fromAsset(`src/lambda/${props.functionName || id}`),
            ...props
     });
}
```

そうすることにより、リポジトリ内の任意の場所で L2 Function のコンストラクトを次のように置き換えることができます。

```
new MyCompanyLambdaFunction(this, "MyFunction");
new MyCompanyLambdaFunction(this, "MyOtherFunction");
new MyCompanyLambdaFunction(this, "MyThirdFunction", {
   runtime: Runtime.PYTHON_3_11
});
```

デフォルト設定を使用すると、新しい Lambda 関数を 1 行で作成でき、必要に応じてデフォルトのプロパティをオーバーライドできるように L3 コンストラクトを設定することも可能です。

既存の L2 コンストラクトに新しいデフォルト設定を追加するだけの場合に最良なのは、L2 コンストラクトを直接継承することです。他のカスタムロジックも必要な場合は、Construct クラスを継承すると良いでしょう。その理由は、コンストラクタ内で呼び出す super メソッドにあります。他のクラスを継承したクラスでは、super メソッドを使用して親クラスのコンストラクタを呼び出します。これは、コンストラクタ内で最初に実行する必要があります。つまり、渡す引数やその他のカスタムロジックの操作は、元の L2 コンストラクトが作成された後にのみ実行できます。L2 コンストラクトのインスタンス化前にこうしたカスタムロジックを実行する必要がある場合は、「リソースインタラクション」セクションで前述したパターンに従うことをお勧めします。

カスタムリソース

カスタムリソースは、CloudFormation の強力な機能であり、これによって、スタックのデプロイ中にアクティブ化した Lambda 関数からカスタムロジックを実行できます。CloudFormation で直接サポートされていないプロセスがデプロイ中に必要な場合は、カスタムリソースを使用して処理すると良いでしょう。AWS CDK には、カスタムリソースをプログラムで作成可能なクラスも用意されています。L3 コンストラクタ内でカスタムリソースを使用すると、ほぼあらゆるリソースからコンストラクトを作成できます。

Amazon CloudFront を使用する利点の 1 つは、強力なグローバルキャッシュ機能です。キャッシュを手動でリセットしてオリジンへの新しい変更をウェブサイトにすぐに反映したい場合は、CloudFront の無効化を利用できます。ただし、無効化は、CloudFront ディストリビューション上で実行されるプロセスであり、CloudFront ディストリビューションのプロパティではありません。既存のディストリビューションにいつでも作成し適用できるため、プロビジョニングおよびデプロイプロセスに、ネイティブには組み込まれていません。

このシナリオでは、ディストリビューションのオリジンを更新するたびに無効化を作成し実行するという要件が考えられます。カスタムリソースを使用すると、次のような L3 コンストラクトを作成できます。

```
export interface CloudFrontInvalidationProps {
     distribution: Distribution
     region?: string
     paths?: string[]
}

export class CloudFrontInvalidation extends Construct {
      constructor(
        scope: Construct,
        id: string,
```

```
        props: CloudFrontInvalidationProps
    ) {
        super(scope, id);
        const policy = AwsCustomResourcePolicy.fromSdkCalls({
            resources:AwsCustomResourcePolicy.ANY_RESOURCE
        });
        new AwsCustomResource(scope, `${id}Invalidation`, {
           policy,
           onUpdate: {
                service: 'CloudFront',
                action: 'createInvalidation',
                region: props.region || 'us-east-1',
                physicalResourceId:
 PhysicalResourceId.fromResponse('Invalidation.Id'),
                parameters: {
                   DistributionId: props.distribution.distributionId,
                   InvalidationBatch: {
                       Paths: {
                          Quantity: props.paths?.length || 1,
                          Items: props.paths || ['/*']
                       },
                       CallerReference: crypto.randomBytes(5).toString('hex')
                   }
                }
           }
        }
    }
}
```

先ほど CloudFrontWebsite L3 コンストラクトで作成したディストリビューションを使用すると、これを非常に簡単に実行できます。

```
new CloudFrontInvalidation(this, 'MyInvalidation', {
       distribution: siteADotCom.distribution
});
```

この L3 コンストラクトでは、AwsCustomResource という AWS CDK L3 コンストラクトを使用して、カスタムロジックを実行するカスタムリソースを作成します。Lambda コードの記述が不要という点を考えると、AwsCustomResource は、AWS SDK 呼び出しを 1 回だけ実行する必要がある場合に非常に便利です。より複雑な要件があり、独自ロジックの実装が必要な場合は、基本的な CustomResource クラスを直接使用すると良いでしょう。

AWS CDK でカスタムリソース L3 コンストラクトを使用するもう 1 つの良い例は、S3 バケットのデプロイです。カスタムリソースによってこの L3 コンストラクトのコンストラクタ内に作成された Lambda 関数は、CloudFormation では処理できない機能、つまり S3 バケットにオブジェクトを追加し更新する機能を備えるようになります。S3 バケットのデプロイ機能がないと、スタックの一部として作成した S3 バケットにコンテンツを追加できないため、非常に不便です。

AWS CDK では、膨大な CloudFormation 構文が不要になります。この基本的な `S3BucketDeployment` の例を見ると、それが非常によくわかるでしょう。

```
new BucketDeployment(this, 'BucketObjects', {
  sources: [Source.asset('./path/to/amzn-s3-demo-bucket')],
  destinationBucket: amzn-s3-demo-bucket
});
```

これと同じ目的を達成するために、CloudFormation ではどれくらいの量のコードが必要かを比較してみましょう。

```
"lambdapolicyA5E98E09": {
   "Type": "AWS::IAM::Policy",
   "Properties": {
    "PolicyDocument": {
     "Statement": [
      {
       "Action": "lambda:UpdateFunctionCode",
       "Effect": "Allow",
       "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function"
      }
     ],
     "Version": "2012-10-17"
    },
    "PolicyName": "lambdaPolicy",
    "Roles": [
     {
      "Ref": "myiamroleF09C7974"
     }
    ]
   },
   "Metadata": {
    "aws:cdk:path": "CdkScratchStack/lambda-policy/Resource"
   }
  },
  "BucketObjectsAwsCliLayer8C081206": {
```

```
    "Type": "AWS::Lambda::LayerVersion",
    "Properties": {
     "Content": {
      "S3Bucket": {
       "Fn::Sub": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}"
      },
      "S3Key": "e2277687077a2abf9ae1af1cc9565e6715e2ebb62f79ec53aa75a1af9298f642.zip"
     },
     "Description": "/opt/awscli/aws"
    },
    "Metadata": {
     "aws:cdk:path": "CdkScratchStack/BucketObjects/AwsCliLayer/Resource",
     "aws:asset:path":
"asset.e2277687077a2abf9ae1af1cc9565e6715e2ebb62f79ec53aa75a1af9298f642.zip",
     "aws:asset:is-bundled": false,
     "aws:asset:property": "Content"
    }
   },
   "BucketObjectsCustomResourceB12E6837": {
    "Type": "Custom::CDKBucketDeployment",
    "Properties": {
     "ServiceToken": {
      "Fn::GetAtt": [
       "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756C81C01536",
       "Arn"
      ]
     },
     "SourceBucketNames": [
      {
       "Fn::Sub": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}"
      }
     ],
     "SourceObjectKeys": [
      "f888a9d977f0b5bdbc04a1f8f07520ede6e00d4051b9a6a250860a1700924f26.zip"
     ],
     "DestinationBucketName": {
      "Ref": "amzn-s3-demo-bucket77F80CC0"
     },
     "Prune": true
    },
    "UpdateReplacePolicy": "Delete",
    "DeletionPolicy": "Delete",
    "Metadata": {
     "aws:cdk:path": "CdkScratchStack/BucketObjects/CustomResource/Default"
```

```
    }
   },
   "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRole89A01265": {
    "Type": "AWS::IAM::Role",
    "Properties": {
     "AssumeRolePolicyDocument": {
      "Statement": [
       {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
         "Service": "lambda.amazonaws.com"
        }
       }
      ],
      "Version": "2012-10-17"
     },
     "ManagedPolicyArns": [
      {
       "Fn::Join": [
        "",
        [
         "arn:",
         {
          "Ref": "AWS::Partition"
         },
         ":iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
        ]
       ]
      }
     ]
    },
    "Metadata": {
     "aws:cdk:path": "CdkScratchStack/
Custom::CDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756C/ServiceRole/Resource"
    }
   },

  "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRoleDefaultPolicy88902FDF":
  {
    "Type": "AWS::IAM::Policy",
    "Properties": {
     "PolicyDocument": {
      "Statement": [
```

```
      {
       "Action": [
        "s3:GetBucket*",
        "s3:GetObject*",
        "s3:List*"
       ],
       "Effect": "Allow",
       "Resource": [
        {
         "Fn::Join": [
          "",
          [
           "arn:",
           {
            "Ref": "AWS::Partition"
           },
           ":s3:::",
           {
            "Fn::Sub": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}"
           },
           "/*"
          ]
         ]
        },
        {
         "Fn::Join": [
          "",
          [
           "arn:",
           {
            "Ref": "AWS::Partition"
           },
           ":s3:::",
           {
            "Fn::Sub": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}"
           }
          ]
         ]
        }
       ]
      },
      {
       "Action": [
        "s3:Abort*",
```

```
        "s3:DeleteObject*",
        "s3:GetBucket*",
        "s3:GetObject*",
        "s3:List*",
        "s3:PutObject",
        "s3:PutObjectLegalHold",
        "s3:PutObjectRetention",
        "s3:PutObjectTagging",
        "s3:PutObjectVersionTagging"
       ],
       "Effect": "Allow",
       "Resource": [
        {
         "Fn::GetAtt": [
          "amzns3demobucket77F80CC0",
          "Arn"
         ]
        },
        {
         "Fn::Join": [
          "",
          [
           {
            "Fn::GetAtt": [
             "amzns3demobucket77F80CC0",
             "Arn"
            ]
           },
           "/*"
          ]
         ]
        }
       ]
      }
     ],
     "Version": "2012-10-17"
    },
    "PolicyName":
"CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRoleDefaultPolicy88902FDF",
    "Roles": [
     {
      "Ref":
"CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRole89A01265"
     }
```

```
    ]
   },
   "Metadata": {
    "aws:cdk:path": "CdkScratchStack/
Custom::CDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756C/ServiceRole/DefaultPolicy/
Resource"
   }
  },
  "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756C81C01536": {
   "Type": "AWS::Lambda::Function",
   "Properties": {
    "Code": {
     "S3Bucket": {
      "Fn::Sub": "cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}"
     },
     "S3Key": "9eb41a5505d37607ac419321497a4f8c21cf0ee1f9b4a6b29aa04301aea5c7fd.zip"
    },
    "Role": {
     "Fn::GetAtt": [
      "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRole89A01265",
      "Arn"
     ]
    },
    "Environment": {
     "Variables": {
      "AWS_CA_BUNDLE": "/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem"
     }
    },
    "Handler": "index.handler",
    "Layers": [
     {
      "Ref": "BucketObjectsAwsCliLayer8C081206"
     }
    ],
    "Runtime": "python3.9",
    "Timeout": 900
   },
   "DependsOn": [

 "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRoleDefaultPolicy88902FDF",
    "CustomCDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756CServiceRole89A01265"
   ],
   "Metadata": {
```

```
    "aws:cdk:path": "CdkScratchStack/
Custom::CDKBucketDeployment8693BB64968944B69AAFB0CC9EB8756C/Resource",
    "aws:asset:path":
 "asset.9eb41a5505d37607ac419321497a4f8c21cf0ee1f9b4a6b29aa04301aea5c7fd",
    "aws:asset:is-bundled": false,
    "aws:asset:property": "Code"
   }
  }
```

4 行に対し 241 行とは、かなりの差です。しかし、これは、レイヤー 3 を活用してスタックをカスタマイズする場合の可能性を示す一例にすぎません。

ベストプラクティス

L1 コンストラクト

- L1 コンストラクトを直接使用することを常に避けることはできないものの、可能な限り避けるようにすべきです。特定の L2 コンストラクトがエッジケースに対応していない場合は、L1 コンストラクトを直接使用する代わりに、以下の 2 つの選択肢を検討すると良いでしょう。
  - **`defaultChild`** にアクセスする: 必要な CloudFormation プロパティが L2 コンストラクトで利用できない場合は、`L2Construct.node.defaultChild` を使用して、基盤となる L1 コンストラクトにアクセスできます。このプロパティを介してアクセスすると、L1 コンストラクトの public プロパティを更新可能なため、L1 コンストラクトを手動で作成しなくても済みます。
  - プロパティオーバーライドを使用する: 更新対象のプロパティが public でない場合の対処を考えてみましょう。CloudFormation テンプレートで実行可能な操作を、AWS CDK でも実行できるようにする究極の方法は、addPropertyOverride を使用することです。このメソッドは、すべての L1 コンストラクトで利用できます。このメソッドに CloudFormation プロパティの名前と値を直接渡すと、CloudFormation テンプレートレベルでスタック操作が可能になります。

L2 コンストラクト

- ここで重要なのは、多くの場合 L2 コンストラクトに用意されているヘルパーメソッドの活用です。レイヤー 2 では、インスタンス化時にすべてのプロパティを渡す必要はありません。L2 ヘルパーメソッドを使用すると、リソースのプロビジョニングが飛躍的に容易になります。特に、条件付きロジックが必要な場合にそれが顕著です。Grant クラスから派生したヘルパーメソッドは、最も便利なヘルパーメソッドの 1 つと言えるでしょう。このクラスは、直接使用するものではありませんが、多くの L2 コンストラクトでは、このクラスを使用して、権限の実装を大幅に簡素化するヘルパーメソッドが提供されています。例えば、L2 Lambda 関数に L2 S3 バケットへのアクセス権限を付与する場合、`s3Bucket.grantReadWrite(lambdaFunction)` を呼び出すことができるため、新しいロールやポリシーを作成しなくても済みます。

L3 コンストラクト

- L3 コンストラクトは、スタックの再利用性とやカスタマイズ性を高めるときに非常に便利なものですが、慎重に使用することをお勧めします。どのような L3 コンストラクトのタイプが必要かや、そもそも L3 コンストラクトが必要かどうかを検討してください。
  - AWS リソースと直接やり取りしない場合は、一般的に、`Construct` クラスを継承するよりもヘルパークラスを作成する方が適切です。なぜなら、`Construct` クラスがデフォルトで実行す

るアクションの多くは、AWS リソースと直接やり取りする場合にのみ必要となるからです。こうしたアクションの実行が不要な場合は、それらを実行しない方が効率的です。

- L3 コンストラクトの新規作成が適切と判断したら、`Construct` クラスを直接継承してください。他の L2 コンストラクトの継承は、そのコンストラクトのデフォルトプロパティを更新する場合にのみ行います。他の L2 コンストラクトやカスタムロジックが必要な場合は、`Construct` を直接継承し、そのコンストラクタ内ですべてのリソースをインスタンス化します。

よくある質問

レイヤーを理解 AWS CDK せずに を使用することはできませんか?

もちろんです。ただし、最も強力なツールと同様に、 はより強力 AWS CDK になります。 AWS CDKレイヤーがどのように相互作用するかを理解することで、基本的な AWS CDK 知識だけでは不可能なスタックのデプロイを簡素化するのに役立つ新しいレベルの理解が解き放たれます。

L2 から L3 コンストラクトを作成する場合と同じ方法で L1 から L2 コンストラクトを作成することはできますか?

リソースに既に L2 コンストラクトがある場合は、そのコンストラクトを使用してレイヤー 3 でカスタマイズすることをお勧めします。なぜなら、特定のリソースに既存の L2 コンストラクトを設定する最善の方法を解き明かそうと、多くの研究が既に行われているからです。しかし、一部の L1 コンストラクトには、L2 コンストラクトがまだ存在しません。そのような場合は、 AWS CDK オープンソースライブラリのコントリビューターになることをお勧めします。ぜひ、独自の L2 コンストラクトを作成し、他のユーザーと共有してください。 AWS CDKのコントリビューターガイドラインには、それを始めるために必要なあらゆる情報が記載されています。

公式の L2 コンストラクトがまだない AWS リソースはどれですか?

L2 コンストラクトを持たない AWS リソースの数は日ごとに減少していますが、これらのリソースのいずれかの L2 コンストラクトの作成を支援したい場合は、 AWS CDK API リファレンスを参照してください。左側ペインのリソースリストで、名前の横に上付き文字「1」があるリソースには、公式の L2 コンストラクトがありません。

が AWS CDK サポートする任意の言語で L2 または L3 コンストラクトを作成できますか?

は、TypeScript、JavaScript、Python、Java、C#、Go など、いくつかのプログラミング言語 AWS CDK をサポートしています。関連する言語にコンパイルされた AWS CDK コードを使用して、個人

用 L3 コンストラクトを作成できます。ただし、 に貢献 AWS CDK したり、ネイティブ AWS CDK コンストラクトを作成したりする場合は、TypeScript を使用する必要があります。 AWS CDKでは、TypeScript が唯一のネイティブ言語だからです。他の言語 AWS CDK のバージョンは、JSii という AWS ライブラリを使用してネイティブ TypeScript コードから構築されます。

AWS CDK以外では、どこで、既存の L3 コンストラクトを入手できますか?

ここで共有する場所が多すぎますが、最も人気のあるコンストラクトの多くについては、AWS「ソリューションコンストラクト」のウェブサイトと「コンストラクトハブ」の AWS CDK 「」セクションを参照してください。 https://constructs.dev/search?q=&cdk=aws-cdk&cdkver=2&sort=downloadsDesc&offset=0

リソース

- AWS CDK API リファレンス
- AWS CloudFormation リソース仕様
- AWS CDK コンストラクトに関するドキュメント
- AWS CDK の抽象化およびエスケープハッチ
- Leverage L2 constructs to reduce the complexity of your AWS CDK application (AWS ブログ記事)
- AWS CloudFormation カスタムリソース
- AWS Solutions Constructs
- Construct Hub
- AWS CDK の例 (GitHub リポジトリ)

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
初版発行	—	2023 年 12 月 4 日

AWS 規範ガイダンスの用語集

以下は、 AWS 規範ガイダンスによって提供される戦略、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 PostgreSQL 互換エディションに移行する。
- 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の最適化レベルを導入します。例: お客様のオンプレミスの Oracle データベースを AWS クラウドの Oracle 用の Amazon Relational Database Service (Amazon RDS) に移行する。
- 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 に移行する。
- 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: お客様のオンプレミスの Oracle データベースを AWS クラウドの EC2 インスタンス上の Oracle に移行する。
- 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： Microsoft Hyper-Vアプリケーションを に移行します 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。
- 廃止 — 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制御」をご覧ください。

抽象化されたサービス

「マネージドユーザー」をご覧ください。

ACID

「原子性、一貫性、分離性、耐久性 (ACID)」をご覧ください。

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な方法ですが、さらに多くの作業が必要となります。

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

複数行に処理を行い、グループ全体を対象に単一の戻り値を計算する SQL 関数。集計関数の例としては、`SUM` や `MAX` などがあります。

AI

「人工知能」をご覧ください。

AIOps

「AI オペレーション」をご覧ください。

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。

アプリケーション制御

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 の重要な要素であり、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?」をご覧ください。

AI 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。 AWS 移行戦略での AIOps の使用方法については、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。

原子性、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制御 (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「 の 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 (AZ)

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)

組織がクラウドへの移行を成功させるための効率的で効果的な計画を立て AWS るための、 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整理しています。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 のホワイトペーパー 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

個人や組織に混乱や損害を与えることを目的としたボット。

BCP

「ビジネス継続性計画 (BCP)」をご覧ください。

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 の動作グラフを使用すると、失敗したログオンの試行、不審な API 呼び出し、その他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「動作グラフのデータ」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。「エンディアン性」もご覧ください。

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

それぞれが独立しているが、同一の環境を 2 つ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を 1 つの環境 (ブルー) で実行し、新しいアプリケーションバージョンを別の環境 (グリーン) 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、個人や組織を混乱させたり、損害を与えたりすることを意図したものもあります。

ボットネット

マルウェアに感染しており、ボットハーダーまたはボットオペレーターと呼ばれる単一の当事者によって制御されているボットのネットワーク。ボットネットは、ボットとその影響力を拡大する仕組みとして、非常によく知られています。

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、「ブランチの概要」(GitHub ドキュメント) 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、 AWS Well-Architected ガイダンスの「ブレークグラス手順の実装」インジケータ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、AWSでのコンテナ化されたマイクロサービスの実行ホワイトペーパーの「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。

ビジネス継続性計画 (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

カナリアデプロイ

エンドユーザーへのバージョンリリースを、時間をかけて段階的に行うこと。確信が持てたら新規バージョンをデプロイして、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、ターゲットシステムでの変更を監査またはレプリケートして同期を維持するなど、さまざまな目的に使用できます。

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こと。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 を受信する前のローカルでのデータの暗号化。

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の CCoE 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

組織が、 AWS クラウドへの移行時に通常実行する 4 つの段階。

- 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
- 基礎固め — お客様のクラウドの導入を拡大するための基礎的な投資 (ランディングゾーンの作成、CCoE の定義、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 によって定義されました。移行戦略との関連性については、 AWS 「移行準備ガイド」を参照してください。

CMDB

「構成管理データベース (CMDB)」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、GitHub や Bitbucket Cloud があります。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といった、ビジュアル形式の情報を分析および抽出する AI の分野。例えば、Amazon SageMaker AI では、CV 用の画像処理アルゴリズムを利用できます。

設定ドリフト

ワークロードにおいて、設定が想定した状態から変化すること。これによって、ワークロードが非準拠になる可能性がありますが、この状態は、徐々に生じ、意図的なものではありません。

構成管理データベース (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、CMDB のデータは、移行のポートフォリオの検出と分析の段階で使用します。

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 および リージョンの単一のエンティティとしてデプロイすることも、組織全体にデプロイすることもできます。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、本番の各ステージを自動化するプロセス。CI/CD は一般的にパイプラインと呼ばれます。プロセスの自動化、生産性の向上、コード品質の向上、配信の加速化を可能にします。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

非一元的で分散型のデータ所有権を持つとともに、一元的な管理およびガバナンスを行え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。信頼された ID のみが、期待されるネットワークから信頼されたリソースにアクセスできるようにします。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、一般的に、大量の履歴データが含まれており、多くの場合、それらは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ます。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。

深層学習

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。

多層防御

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 に採用するときは AWS、リソースの保護に役立つように、 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ます。たとえば、多層防御アプローチでは、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。

トラブルシューティング

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、「 AWSでのセキュリティコントロールの実装」の「検出的コントロール」を参照してください。

開発バリュー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スピードと品質に悪影響を及ぼす制約を特定し、優先順位を付けるために使用されるプロセス。DVSM は、もともとリーンマニュファクチャリング・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・プロセスを拡張したものです。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

スタースキーマにおいて、ファクトテーブルの定量データに関するデータ属性が含まれる小さいテーブル。ディメンションテーブルの属性は、通常、テキストフィールド、またはテキストのように扱える個別の数値で示されます。これらの属性は、一般的に、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)

ディザスタ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の詳細については、「コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET (ASMX) ウェブサービスを段階的にモダナイズ」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を追跡します。たとえば、 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するランディングゾーンの変更を検出したりできます。

DVSM

「開発バリューストリームマッピング」を参照してください。

E

EDA

「探索的データ分析」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、応答時間を改善できます。

電子データ交換 (EDI)

組織間で行う、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処理。

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) 内でホストして、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 VPC エンドポイントを作成することで、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を作成する」を参照してください。

エンタープライズリソースプランニング (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、ID 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。お客様は、データを収集または集計してから、パターンの検出、異常の検出、および前提条件のチェックのための初期調査を実行します。EDA は、統計の概要を計算し、データの可視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

スタースキーマの中央にあるテーブル。ビジネスオペレーションに関する定量的データが保存されます。一般的に、ファクトテーブルは、2 種類の列で構成されます。1 つは測定値が含まれる列、もう 1 つはディメンションテーブルへの外部キーが含まれる列です。

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、頻繁かつ段階的にテストを行う哲学であり、アジャイルアプローチでは、この考え方がきわめて重要です。

障害分離境界

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高めるのに役立つ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。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 Deskonations (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で表されます。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

LLM に、タスクと望ましい出力を示す例を少数提示した後に、類似のタスクを実行させること。この手法は、プロンプトに記述された例 (ショット) からモデルが学習する「インコンテキスト学

習」の一種です。数ショットプロンプトは、特定のフォーマット、推論、専門知識が必要なタスクに効果的です。「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制御」を参照してください。

きめ細かなアクセス制御 (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

変更データのキャプチャによる継続的なデータ複製を利用して、段階的なアプローチではなく、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)

大規模な深層学習ニューラルネットワークであり、一般化およびラベル付けされていないデータからなる大規模データセットでトレーニングされています。FM により、言語理解、テキストおよび画像生成、自然言語での会話といった、一般的な各種タスクを実行できます。詳細については、「基盤モデルとは何ですか?」を参照してください。

G

生成 AI

AI モデルのサブセット。大量のデータでトレーニングされており、シンプルなテキストプロンプトを使用して、画像、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ます。詳細については、「生成 AI とは何ですか?」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)

特定の国のユーザーがコンテンツ配信にアクセスできないようにするための、Amazon CloudFront のオプション。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 ワークフローは古いと見なされている方法であり、トランクベースのワークフローは推奨されている新しい方法です。

ゴールデンイメージ

システムまたはソフトウェアのスナップショットであり、システムまたはソフトウェアの新規インスタンスをデプロイするテンプレートとして使用されます。製造の例で言えば、ゴールデンイメージを使用すると、複数のデバイスにソフトウェアをプロビジョニングして、デバイス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 (OU) 全般のリソース、ポリシー、コンプライアンスを管理するのに役立つ概略的なルール。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、、 AWS Security Hub CSPM、Amazon 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。

高可用性 (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用データセットから保留される、ラベル付き履歴データの一部。ホールドアウトデータを使用すると、モデル予測をホールドアウトデータと比較して、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

お客様の出典データベースを、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) に移行する。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。緊急性が高いため、通常の DevOps のリリースワークフローからは外れた形で実施されます。

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 日間の平均的な CPU およびメモリ使用率が 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

「インダストリアル IoT」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

既存インフラストラクチャの更新、パッチ適用、変更などを行わずに、本番環境ワークロードに使用する新規インフラストラクチャをデプロイするモデル。本質的に、イミュータブルインフラストラクチャは、ミュータブルインフラストラクチャよりも一貫性、信頼性、予測性に優れています。詳細については、 AWS Well-Architected フレームワークにある「イミュータブルインフラストラクチャを使用してデプロイする」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 (受信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーションの外部からネットワーク接続を受け入れ、検査し、ルーティングする VPC。AWS Security Reference Architecture 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のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

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 氏が提唱した用語で、接続、リアルタイムデータ、オートメーション、分析、AI/ML の進歩による、ビジネスプロセスのモダナイズを意味します。

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。

インダストリアル 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、「インダストリアル IoT (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。

インスペクション VPC

AWS マルチアカウントアーキテクチャでは、VPC (同一または異なる 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を管理する一元化された VPCs。AWS Security Reference Architecture 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のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

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

「IoT」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 ITSM の基盤を提供します。

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

ラベルベースアクセス制御 (LBAC)

強制アクセス制御 (MAC) の実装で、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ます。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)

大量のデータで事前トレーニングされた深層学習 AI モデル。LLM では、質問への回答、ドキュメントの要約、他言語へのテキスト翻訳、文を完成させるなど、さまざまなタスクを実行できます。詳細については、「大規模言語モデル (LLM) とは何ですか?」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

「ラベルベースアクセス制御」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 Rs」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。「エンディアン性」もご覧ください。

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やプライバシーを侵害するように設計されたソフトウェア。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、不正アクセスを招く可能性があります。マルウェアの例には、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 AWS を運用し、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。マネージドサービスの例として、Amazon Simple Storage Service (Amazon S3) と Amazon DynamoDB が挙げられます。このサービスは、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、制御するソフトウェアシステムであり、工場では、これによって、原材料から製品を完成させます。

MAP

「Migration Acceleration Program」を参照してください。

メカニズム

ツールを作成してその導入を推進し、導入結果を調べて調整を行うための包括的なプロセス。メカニズムとは、運用中にそれ自体を強化し改善するサイクルを意味します。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。

Message Queuing Telemetry Transport (MQTT)

発行/サブスクライブのパターンに基づく、軽量のマシンツーマシン (M2M) 通信プロトコルであり、リソースに限りのある IoT デバイスに使用されます。

マイクロサービス

明確に定義された API を介して通信し、通常は小規模な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です。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 API 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。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)

組織がクラウドに移行するための強力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、組織的な方法でレガシー移行を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および高速化する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および所有者、移行エンジニア、デベロッパー、およびスプリントで作業する 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としては、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 を使用して Amazon EC2 への移行をリホストします。

Migration Portfolio Assessment (MPA)

オンラインツール。これによって、 AWS クラウドに移行するビジネスケースの検証に必要な情報を得られます。MPA は、詳細なポートフォリオ評価 (サーバーの適切なサイジング、価格設定、TCO 比較、移行コスト分析) および移行プラン (アプリケーションデータの分析とデータ収集、アプリケーションのグループ化、移行の優先順位付け、および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)

AWS CAF 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。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。MRA は、AWS 移行戦略の第一段階です。

移行戦略

ワークロードを AWS クラウドに移行するために使用するアプローチ。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、「AWS クラウドでのアプリケーションのモダナイズ戦略」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、「AWS クラウドで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評価する」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

「Migration Portfolio Assessment」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。

ミュータブルなインフラストラクチャ

本番ワークロードに使用する既存のインフラストラクチャを更新および変更するためのモデル。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の使用をベストプラクティスとして推奨しています。

O

OAC

「オリジンアクセス制御」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

「オペレーション統合」を参照してください。

Ola

「オペレーショナルレベルアグリーメント」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

「Open Process Communications - Unified Architecture」を参照してください。

Open Process Communications - Unified Architecture (OPC-UA)

産業オートメーション用のマシンツーマシン (M2M) 通信プロトコル。OPC-UA により、相互運用の際に、データ暗号化、認証、認可の各スキームを標準化できます。

オペレーショナルレベルアグリーメント (OLA)

サービスレベルアグリーメント (SLA) をサポートするために、どの機能的 IT グループが互いに提供することを約束するかを明確にする契約。

運用準備状況レビュー (ORR)

質問と関連するベストプラクティスのチェックリスト。インシデントや起こり得る障害を理解、評価、防止したり、その範囲を縮小したりする際に役立ちます。詳細については、 AWS Well-Architected フレームワークの「Operational Readiness Reviews (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させるハードウェアおよびソフトウェアシステム。製造分野では、Industry 4.0 への変革を進める上で、OT と情報技術 (IT) システムの統合に焦点が当てられています。

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、CloudTrail ドキュメントの「組織の証跡の作成」を参照してください。

組織変更管理 (OCM)

人材、文化、リーダーシップの観点から、主要な破壊的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、文化や組織の変化を推進することで、組織が新しいシステムと戦略の準備と移行するのを支援します。 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により、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。

オリジンアクセス制御 (OAC)

Amazon Simple Storage Service (Amazon S3) コンテンツを保護するための、CloudFront のアクセス制限の強化オプション。OAC は AWS リージョン、すべての S3 バケット、 AWS KMS (SSE-KMS) によるサーバー側の暗号化、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)

CloudFront の、Amazon S3 コンテンツを保護するためのアクセス制限オプション。OAI を使用すると、CloudFront が、Amazon S3 に認証可能なプリンシパルを作成します。認証されたプリンシパルは、S3 バケット内のコンテンツに、特定の CloudFront ディストリビューションを介してのみアクセスできます。OAC も併せて参照してください。OAC では、より詳細な、強化されたアクセス制御が可能です。

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 (送信) 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を処理する VPC。AWS Security Reference Architecture 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のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

P

アクセス許可の境界

ユーザーまたはロールが使用できるアクセス許可の上限を設定する、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。PII の例には、氏名、住所、連絡先情報などがあります。

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

「プログラマブル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

次の操作を可能にするオブジェクト: アクセス許可を定義する (ID ベースのポリシーを参照)。アクセス条件を指定する (リソースベースのポリシーを参照)。 AWS Organizations の組織における全アカウントにアクセス許可の上限を定義する (サービスコントロールポリシーを参照)。

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。

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、「移行の準備状況の評価」を参照してください。

述語

`true` または `false` を返すためのクエリ条件。一般的に、`WHERE` 句に記述されます。

述語プッシュダウン

データベースクエリを最適化する手法。これによって、転送前にクエリ内のデータをフィルタリングします。この手法を取ると、リレーショナルデータベースから取得し処理する必要のあるデータの量が減少するため、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、「AWSでのセキュリティコントロールの実装」の「予防的コントロール」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、、IAM AWS アカウントロール、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「ロールに関する用語と概念」にある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

開発プロセス全体を通してプライバシーが考慮されているシステムエンジニアリングのアプローチ。

プライベートホストゾーン

1 つ以上の VPC 内のドメインとそのサブドメインへの DNS クエリに対し、Amazon Route 53 が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。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防止を目的としたセキュリティコントロール。このコントロールにより、プロビジョニング前にリソースをスキャンします。コントロールに準拠していないリソースは、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)

製品の設計、開発、発売から、成長、成熟、衰退、廃棄に至る、製品のライフサイクル全体を通してデータとプロセスを管理すること。

本番環境

「環境」を参照してください。

プログラマブルロジックコントローラー (PLC)

製造分野で使用される、信頼性と適応性に優れたコンピュータであり、これによって、マシンをモニタリングするとともに、製造プロセスを自動化します。

プロンプトチェイニング

1 つの LLM プロンプトによる出力を次のプロンプトの入力に使用して、より良いレスポンスを生成します。この手法を使用すると、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改良または拡張したりできます。これによって、モデルのレスポンスの精度と関連性が向上し、粒度の高いパーソナライズされた結果を得られます。

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。

発行/サブスクライブ (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。これにより、スケーラビリティと応答性を向上させます。例えば、マイクロサービスベースの MES の場合、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可能なチャネルにイベントメッセージを発行できます。このシステムでは、発行サービスの変更なしに、新規マイクロサービスを追加できます。

Q

クエリプラン

手順などの一連のステップであり、SQL リレーショナルデータベースシステムのデータにアクセスするために使用されます。

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

「実行責任者、説明責任者、協業先、報告先 (RACI)」を参照してください。

RAG

「検索拡張生成」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

「実行責任者、説明責任者、協業先、報告先 (RACI)」を参照してください。

RCAC

「行と列のアクセス制御」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。

リアーキテクト

「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 から分離され、独立しています。詳細については、「アカウントが使用できる AWS リージョン を指定する」を参照してください。

リグレッション

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 Rs」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 Rs」を参照してください。

リプラットフォーム

「7 Rs」を参照してください。

再購入

「7 Rs」を参照してください。

回復性

中断に抵抗または中断から回復するアプリケーションの機能。 AWS クラウドでの回復力を計画する際には、一般的に、高可用性とディザスタリカバリが考慮されます。詳細については、「AWS クラウド の耐障害性」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。

実行責任者、説明責任者、協業先、報告先 (RACI) に基づくマトリックス

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。サポートが含まれる場合は RASCI マトリックスと呼ばれ、含まれない場合は 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、「 AWSでのセキュリティコントロールの実装」の「レスポンシブコントロール」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。

検索拡張生成 (RAG)

生成 AI の技術。これにより、LLM では、レスポンスの生成前に、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が参照されます。例えば、RAG モデルによって、組織のナレッジベースまたはカスタムデータのセマンティック検索を実行できる場合があります。細については、「RAG (検索拡張生成) とは何ですか?」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情報を更新して、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。

行と列のアクセス制御 (RCAC)

アクセスルールが定義された、基本的で柔軟な SQL 表現の使用。RCAC は行権限と列マスクで構成されています。

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 (IdP) が使用しているオープンスタンダード。この機能を使用すると、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、ユーザーは組織内のすべてのユーザーを IAM で作成しなくても、 AWS マネジメントコンソール にログインしたり AWS 、API オペレーションを呼び出すことができます。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「SAML 2.0 ベースのフェデレーションについて」を参照してください。

SCADA

「監視制御とデータ取得」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 AWS Secrets Managerパスワードやユーザー認証情報などの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。シークレット値には、バイナリ、1 つの文字列、複数の文字列を指定できます。詳細については、Secrets Manager ドキュメントの「Secrets Manager シークレットの概要」を参照してください。

セキュリティバイデザイン

開発プロセス全体を通してセキュリティが考慮されているシステムエンジニアリングのアプローチ。

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、主に 4 つの種類があります。4 つとは、予防、検出、レスポンシブ、プロアクティブです。

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。

Security Information and Event Management (SIEM) システム

セキュリティ情報管理 (SIM) とセキュリティイベント管理 (SEM) の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、アラートを発信します。

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への自動レスポンスまたは自動修復を目的として、事前定義および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、認証情報の更新などがあります。

サーバー側の暗号化

送信先で、それ AWS のサービス を受け取る 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)

AWS Organizations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管理するポリシー。SCP は、管理者がユーザーまたはロールに委任するアクションに、ガードレールを定義したり、アクションの制限を設定したりします。SCP は、許可リストまたは拒否リストとして、許可または禁止するサービスやアクションを指定する際に使用できます。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の URL 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、スループットといった、サービスパフォーマンス面の指標。

サービスレベル目標 (SLO)

サービスレベルインジケータによって測定され、サービスの状態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、 はクラウド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

「Security Information and Event Management システム」を参照してください。

単一障害点 (SPOF)

特定のアプリケーションを構成する単一の重要なコンポーネントで発生し、システム稼働に支障をきたす可能性のある障害。

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。

スプリットアンドシードモデル

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、「AWS クラウドで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」を参照してください。

SPOF

「単一障害点」を参照してください。

スタースキーマ

データベースの編成構造を意味し、1 つの大きいファクトテーブルにトランザクションデータまたは測定データが保存され、1 つ以上の小さいディメンションテーブルにデータ属性が保存されます。この構造は、データウェアハウスやビジネスインテリジェンスを用途とするよう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。このパターンの適用方法の例については、「コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET (ASMX) ウェブサービスを段階的にモダナイズ」を参照してください。

サブネット

VPC 内の IP アドレスの範囲。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。

監視制御とデータ取得 (SCADA)

製造分野において、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本番運用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、起こり得る問題を検出したり、パフォーマンスをモニタリングしたりすることで、システムをテストします。Amazon CloudWatch Synthetics を使用すると、こうしたテストを作成できます。

システムプロンプト

コンテキスト、指示、ガイドラインなどを提示して、LLM に動作を指示する手法。システムプロンプトは、コンテキストを設定して、ユーザーとやり取りするルールを確立するのに有用です。

T

タグ

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

VPC とオンプレミスネットワークを相互接続するために使用できる、ネットワークの中継ハブ。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で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「Using AWS Organizations with other AWS services 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

2 枚のピザを分け合えることができるくらい小さな 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。

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できる、2 つの VPC 間の接続。詳細については、Amazon VPC ドキュメントの「VPC ピア機能とは」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

現在のレコードに何らかの形で関連している行のグループに計算を実行する SQL 関数。ウィンドウ関数は、移動平均を計算したり、現在の行の相対位置に基づいて他の行の値にアクセスするといった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

「Write-Once-Read-Many」を参照してください。

WQF

「AWS ワークロード資格フレームワーク」を参照してください

Write-Once-Read-Many (WORM)

データを 1 回のみ書き込むことで、データの削除や変更を防ぐストレージモデル。承認済みユーザーは、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

ゼロデイ脆弱性を悪用した攻撃 (一般的にマルウェアによる)。

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

LLM にタスク実行の手順は提示するが、実行のガイドとして役立つ例 (ショット) は提示しない方法。LLM 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 CPU およびメモリ使用率が 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。