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

Steven Guggenheimer、Amazon Web Services (AWS)

2023 年 12 月 (ドキュメント履歴)

の背後にある主な概念の 1 つは AWS Cloud Development Kit (AWS CDK) 、コールドデーにウォームを維持するという概念とよく似ています。この概念はレイヤーと呼ばれます。コールドデーには、着物やジャケットを着ます。また、気温によってはさらに大きなジャケットを着ることもあります。次に、内部に入り、ヒータが点滅している場合は、過熱しないように、一方または両方のジャケットレイヤーを切り離すことができます。 AWS CDK は、レイヤー化を使用して、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提供します。レイヤー化により、Infrastructure as Code (IAC) スタックをデプロイするときに、コードの書き込みが多すぎたり、リソースプロパティへのアクセスが少なすぎたりする必要がなくなります。

を使用しない場合は AWS CDK、AWS CloudFormationテンプレートを手動で記述する必要があります。つまり、通常は必要な数よりもはるかに多くのコードを強制的に記述するレイヤーを 1 つだけ活用します。一方、 AWS CDK が CloudFormation で通常書き込む必要がないものをすべて抽象化すると、エッジケースを処理できなくなります。

この問題に対処するために、 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 AWS CDK 分割します。

- レイヤー 1 – CloudFormation レイヤー: CloudFormation リソースと AWS CDK リソースがほぼ同じである最も基本的なレイヤー。
- 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、内部の定型 CloudFormation 構文の多くを効率化するレイヤー。このレイヤーは、 の大部分を構成します AWS CDK。
- レイヤー 3 – パターンレイヤー: レイヤー 1 と 2 が提供する構成要素を使用して、特定のユースケースに合わせてコードをカスタマイズできる最も抽象化されたレイヤー。

各レイヤーの各項目は、 と呼ばれる特別な AWS CDK クラスのインスタンスです`Construct`。AWS ドキュメントによると、コンストラクトは AWS CDK 「アプリの基本的な構成要素」です。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 AWS CloudFormation に必要なすべてをカプセル化します。」 これらのレイヤー内のコンストラクトは、それらが属するレイヤーに応じて L1、L2、および L3 コンストラクトと呼ばれます。このガイドでは、各 AWS CDK レイヤーについてツアーを行い、それらの用途と重要性を確認します。

このガイドは、 AWS CDK 作業を行うコアコンセプトをより深く掘り下げることに関心のある技術マネージャー、リーダー、開発者を対象としています。 AWS CDK は人気のあるツールですが、チームが提供すべきものの大部分を見逃すのは非常に一般的です。このガイドで説明されている概念を理解し始めると、まったく新しい可能性を引き出し、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ます。

このガイドでは、次の操作を行います。

- レイヤー 1 コンストラクト
- レイヤー 2 コンストラクト
- レイヤー 3 コンストラクト
- ベストプラクティス
- よくある質問
- リソース

レイヤー 1 コンストラクト

L1 コンストラクトは の構成要素 AWS CDK であり、プレフィックス によって他のコンストラクトと簡単に区別できますCfn。例えば、 の Amazon DynamoDB パッケージには、L2 Tableコンストラクトである コンストラクト AWS CDK が含まれています。対応する L1 コンストラクトは と呼ばれCfnTable、CloudFormation DynamoDB を直接表しますTable。 AWS CDK アプリケーションは通常、L1 コンストラクトを直接使用することは決してありませんが、この最初のレイヤーにアクセス AWS CDK せずに を使用することは不可能です。ただし、ほとんどの場合、デベロッパーが L1 コンストラクトの使用に慣れている L2 コンストラクトと 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ます。したがって、L1 コンストラクトは CloudFormation と の間のブリッジと考えることができます AWS CDK。

の唯一の目的 AWS CDK は、標準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です。cdk synth CLI コマンドを実行し、結果の CloudFormation テンプレートが生成されると、 AWS CDKジョブは完了です。cdk deploy コマンドは便利なように用意されていますが、そのコマンドの実行時に実行していることは CloudFormation 内で完全に行われます。CloudFormation が理解する形式に AWS CDK コードを変換するパズルの部分は L1 コンストラクトです。

L1 コンストラクトの AWS CDK–CloudFormation ライフサイクル

L1 コンストラクトを作成および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 ビルドプロセスは、CloudFormation の仕様を L1 コンストラクトの形式でプログラムコードに変換します。
2. 開発者は、 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. デベロッパーは cdk synth コマンドを実行して、プログラムによるコードを CloudFormation 仕様 (テンプレート) で指定された形式に変換します。
4. デベロッパーは cdk deploy コマンドを実行して、これらのテンプレート内の CloudFormation スタックを AWS アカウント環境にデプロイします。

では、少し演習してみましょう。GitHub のAWS CDK オープンソースリポジトリに移動し、ランダムな AWS サービスを選択し、そのサービスの AWS CDK パッケージ (フォルダ packages、aws-cdk-lib、aws-<servicename>、) に移動しますlib。この例では Amazon S3 を選択しますが、これはどのサービスでも機能します。そのパッケージのメイン index.ts ファイルを見ると、次の行が表示されます。

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

ただし、対応するディレクトリのどこにもs3.generatedファイルが表示されません。これは、AWS CDK ビルドプロセス中に L1 コンストラクトが CloudFormation リソース仕様から自動生成されるためです。したがって、パッケージs3.generatedの AWS CDK ビルドコマンドを実行した後にのみ、パッケージに が表示されます。

AWS CloudFormation リソース仕様

AWS CloudFormation リソース仕様は、 の Infrastructure as Code (IAC) を定義 AWS し、CloudFormation テンプレート内のコードを AWS アカウントのリソースに変換する方法を決定します。この仕様では、リージョンレベルで JSON 形式の AWS リソースを定義します。各リソースには、 の形式に従う一意のリソースタイプ名が割り当てられますprovider::service::resource。例えば、Amazon S3 バケットのリソースタイプ名は でAWS::S3::Bucket、Amazon S3 アクセスポイントのリソースタイプ名は ですAWS::S3::AccessPoint。これらのリソースタイプは、リソース AWS CloudFormation 仕様で定義された構文を使用して CloudFormation テンプレートでレンダリングできます。 AWS CDK ビルドプロセスが実行されると、各リソースタイプも L1 コンストラクトになります。

したがって、各 L1 コンストラクトは、対応する CloudFormation リソースのプログラムによるミラーイメージです。CloudFormation テンプレートに適用するすべてのプロパティは、L1 コンストラクトをインスタンス化するときに使用でき、対応する L1 コンストラクトをインスタンス化するときに、必要なすべての CloudFormation プロパティも引数として必要です。次の表は、CloudFormation テンプレートで表される S3 バケットと、an AWS CDK L1 コンストラクトとして定義された同じ 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",
      "BucketEncryption": {
        "ServerSideEncryptionConfig
uration": [
          {
```

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"
```

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

```
        }
      }
    ]
  },
  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 リソースのコード内の正確なマニフェストです。ショートカットや単純化はないため、書き込む必要がある定型テキストの量はほぼ同じです。ただし、 を使用すること AWS CDK の大きな利点の 1 つは、その共通 CloudFormation 構文の多くを排除できることです。では、どのように行われますか？ そこで L2 コンストラクトが登場します。

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は、主に TypeScript プログラミング言語を使用して記述され、多数のパッケージとモジュールで構成されています。と呼ばれるメインパッケージライブラリはaws-cdk-lib、 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ますが、必ずしもそうとは限りません。前述のように、L1 コンストラクトはビルドプロセス中に自動的に生成されるため、リポジトリ内を見るときに表示されるすべてのコードは何ですか？ これらは L2 コンストラクトの抽象化である L12 コンストラクトです。

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と、より多くの機能を追加するヘルパークラスのコレクションも含まれていますが、それらの項目はすべて L2 コンストラクトを提供します。すべての L2 コンストラクトは、インスタンス化時にコンストラクタで対応する 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 コンストラクトは、デフォルトのプロパティ、便利なメソッド、およびその他の構文粥を受け取り、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繰り返しと詳細度の大部分が取り除かれます。

すべての L2 コンストラクトは、対応する L1 コンストラクトを内部に構築します。ただし、L2 コンストラクトは実際には L1 コンストラクトを拡張しません。L1 コンストラクトと L2 コンストラクトの両方が、コンストラクトという特別なクラスを継承します。バージョン 1 の AWS CDK 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 では別のスタンドアロンパッケージです。これは、Cloud Development Kit for Terraform (CDKTF) などの他のパッケージが依存関係として含めることができるためです。クラスを継承するConstructクラスは、L1, L2、または L3 コンストラクトです。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拡張しCfnResource、L1 コンストラクトは というクラスを拡張します。

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	→ クラスコンストラクト
→→ 抽象クラス CfnRefElement	

→ → → 抽象クラス CfnElement

→ → → → クラスコンストラクト

L1 コンストラクトと L2 コンストラクトの両方が Construct クラスを継承する場合、L2 コンストラクトが L1 を拡張しないのはなぜですか？ クラスConstructとレイヤー 1 の間のクラスは、CloudFormation リソースのミラーイメージとして L1 コンストラクトを所定の位置にロックします。これには、 のような抽象メソッド (ダウンストリームクラスに含める必要があるメソッド) が含まれており_toCloudFormation、コンストラクトが CloudFormation 構文を直接出力するように強制します。L2 コンストラクトはこれらのクラスをスキップし、Constructクラスを直接拡張します。これにより、コンストラクタ内で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ます。

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された同じ S3 バケットをside-by-side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示され、L1 コンストラクトは CloudFormation コンストラクトと比較して 3 行または 4 行のみを保存します。次に、L1 コンストラクトを同じ S3 バケットの L2 コンストラクトと比較します。

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: [
    {
      id: "myConfig"
    }
```

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
});
```

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

ご覧のとおり、L2 コンストラクトは L1 コンストラクトのサイズの半分未満です。L2 コンストラクトは、この統合を達成するために多くの手法を使用します。これらの手法の中には、単一の L2 コンストラクトに適用されるものもありますが、再利用できるように独自のクラスに分離されるように、複数のコンストラクト間で再利用できるものもあります。L2 コンストラクトは、以下のセクションで説明するように、いくつかの方法で CloudFormation 構文を統合します。

デフォルトのプロパティ

リソースをプロビジョニングするためのコードを統合する最も簡単な方法は、最も一般的なプロパティ設定をデフォルトに変換することです。 AWS CDK は強力なプログラミング言語にアクセスでき、CloudFormation はアクセスできないため、これらのデフォルトは多くの場合、本質的に条件付きです。コンストラクトに渡される他のプロパティの値からこれらの設定を推測できるため、CloudFormation 設定の複数の行を AWS CDK コードから削除できる場合があります。

構造体、タイプ、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、TypeScript でネイティブに記述されるため、言語の型システムを使用して L2 コンストラクトを構成する型を定義します。そのタイプシステムに深く掘り下げることは、このガイドの範囲外です。詳細については[TypeScript のド

キュメントを参照してください。要約すると、TypeScript は特定の変数が保持するデータの種類をtype記述します。これは、 などの基本データでもstring、 などのより複雑なデータでもかまいませんobject。TypeScript interfaceは TypeScript オブジェクトタイプを表す別の方法であり、structはインターフェイスの別の名前です。

TypeScript は struct 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、構造体は実際にはコード内の別の TypeScript インターフェイスにすぎないことがわかります。API リファレンスでは、特定のインターフェイスもインターフェイスとして参照されます。構造体とインターフェイスが同じ場合、ドキュメントで AWS CDK それらが区別されるのはなぜですか？

が構造体と AWS CDK 呼ぶのは、L2 コンストラクトで使用されるオブジェクトを表すインターフェイスです。これには、インスタンス化中に L2 コンストラクトに渡されるプロパティ引数のオブジェクトタイプが含まれます。たとえば、S3 バケットコンストラクトBucketPropsの場合は 、DynamoDB テーブルコンストラクトTablePropsの場合は 、 内で使用されるその他の TypeScript インターフェイスなどです AWS CDK。つまり、 内の TypeScript インターフェイス AWS CDK で、その名前に という文字のプレフィックスが付いていない場合I、 はそれを 構造体と AWS CDK 呼びます。

逆に、 AWS CDK は インターフェイスという用語を使用して、プレーンオブジェクトが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必要がある基本要素を表します。つまり、インターフェイスは 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。すべての AWS CDK インターフェイス名は、 という文字のプレフィックスが付いた既存のコンストラクトまたはヘルパークラスの名前ですI。すべての L2 コンストラクトは Construct クラスを拡張しますが、対応するインターフェイスも実装します。したがって、L2 コンストラクトは IBucketインターフェイスBucketを実装します。

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、その逆ではありません。これは、構造体を調べて、どのデータ型が必要かを確認するときに重要です。構造体に というプロパティがありbucket、データ型 が必要な場合はIBucket、IBucketインターフェイスにリストされているプロパティを含むオブジェクトまたは L2 のインスタンスを渡すことができますBucket。どちらかが機能します。ただし、そのbucketプロパティが L2 を呼び出す場合Bucket、そのフィールドのBucketインスタンスのみを渡すことができます。

この区別は、既存のリソースをスタックにインポートするときに非常に重要になります。スタックにネイティブな任意のリソースの L2 コンストラクトを作成できますが、スタックの外部で作成されたリソースを参照する必要がある場合は、その L2 コンストラクトのインターフェイスを使用する必要

があります。これは、L2 コンストラクトを作成すると、そのスタック内に新しいリソースが存在しない場合に新しいリソースが作成されるためです。既存のリソースへの参照は、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である必要があります。

これを実際に簡単にするために、ほとんどの L2 コンストラクトには、その L2 コンストラクトのインターフェイスを返す一連の静的メソッドが関連付けられています。これらの静的メソッドは通常、という単語で始まりますfrom。これらのメソッドに渡される最初の 2 つの引数は、標準の L2 コンストラクトに必要な引id数scopeと同じです。ただし、3 番目の引数は propsではなく、インターフェイスを定義するプロパティの小さなサブセット (または 1 つのプロパティのみ) です。このため、L2 コンストラクトを渡す場合、ほとんどの場合、インターフェイスの要素のみが必要です。これは、インポートされたリソースを可能な限り使用できるようにするためです。

```
// 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大きく依存しないでください。L2 コンストラクトをそれほど強力にするヘルパーメソッドなどのプロパティの多くがインターフェイスによって提供されていないため、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必要があるのは絶対的な場合のみです。

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によるクラスであるため、インスタンス化の実行後にリソース設定を操作できるクラスメソッドを公開できます。その良い例は、 AWS Identity and Access Management (IAM) L2 ロール構造です。次のスニペットは、L2 コンストラクトを使用して同じ IAM ロールを作成する 2 つの方法を示しています。 Role

ヘルパーメソッドがない場合：

```
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 は前のレイヤーよりもはるかに柔軟性が高まります。L1 コンストラクトもいくつかのリソースメソッド (などaddPropertyOverride) を継承しますが、レイヤー 2 がリソースとそのプロパティ用に特別に設計されたメソッドを取得するまでは続きません。

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くの詳細を指定する必要があります。ただし、ユースケースの大部分は、ごく一部の設定でカバーされることがよくあります。一連の列挙値を使用してこれらの設定を表すと、必要なコードの量を大幅に減らすことができます。

例えば、このセクションの前半の S3 バケット L2 コード例では、CloudFormation テンプレートの bucketEncryptionプロパティを使用して、使用する暗号化アルゴリズムの名前を含むすべての

詳細を指定する必要があります。代わりに、 AWS CDK `BucketEncryption` は 列挙型を提供します。この列挙型は、バケット暗号化の最も一般的な 5 つの形式を取り、単一の変数名を使用してそれぞれを表現できます。

列挙型でカバーされていないエッジケースについてはどうですか？ L2 コンストラクトの目標の 1 つは、レイヤー 1 リソースをプロビジョニングする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場合があります。これらのエッジケースをサポートするために、 AWS CDK では addPropertyOverride メソッドを使用して、基盤となる CloudFormation リソースプロパティを直接操作できます。プロパティの上書きの詳細については、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「抽象化とエスケープハッチ」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を設定するために必要なプログラムロジックを実行できない場合があります。このような状況では、 が代わりにヘルパークラスを提供することが AWS CDK よくあります。列挙型は、一連のキーと値のペアを提供するシンプルなオブジェクトですが、ヘルパークラスは TypeScript クラスの完全な機能を提供します。ヘルパークラスは静的プロパティを公開することで列挙型のように動作できますが、これらのプロパティは、ヘルパークラスコンストラクタまたはヘルパーメソッドで条件ロジックを使用して内部的に値を設定することができます。

したがって`BucketEncryption`、列挙型は S3 バケットに暗号化アルゴリズムを設定するために必要なコードの量を減らすことができますが、選択できる値が多すぎるため、同じ戦略は期間の設定には機能しません。値ごとに列挙型を作成すると、値の価値よりもはるかに問題になります。このため、ObjectLockRetention クラスで表される S3 バケットのデフォルトの S3 オブジェクトロック設定にはヘルパークラスが使用されます。 `ObjectLockRetention`には、コンプライアンス保持用の 1 つとガバナンス保持用の 2 つの静的メソッドが含まれています。どちらの方法でも、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、ロックを設定する時間を表します。

もう 1 つの例は、 AWS Lambda ヘルパークラスランタイムです。一見すると、このクラスに関連付けられた静的プロパティを列挙型で処理できるように見える場合があります。ただし、内部では、各プロパティ値は`Runtime`クラス自体のインスタンスを表すため、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内では達成できませんでした。

レイヤー 3 コンストラクト

L1 コンストラクトが CloudFormation リソースをプログラムコードにリテラル変換し、L2 コンストラクトが詳細な CloudFormation 構文の多くをヘルパーメソッドとカスタムロジックに置き換える場合、L3 コンストラクトは何をしますか？ その答えは、想像力によってのみ制限されます。特定のユースケースに合わせてレイヤー 3 を作成できます。プロジェクトに特定のプロパティのサブセットを持つリソースが必要な場合は、そのニーズを満たすために再利用可能な L3 コンストラクトを作成できます。

L3 コンストラクトは、 内のパターンと呼ばれます AWS CDK。パターンは、 のConstructクラスを拡張する AWS CDK （ または クラスを拡張するクラスを拡張するConstruct ）オブジェクトで、レイヤー 2 を超えて抽象化されたロジックを実行します。 AWS CDK CLI を使用して 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は、app、、 libの 3 つの AWS CDK アプリケーションタイプから選択する必要がありますsample-app。

```
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、CloudFormation スタックを構築して AWS 環境にデプロイする従来の AWS CDK アプリケーションを表します。を選択するとlib、まったく新しい L3 コンストラクトを構築することになります。 appと sample-appでは、 が AWS CDK サポートする任意の言語を選択できますが、 では TypeScript しか選択できませんlib。これは、 AWS CDK が TypeScript でネイティブに記述され、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されている言語に変換するためです。プロジェクトlibを開始する場合は、 の拡張機能を構築することを選択します AWS CDK。

Construct クラスを拡張するクラスは L3 コンストラクトにすることができますが、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 コンストラクトは、機能を拡張 AWS CDK するために、これら 3 つのケースのうち 1 つ以上を使用します。

リソースインタラクション

ソリューションは通常、連携する複数の AWS サービスを使用します。例えば、Amazon CloudFront ディストリビューションでは、S3 バケットをオリジンとして使用し、一般的なエクスプロイトから保護 AWS WAF することがよくあります。 AWS AppSync および Amazon API Gateway では、APIs のデータソースとして Amazon DynamoDB テーブルがよく使用されます。のパイプラインでは AWS CodePipeline 、多くの場合Amazon S3 をソースとして使用し、ビルドステージ AWS CodeBuild に使用します。このような場合、相互接続された 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単一の L3 コンストラクトを作成すると便利です。 L2

CloudFront ディストリビューションを SL3 オリジン、その前に配置 AWS WAF する 、Amazon Route 53 レコード、転送中の暗号化でカスタムエンドポイントを追加するための AWS Certificate Manager (ACM) 証明書とともにプロビジョニングする L3 コンストラクトの例を次に示します。これらはすべて 1 つの再利用可能なコンストラクトです。 S3

```
// 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、および ACM はすべて L2 コンストラクトを使用しますが、ウェブ ACL (ウェブリクエストを処理するためのルールを定義する) は L1 コンストラクトを使用することに注意してください。これは、 AWS CDK は完全には完了していない進化するオープンソースパッケージであり、 の L2 コンストラクトWebAclがまだないためです。ただし、新しい L2 コンストラクトを作成 AWS CDK することで、誰でも に貢献できます。したがって、 が の L2 コンストラクト AWS CDK を提供するまではWebAcl、L1 コンストラクトを使用する必要があります。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CloudFrontWebsite、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 L2 Distribution コンストラクトは L3 コンストラクトのパブリックプロパティとして公開されています。このような L3 プロパティを必要に応じて公開する必要がある場合もあります。実際には、後で「カスタムリソース」セクションでDistributionもう一度確認します。

には、このようなリソースインタラクションパターンの例がいくつか AWS CDK 含まれています。Amazon Elastic Container Service (Amazon ECS) の L2 コンストラクトを含む aws-ecs パッケージに加えて、 AWS CDK には aws-ecs-patterns というパッケージがあります。このパッケージには、Amazon ECS を Application Load Balancer、Network Load Balancer、およびターゲットグループと組み合わせながら、Amazon Elastic Compute Cloud (Amazon EC2) および 用にプリセットされた異なるバージョンを提供する複数の L3 コンストラクトが含まれています AWS Fargate。多くのサーバーレスアプリケーションは Fargate でのみ Amazon ECS を使用するため、これらの L3 コンストラクトは開発者の時間と顧客のコストを削減できる利便性を提供します。

リソース拡張機能

一部のユースケースでは、L2 コンストラクトにネイティブではない特定のデフォルト設定を持つリソースが必要です。スタックレベルでは、これは側面を使用して処理できますが、L2 コンストラクトに新しいデフォルトを与えるもう 1 つの便利な方法は、レイヤー 2 を拡張することです。コンストラクトは クラスを継承する任意のConstructクラスであり、L2 コンストラクトはそのクラスを拡張するため、L2 コンストラクトを直接拡張して L3 コンストラクトを作成することもできます。L2

これは、顧客の単一のニーズをサポートするカスタムビジネスロジックに特に役立ちます。ある会社で、すべての AWS Lambda 関数コードを という単一のディレクトリに保存src/lambdaし、ほとんどの Lambda 関数が毎回同じランタイムとハンドラー名を再利用しているとします。新しい Lambda 関数を設定するたびにコードパスを設定する代わりに、新しい L3 コンストラクトを作成できます。

```
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
     });
}
```

その後、次のようにリポジトリ内の任意の場所で L2 Functionコンストラクトを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
});
```

デフォルトにより、新しい Lambda 関数を 1 行で作成でき、L3 コンストラクトが設定されているため、必要に応じてデフォルトのプロパティを上書きできます。

既存の L2 コンストラクトに新しいデフォルトを追加するだけの場合、L2 コンストラクトを直接拡張するのが最適です。他のカスタムロジックも必要な場合は、 `Construct` クラスを拡張することをお勧めします。その理由は、コンストラクタ内で呼び出される `super`メソッドに由来しています。他のクラスを拡張するクラスでは、 `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されるものです。つまり、渡された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 コンストラクトをインスタンス化する前にこのカスタムロジックのいずれか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、スタックのデプロイ中にアクティブ化された Lambda 関数からカスタムロジックを実行できます。CloudFormation によって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になる場合は、カスタムリソースを使用してそれを実現できます。 AWS CDK には、カスタムリソースをプログラムで作成できる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ことで、ほとんどすべて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。オリジンに加えられた新しい変更をウェブサイトにすぐに反映するように、そのキャッシュを手動でリセットする場合は、CloudFront の無効化を使用できます。ただし、無効化は、CloudFront ディストリビューションのプロパティではなく、CloudFront ディストリビューションで実行されるプロセスです。これらはいつでも作成して既存のディストリビューションに適用できるため、プロビジョニングとデプロイのプロセスにネイティブには含まれません。

このシナリオでは、ディストリビューションのオリジンが更新されるたびに無効化を作成して実行できます。カスタムリソースのため、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')
                     }
                  }
             }
          }
     }
}
```

CloudFrontWebsite L3 コンストラクトで先ほど作成した 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行うことができます。

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

この L3 コンストラクトは、AwsCustomResource と呼ばれる 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。AwsCustomResourceは、Lambda コードを記述しなくても AWS SDK 呼び出しを 1 回だけ行う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、独自のロジックを実装する場合は、基本的な CustomResource クラスを直接使用できます。

カスタムリソース L3 コンストラクト AWS CDK を使用するもう 1 つの良い例は、S3 バケットデプロイです。この L3 コンストラクトのコンストラクタ内のカスタムリソースによって作成された

Lambda 関数は、CloudFormation が処理できない機能を追加します。S3 バケット内のオブジェクトを追加および更新します。S3 バケットのデプロイがないと、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配置できないため、非常に不便になります。

CloudFormation 構文のリームを書き出す必要がなく AWS CDK なる最適な例は、次の基本的な ですS3BucketDeployment。

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

同じことを実現するために記述する必要がある CloudFormation コードと比較します。

```
"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 行と 241 行は大きな違いです。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に可能なことの一例にすぎません。

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は必ずしも避けることはできませんが、可能な限り避ける必要があります。特定の L2 コンストラクトがエッジケースをサポートしていない場合は、L1 コンストラクトを直接使用する代わりに、次の 2 つのオプションを調べることができます。
  - アクセス`defaultChild`： 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、 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`L2Construct.node.defaultChild`。L1 コンストラクトのパブリックプロパティは、自分で L1 コンストラクトを作成するのではなく、このプロパティを通じてアクセスすることで更新できます。
  - プロパティオーバーライドを使用する: 更新するプロパティがパブリックでない場合はどうなりますか？ CloudFormation テンプレートが実行できるすべての操作 AWS CDK を に許可する最終的なエスケープハッチは、すべての L1 コンストラクトで使用できるメソッド addPropertyOverride を使用することです。CloudFormation テンプレートレベルでスタックを操作するには、CloudFormation プロパティ名と値をこのメソッドに直接渡します。

L2 コンストラクト

- L2 コンストラクトでよく提供されるヘルパーメソッドを必ず活用してください。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 ヘルパーメソッドは、特に条件付きロジックが必要な場合に、リソースのプロビジョニングを指数関数的に便利にすることができます。最も便利なヘルパーメソッドの 1 つは Grant クラスから派生しています。このクラスは直接使用されませんが、多くの L2 コンストラクトはそれを使用して、アクセス許可を実装しやすくするヘルパーメソッドを提供します。例えば、L2 S3 バケットにアクセスするためのアクセス許可を L2 Lambda 関数に付与する場合は、新しいロールとポリシーを作成する`s3Bucket.grantReadWrite(lambdaFunction)`代わりに を呼び出すことができます。

L3 コンストラクト

- L3 コンストラクトは、スタックをより再利用しやすくカスタマイズ可能なものにする場合に非常に便利ですが、慎重に使用することをお勧めします。どのタイプの L3 コンストラクトが必要か、または L3 コンストラクトをまったく必要かを検討します。
  - AWS リソースを直接操作していない場合は、クラスを拡張するのではなく、ヘルパー`Construct`クラスを作成する方が適切な場合があります。これは、 `Construct` クラス

が、 AWS リソースと直接やり取りしている場合にのみ必要な多くのアクションをデフォルトで実行するためです。したがって、これらのアクションを実行する必要がない場合は、それらを回避する方が効率的です。

- 新しい L3 コンストラクトの作成が適切であると判断した場合は、ほとんどの場合、`Construct`クラスを直接拡張する必要があります。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、他の L2 コンストラクトを拡張します。他の L2 コンストラクトまたはカスタムロジックが含まれる場合は、`Construct`直接拡張し、コンストラクタ内の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？

できることは間違いありません。しかし、最も強力なツールと同様に、 は、その知識が深まるほど強力 AWS CDK になります。 AWS CDKの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のと同じ方法で L1 から L2 コンストラクトを作成できますか？

リソースにすでに 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。これは、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適な方法をすでに多くの調査で検討しているためです。ただし、L1L21 コンストラクトがいくつかあります。このような場合は、独自の L2 コンストラクトを作成し、オープンソースライブラリの AWS CDK 寄稿者になることで他のユーザーと共有することをお勧めします。開始するために必要なものはすべて、 のコントリビューションガイドラインに記載されています AWS CDK。

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 リソースの数は日単位で減少していますが、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。左ペインのリソースのリストを確認します。名前の横にスーパースクリプト 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。

がサポートする AWS CDK 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。これは、TypeScript が にネイティブな唯一の言語であるためです AWS CDK。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。

の外部にある既存の L3 コンストラクトはどこにあります AWS CDK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 セクションにあります。

リソース

- AWS CDK API リファレンス
- AWS CloudFormation リソース仕様
- AWS CDK コンストラクトドキュメント
- AWS CDK 抽象化とエスケープハッチ
- L2 コンストラクトを活用して AWS CDK アプリケーションの複雑さを軽減する (AWS ブログ記事)
- AWS CloudFormation カスタムリソース
- AWS ソリューションコンストラクト
- コンストラクトハブ
- AWS CDK 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の最適化レベルを導入します。例: オンプレミスの Oracle データベースを の 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: カスタマーリレーションシップ管理 (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: 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。

- 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

アトミック性、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。より柔軟ですが、アクティブ/パッシブ移行よりも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が同期されるデータベース移行方法。ただし、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、ソースデータベースのみ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します。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

行のグループで動作し、グループの単一の戻り値を計算する SQL 関数。集計関数の例としては、 `SUM`や などがありま`MAX`す。

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。

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

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のための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

個人や組織を混乱させたり、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 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。エンディアン性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を 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を別の環境 (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、ボットハー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制御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、「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、 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 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。

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。

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

「データキャプチャの変更」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します。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

組織が に移行するときに通常実行する 4 つのフェーズ 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。AWS 移行戦略とどのように関連しているかについては、「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、 GitHubまたは が含まれます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、Amazon SageMaker AI は CV 用の画像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、設定が想定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、通常は段階的かつ意図的ではありません。

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

一元管理とガバナンスを備えた分散型の分散型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。信頼された ID のみが、予想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通常、大量の履歴データが含まれており、通常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。

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。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。

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一般的に使用されます。

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)

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に使用する戦略とプロセス。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) 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

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、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、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のに役立ちます。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 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

同様のタスクの実行を求める前に、タスクと必要な出力を示す少数の例を LLM に提供します。この手法は、プロンプトに埋め込まれた例 (ショット) からモデルが学習するコンテキスト内学習のアプリケーションです。少数ショットプロンプトは、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

段階的なアプローチを使用する代わりに、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)

一般化および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、テキストと画像の生成、自然言語の会話など、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、「基盤モデルとは」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)

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 ワークフローはレガシーと見なされ、トランクベースのワークフローはモダンで推奨されるアプローチです。

ゴールデンイメージ

その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、、Amazon GuardDuty AWS Security Hub、、 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。モデル予測をホールドアウトデータと比較することで、ホールドアウトデータを使用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

「産業用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、または変更する代わりに、本番環境のワークロード用に新しいインフラストラクチャをデプロイするモデル。イミュータブルインフラストラクチャは、本質的に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、予測性が高くなります。詳細については、 AWS 「 Well-Architected Framework」の「Deploy using immutable infrastructure best practice」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 によって導入された用語で、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。

産業分野における 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、「Building an industrial Internet of Things (IIoT) digital transformation strategy」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 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

「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 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)

大量のデータに対し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、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 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。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で、原材料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、調整を行う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 を除くすべて 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、パブリッシュ/サブスクライブパターンに基づく軽量 machine-to-machine (M2M) 通信プロトコル。

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)

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 に移行するために使用するアプローチ AWS クラウド。詳細については、この用語集の「7 Rs エントリ」および「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

「Open Process Communications - Unified Architecture」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)

産業用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。

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)

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とそれに関連するベストプラクティスのチェックリスト。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 変換の主な焦点です。

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、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。

オリジンアクセスコントロール (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コントロールが可能です。

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

アクセス許可を定義 (アイデンティティベースのポリシーを参照)、アクセス条件を指定 (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を定義 AWS Organizations （ サービスコントロールポリシーを参照) できるオブジェクト。

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。一般的に`false`は `WHERE`句にあります。

述語プッシュダウン

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、、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内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じて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

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

非準拠の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まで、製品のデータとプロセスのライフサイクル全体にわたる管理。

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、製造プロセスを自動化する、信頼性の高い適応可能なコンピュータです。

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、予備応答を反復的に絞り込んだり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。

パブリッシュ/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。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 MES では、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、情報 (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、情報 (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。

再設計

「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 とは独立しています。詳細については、「アカウントで使用できる を指定する AWS リージョン」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 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。

プラットフォーム変更

「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

中断に抵抗または回復するアプリケーションの機能。高可用性とディザスタリカバリは、 で回復性を計画する際の一般的な考慮事項です AWS クラウド。詳細については、AWS クラウド「レジリエンス」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。サポートを含めると、そのマトリックスは RASCI マトリックスと呼ばれ、サポートを除外すると 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)

LLM がレスポンスを生成する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または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

攻撃者が認証情報にアクセスすることをより困難にするために、シークレットを定期的に更新するプロセス。

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 Management Console 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 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。

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に自動的に応答または修復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としては、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

AWS のサービス 送信先にあるデータの、それを受け取る による暗号化。

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

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

「サービスレベルの契約」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、『』の「アプリケーションのモダナイズに対する段階的なアプローチ 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。

スタースキーマ

1 つの大きなファクトテーブルを使用してトランザクションデータまたは測定データを保存し、1 つ以上の小さなディメンションテーブルを使用してデータ属性を保存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。

監視コントロールとデータ取得 (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本番稼働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 CloudWatch Synthetics 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

LLM にコンテキスト、指示、またはガイドラインを提供して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、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

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「 を他の AWS のサービス AWS Organizations 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

2 枚のピザで養う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。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」、「読み取り多数」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。承認されたユーザーは、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

LLM にタスクを実行する手順を提供しますが、タスクのガイドに役立つ例 (ショット) はありません。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。