



图 AWS CDK 层指南

AWS 规范性指导



AWS 规范性指导: 图 AWS CDK 层指南

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

简介	1
第 1 层构造	3
L1 构造的 AWS CDK-CloudFormation 生命周期	3
AWS CloudFormation 资源规范	4
第 2 层构造	6
默认属性	8
结构、类型和接口	8
静态方法	9
帮助程序方法	9
枚举	10
帮助程序类	11
第 3 层构造	12
资源交互	12
资源扩展	14
自定义资源	15
最佳实践	24
常见问题解答	25
如果不了解图层，我 AWS CDK 就不能使用吗？	25
我能否像从 L2 创建 L3 构造一样，从 L1 创建 L2 构造？	25
哪些 AWS 资源还没有官方的 L2 结构？	25
我能用它们支持的任何语言制作 L2 或 L3 结构吗？ AWS CDK	25
在哪里可以找到 AWS CDK 之外的现有 L3 构造？	25
资源	26
文档历史记录	27
术语表	28
#	28
A	28
B	31
C	32
D	35
E	38
F	40
G	41
H	42

我	43
L	45
M	46
O	50
P	52
Q	54
R	55
S	57
T	60
U	61
V	62
W	62
Z	63
.....	ixiv

AWS CDK 层指南

Steven Guggenheimer , Amazon Web Services (AWS)

2023 年 12 月 ([文档历史记录](#))

AWS Cloud Development Kit (AWS CDK)背后的主要概念之一与在寒冷的天气里保持温暖的概念非常相似。该概念称为分层。在寒冷的天气里，您会穿一件衬衫、一件夹克，有时还会穿一件更厚的夹克，具体取决于天气有多冷。如果进屋后发现暖气很热，您可以脱掉一件或两件外套，这样就不会太热了。AWS CDK 使用分层为使用云组件提供不同的抽象级别。分层可确保您在部署基础设施即代码 (IaC) 堆栈时，永远不会编写过多的代码或对资源属性的访问权限过少。

如果不使用 AWS CDK，您必须手动编写 [AWS CloudFormation](#) 模板；也就是说，您只利用了单一层，这迫使您编写的代码远远超过通常所需的代码。另一方面，如果 AWS CDK 将 CloudFormation 中通常不需要编写的所有内容都抽象化，那么就无法处理任何边缘情况。

为了解决这个问题，AWS CDK 将资源预调配分为三个独立且不同的层：

- 第 1 层 – CloudFormation 层：最基本的层，其中 CloudFormation 资源和 AWS CDK 资源几乎相同。
- 第 2 层 – 精心策划的层：将 CloudFormation 资源抽象成编程类的层，这些类在后台简化了大部分样板 CloudFormation 语法。此层构成了大部分 AWS CDK。
- 第 3 层 – 模式层：最抽象的层，在此层，您可以使用第 1 层和第 2 层提供的构建块针对您的特定使用案例自定义代码。

每个层中的每个项目都是一个名为 Construct 的特殊 AWS CDK 类的实例。根据 [AWS 文档](#)，构造是“AWS CDK 应用程序的基本构建块。构造代表‘云组件’，它封装了 AWS CloudFormation 创建该组件所需的一切。”这些层内的构造称为 L1、L2 和 L3 构造，具体取决于它们所属的层。在本指南中，我们将浏览每个 AWS CDK 层，以了解它们的用途及其重要的原因。

本指南适用于有兴趣深入研究使 AWS CDK 正常工作的核心概念的技术经理、主管和开发者。AWS CDK 是一款流行的工具，但团队常常会错过它所提供的很多功能。当您开始理解本指南中描述的概念时，就可以开启一个充满无限可能的新世界，并优化团队的资源预调配流程。

在本指南中：

- [第 1 层构造](#)
- [第 2 层构造](#)

- [第 3 层构造](#)
- [最佳实践](#)
- [常见问题解答](#)
- [资源](#)。

第 1 层构造

[L1 构造](#)是 AWS CDK 的构建块，可以很容易地通过前缀 Cfn 与其他构造区分开来。例如，AWS CDK 中的 Amazon DynamoDB 软件包包含 Table 构造，它是 L2 构造。相应的 L1 构造称为 CfnTable，它直接表示 CloudFormation DynamoDB Table。虽然 AWS CDK 应用程序通常从不直接使用 L1 构造，但如果不访问第一层，就无法使用 AWS CDK。但是，在大多数情况下，开发者习惯使用的 L2 和 L3 构造在很大程度上依赖于 L1 构造。因此，您可以将 L1 构造视为 CloudFormation 和 AWS CDK 之间的桥梁。

AWS CDK 的唯一目的是使用标准编码语言生成 CloudFormation 模板。运行 cdk synth CLI 命令并生成 CloudFormation 模板后，AWS CDK 的作业就完成了。cdk deploy 命令只是为了方便而存在的，但运行该命令时所执行的操作完全是在 CloudFormation 内完成的。将 AWS CDK 代码转换为 CloudFormation 可理解的格式的关键在于 L1 构造。

L1 构造的 AWS CDK–CloudFormation 生命周期

创建和使用 L1 构造的过程包括以下步骤：

1. AWS CDK 构建过程将 CloudFormation 规范转换为 L1 构造形式的编程代码。
2. 开发者编写的代码可以直接或间接引用 L1 构造作为 AWS CDK 应用程序的一部分。
3. 开发者运行 cdk synth 命令将编程代码转换回 CloudFormation 规范（模板）规定的格式。
4. 开发者运行 cdk deploy 命令将这些模板内的 CloudFormation 堆栈部署到 AWS 账户环境中。

我们来做个小练习。前往 GitHub 上的 [AWS CDK 开源存储库](#)，随机选择一种 AWS 服务，然后转到该服务的 AWS CDK 软件包（位于文件夹 packages、aws-cdk-lib、aws-`<servicename>`、lib 中）。在本示例中，让我们选择 Amazon S3，但这适用于任何服务。如果您查看该软件包的主 [index.ts 文件](#)，您会看到一行内容为：

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

但是，您不会在相应目录中的任何地方看到 s3.generated 文件。这是因为 L1 构造是在 AWS CDK 构建过程中根据 [CloudFormation 资源规范](#) 自动生成的。因此，只有在您运行软件包的 AWS CDK 构建命令后，您才会在软件包中看到 s3.generated。

AWS CloudFormation 资源规范

AWS CloudFormation 资源规范定义了 AWS 的基础设施即代码 (IAC) ，并确定了 CloudFormation 模板内的代码如何转换为 AWS 账户中的资源。此规范在每个区域级别以 [JSON 格式](#) 定义 AWS 资源。每个资源都有一个唯一的 [资源类型名称](#) ，该名称遵循 `provider::service::resource` 格式。例如，Amazon S3 存储桶的资源类型名称将为 `AWS::S3::Bucket` ，Amazon S3 接入点的资源类型名称将为 `AWS::S3::AccessPoint` 。这些资源类型可以使用 AWS CloudFormation 资源规范中定义的语法在 CloudFormation 模板中呈现。AWS CDK 构建过程运行时，每种资源类型也将变为 L1 构造。

因此，每个 L1 构造都是其相应的 CloudFormation 资源的编程镜像。当您实例化 L1 构造时，您要在 CloudFormation 模板中应用的每个属性都可用；在实例化相应的 L1 构造时，还需要将每个必需的 CloudFormation 属性作为参数。下表将 CloudFormation 模板中表示的 S3 存储桶与定义为 AWS CDK L1 构造的相同 S3 存储桶进行了比较。

CloudFormation 模板

```
"amzns3demobucket": {
  "Type": "AWS::S3::Bucket",
  "Properties": {
    "BucketName": "amzn-s3-demo-
bucket",
    "BucketEncryption": {
      "ServerSideEncryptionConfig
uration": [
        {
          "ServerSideEncrypt
ionByDefault": {
            "SSEAlgorithm": "AES256"
          }
        }
      ]
    },
    "MetricsConfigurations": [
      {
        "Id": "myConfig"
      }
    ],
    "OwnershipControls": {
      "Rules": [
        {
```

L1 构造

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

```
        "ObjectOwnership":
        "BucketOwnerPreferred"
      }
    ],
    "PublicAccessBlockConfigura
tion": {
      "BlockPublicAcls": true,
      "BlockPublicPolicy": true,
      "IgnorePublicAcls": true,
      "RestrictPublicBuckets": true
    },
    "VersioningConfiguration": {
      "Status": "Enabled"
    }
  }
}
```

```
    ],
    },
    publicAccessBlockConfiguration: {
      blockPublicAcls: true,
      blockPublicPolicy: true,
      ignorePublicAcls: true,
      restrictPublicBuckets: true
    },
    versioningConfiguration: {
      status: "Enabled"
    }
  });
```

如您所见，L1 构造是 CloudFormation 资源代码中的精确体现。没有捷径或简化方法，因此必须编写的样板文本数量大致相同。但是，使用 AWS CDK 的主要优势之一应该是它有助于消除许多样板 CloudFormation 语法。那么，这是怎么发生的呢？这就是 L2 构造发挥作用的地方。

第 2 层构造

[AWS CDK 开源存储库](#) 主要使用 [TypeScript](#) 编程语言编写，由许多包和模块组成。名为 `aws-cdk-lib` 的主软件包库大致分为每项 AWS 服务一个软件包，尽管情况并非总是如此。如前文所述，L1 构造是在构建过程中自动生成的，那么当您查看存储库内部时，您看到的全部代码是什么？这些是 [L2 构造](#)，它们是 L1 构造的抽象。

这些软件包还包含 TypeScript 类型、枚举和接口的集合，以及添加更多功能的帮助程序类，但这些项目都提供 L2 构造。所有 L2 构造在实例化时都会在其构造函数中调用相应的 L1 构造，并且可以从第 2 层访问所创建的 L1 构造，如下所示：

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

L2 构造采用默认属性、便利方法及其他语法糖，并将其应用于 L1 构造。这消除了直接在 CloudFormation 中预调配资源所必需的大部分重复和冗长。

所有 L2 构造都是在后台构建相应的 L1 结构。但是，L2 构造实际上并没有扩展 L1 构造。L1 和 L2 构造都继承一个名为 [Construct](#) 的特殊类别。在 AWS CDK 的第 1 版中，Construct 类别已内置于开发工具包中，但在第 2 版中，它是一个单独的[独立软件包](#)。这样，其他软件包 [例如 [Terraform 的云开发工具包 \(CDKTF\)](#)] 就可以将其作为依赖项包含在内。继承 Construct 类别的任何类别都是 L1、L2 或 L3 构造。L2 构造直接扩展此类别，而 L1 构造扩展名为 `CfnResource` 的类别，如下表所示。

L1 继承树

L1 构造

→ 类别 [CfnResource](#)

→→ 抽象类别 [CfnRefElement](#)

→→→ 抽象类别 [CfnElement](#)

→→→→ 类别 [Construct](#)

L2 继承树

L2 构造

→ 类别 [Construct](#)

如果 L1 和 L2 构造都继承 Construct 类别，为什么 L2 构造不直接扩展 L1？嗯，这是因为 Construct 类别和 L1 层之间的类别将 L1 构造锁定为 CloudFormation 资源的镜像。它们包含

抽象方法（下游类别必须包含的方法），例如 `_toCloudFormation`，这会强制构造直接输出 CloudFormation 语法。L2 构造会跳过这些类别，并直接扩展 `Construct` 类别。这使它们能够通过构造器中单独构建 L1 构造，来灵活地抽象 L1 构造所需的大部分代码。

上一节对 CloudFormation 模板中的 S3 存储桶和渲染为 L1 构造的相同 S3 存储桶进行了并排比较。该比较表明，属性和语法几乎相同，与 CloudFormation 构造相比，L1 构造仅节省了三四行。现在，让我们比较同一 S3 存储桶的 L1 构造和 L2 构造：

用于 S3 存储桶的 L1 构造

```
new CfnBucket(this, "amzn-s3demo-bucket", {
  bucketName: "amzn-s3-demo-bucket",
  bucketEncryption: {
    serverSideEncryptionConfiguration: [
      {
        serverSideEncryptionByDefault: {
          sseAlgorithm: "AES256"
        }
      }
    ],
    metricsConfigurations: [
      {
        id: "myConfig"
      }
    ],
    ownershipControls: {
      rules: [
        {
          objectOwnership: "BucketOwnerPreferred"
        }
      ]
    },
    publicAccessBlockConfiguration: {
      blockPublicAcls: true,
      blockPublicPolicy: true,
      ignorePublicAcls: true,
      restrictPublicBuckets: true
    }
  }
});
```

用于 S3 存储桶的 L2 构造

```
new Bucket(this, "amzn-s3demo-bucket", {
  bucketName: "amzn-s3-demo-bucket",
  encryption: BucketEncryption.S3_MANAGED,
  metrics: [
    {
      id: "myConfig"
    }
  ],
  objectOwnership: ObjectOwnership.BUCKET_OWNER_PREFERRED,
  blockPublicAccess: BlockPublicAccess.BLOCK_ALL,
  versioned: true
});
```

```
    },  
    versioningConfiguration: {  
      status: "Enabled"  
    }  
  }  
});
```

如您所见，L2 构造的大小不到 L1 构造的一半。L2 构造使用多种技术来完成此整合。其中一些技术适用于单个 L2 构造，但其他技术可以在多个构造中重复使用，因此它们被分成自己的类别以实现可重用性。L2 构造通过多种方式整合 CloudFormation 语法，如以下各部分所述。

默认属性

整合预调配资源代码的最简单方法是将最常用的属性设置转换为默认值。AWS CDK 可以使用强大的编程语言，而 CloudFormation 则不能，因此这些默认值本质上通常是有条件的。有时，可以从 AWS CDK 代码中删除几行 CloudFormation 配置，因为这些设置可以从传递给构造的其他属性的值中推断出来。

结构、类型和接口

虽然 AWS CDK 提供多种编程语言版本，但它是使用 TypeScript 原生编写的，因此该语言的类型系统用于定义构成 L2 构造的类型。深入研究该类型系统超出了本指南的范围；有关详细信息，请参阅 [TypeScript 文档](#)。总而言之，TypeScript type 描述了特定变量所包含的数据类型。这可能是基本数据（例如 string），也可能是更复杂的数据（例如 object）。TypeScript interface 是表达 TypeScript 对象类型的另一种方式，而 struct 是接口的另一个名称。

TypeScript 不使用结构这个词，但是如果您查看 [AWS CDK API 参考](#)，您会发现一个结构实际上只是代码内的另一个 TypeScript 接口。API 参考也将某些接口称为接口。如果结构和接口是同一回事，为什么 AWS CDK 文档要区分它们呢？

AWS CDK 中所谓的结构是指代表 L2 构造所使用的任何对象的接口。其中包括在实例化期间传递给 L2 构造的属性参数的对象类型，例如 S3 存储桶构造的 BucketProps 和 DynamoDB 表构造的 TableProps，以及 AWS CDK 中使用的其他 TypeScript 接口。简而言之，如果是 AWS CDK 中的 TypeScript 接口，并且其名称没有以字母 I 为前缀，则 AWS CDK 会将其称为结构。

相反，AWS CDK 使用术语接口来表示普通对象需要具备的基本元素，才能视为特定构造或帮助程序类的正确表示。也就是说，接口描述了 L2 构造的公共属性必须是什么。所有 AWS CDK 接口名称都是以字母 I 为前缀的现有构造或帮助程序类的名称。所有 L2 构造都扩展了 Construct 类别，但它们也实现了相应的接口。因此，L2 构造 Bucket 实现了 IBucket 接口。

静态方法

L2 构造的每个实例也是其相应接口的实例，但反过来不成立。在浏览结构以查看需要哪些数据类型时，这一点很重要。如果结构具有名为 bucket 的属性，该属性需要数据类型 IBucket，则可以传递一个包含 IBucket 接口中列出的属性的对象，或传递一个 L2 Bucket 的实例。任何一种方法都可以。但是，如果该 bucket 属性调用 L2 Bucket，则只能在该字段中传递一个 Bucket 实例。

当您预先存在的资源导入堆栈时，这种区别变得非常重要。您可以为堆栈原生的任何资源创建 L2 构造，但是如果您需要引用在堆栈之外创建的资源，则必须使用该 L2 构造的接口。这是因为，如果堆栈内尚不存在该资源，则创建 L2 构造会创建一个新资源。对现有资源的引用必须是符合该 L2 构造接口的普通对象。

为了方便实践，大多数 L2 构造都有一组与之关联的静态方法，这些方法返回该 L2 构造的接口。这些静态方法通常以单词 from 开头。传递给这些方法的前两个参数与标准 L2 构造所需的 scope 和 id 参数相同。然而，第三个参数不是 props，而是定义接口的一小部分属性（有时甚至只有一个属性）。因此，当您传递 L2 构造时，大多数情况下只需要接口的元素。这样，您也可以在可能的情况下使用导入的资源。

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

但是，您不应该过分依赖接口。只有在绝对必要时才应导入资源并直接使用接口，因为接口不提供使 L2 构造如此强大的许多属性，例如帮助程序方法。

帮助程序方法

L2 构造是一个编程类别，而不是一个简单的对象，因此它可以公开类别方法，允许您在实例化完成后操作资源配置。这方面的一个很好的示例是 AWS Identity and Access Management (IAM) L2 [角色](#) 构造。以下片段显示了使用 L2 Role 构造创建相同 IAM 角色的两种方法。

不使用帮助程序方法：

```
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' ]
            })
        ]
    })
}
```

使用帮助程序方法：

```
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' ]
        })
    ]
}));
```

L2 构造能够使用实例方法在实例化后操作资源配置，这使得它比上一层具有更大的灵活性。L1 构造还会继承一些资源方法（例如 `addPropertyOverride`），但是直到第二层才获得专门为该资源及其属性设计的方法。

枚举

CloudFormation 语法通常要求您指定许多细节才能正确预调配资源。但是，大多数使用案例通常只能由少数几种配置覆盖。使用一系列枚举值来表示这些配置可以极大减少所需的代码量。

例如，在本节前面的 S3 存储桶 L2 代码示例中，您必须使用 CloudFormation 模板的 `bucketEncryption` 属性来提供所有详细信息，包括要使用的加密算法的名称。相反，AWS CDK 提

供了 `BucketEncryption` 枚举，它采用了五种最常见的存储桶加密形式，并允许您使用单个变量名来表达每种形式。

枚举未涵盖的边缘情况该怎么办？L2 构造的目标之一是简化预调配第 1 层资源的任务，因此第 2 层可能不支持某些不太常用的边缘情况。为了支持这些边缘情况，AWS CDK 允许您使用 [addPropertyOverride](#) 方法直接操作底层 CloudFormation 资源属性。有关属性覆盖的更多信息，请参阅本指南的[最佳实践](#)部分以及 AWS CDK 文档中的[抽象和逃生舱口](#)部分。

帮助程序类

有时，枚举无法完成为给定使用案例配置资源所需的编程逻辑。在这些情况下，AWS CDK 通常会改为提供帮助程序类。枚举是一个提供一系列键值对的简单对象，而帮助程序类提供了 TypeScript 类别的全部功能。通过公开静态属性，帮助程序类仍然可以像枚举一样起作用，但是这些属性的值可以在帮助程序类构造函数或帮助程序方法中使用条件逻辑在内部设置。

因此，虽然 `BucketEncryption` 枚举可以减少在 S3 存储桶上设置加密算法所需的代码量，但同样的策略不适用于设置持续时间，因为有太多可能的值可供选择。为每个值创建枚举会麻烦得不值当。因此，将帮助程序类用于 S3 存储桶的默认 S3 对象锁定配置设置，如 [ObjectLockRetention](#) 类别所示。`ObjectLockRetention` 包含两种静态方法：一种用于合规性保留，另一种用于治理保留。这两种方法都将 [Duration 帮助程序类](#) 的实例作为参数，来表示应配置锁的时间。

另一个示例是 AWS Lambda 帮助程序类[运行时](#)。乍一看，与此类别关联的静态属性似乎可以通过枚举来处理。但是，在后台，每个属性值都代表 `Runtime` 类别本身的一个实例，因此在类别的构造函数中执行的逻辑无法在枚举内实现。

第 3 层构造

如果 L1 构造将 CloudFormation 资源直译为编程代码，而 L2 构造用帮助程序方法和自定义逻辑替换了大部分冗长的 CloudFormation 语法，那么 [L3 构造](#) 会做什么？答案只受您的想象力的限制。您可以创建第 3 层以适应任何特定的使用案例。如果您的项目需要具有特定属性子集的资源，则可以创建可重复使用的 L3 构造来满足该需求。

L3 构造在 AWS CDK 内称为模式。模式是指任何可扩展 AWS CDK 中的 Construct 类别（或扩展可扩展 Construct 类别的类别）的对象，用于执行超出第 2 层的任何抽象逻辑。使用 AWS CDK CLI 运行 `cdk init` 启动新的 AWS CDK 项目时，必须从以下三种 AWS CDK 应用程序类型中选择：`app`、`lib` 和 `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` 和 `sample-app` 都代表经典 AWS CDK 应用程序，您可以在其中构建 CloudFormation 堆栈并将其部署到 AWS 环境。选择 `lib` 时，您就选择了构建一个全新的 L3 构造。`app` 和 `sample-app` 允许您选择 AWS CDK 支持的任何语言，但您只能选择带有 `lib` 的 TypeScript。这是因为 AWS CDK 本身是用 TypeScript 编写的，并且使用名为 [JSii](#) 的开源系统将原始代码翻译成其他受支持的语言。选择 `lib` 启动项目时，您就选择了构建 AWS CDK 的扩展。

任何可扩展 Construct 类别的类别都可以是 L3 构造，但第 3 层最常见的使用案例是资源交互、资源扩展和自定义资源。大多数 L3 构造使用这三种情况中的一种或多种来扩展 AWS CDK 功能。

资源交互

一个解决方案通常使用多个协同工作的 AWS 服务。例如，Amazon CloudFront 分发通常使用 S3 存储桶作为来源，并使用 AWS WAF 来防范常见漏洞攻击。AWS AppSync 和 Amazon API Gateway 通常使用 Amazon DynamoDB 表作为其 API 的数据来源。AWS CodePipeline 中的管线通常使用 Amazon S3 作为来源，并使用 AWS CodeBuild 作为构建阶段。在这些情况下，创建一个 L3 构造来处理两个或多个互连的 L2 构造的预调配通常很有用。

以下是一个 L3 构造示例，该构造预调配 CloudFront 分发及其 S3 源、部署在其前面的 AWS WAF、Amazon Route 53 记录以及 AWS Certificate Manager (ACM) 证书，以添加带有传输加密的自定义端点 – 所有这些都集中在一个可重复使用的构造中：

```
// 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,
    });
  }
}
```

请注意，CloudFront、Amazon S3、Route 53 和 ACM 都使用 L2 构造，但是 Web ACL（定义处理 Web 请求的规则）使用 L1 构造。这是因为 AWS CDK 是一个不断演变的开源软件包，尚未完全完成，目前还没有 WebAcl 的 L2 构造。但是，任何人都可以通过创建新的 L2 构造来为 AWS CDK 做出贡献。因此，在 AWS CDK 为 WebAcl 提供 L2 构造之前，您必须使用 L1 构造。要使用 L3 构造 CloudFrontWebsite 创建新网站，请使用以下代码：

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

在此示例中，CloudFront Distribution L2 构造作为 L3 构造的公共属性公开。在某些情况下，您仍然需要根据需要公开 L3 属性。实际上，我们稍后会在[自定义资源](#)部分中再次看到 Distribution。

AWS CDK 包括一些资源交互模式的示例，例如此示例。除了包含 Amazon Elastic Container Service (Amazon ECS) 的 L2 构造的 `aws-ecs` 软件包之外，AWS CDK 还有一个名为 [aws-ecs-patterns](#) 的软件包。此软件包包含多个 L3 构造，它们将 Amazon ECS 与应用程序负载均衡器、网络负载均衡器和目标组相结合，同时提供针对 Amazon Elastic Compute Cloud (Amazon EC2) 和 AWS Fargate 预设的不同版本。由于许多无服务器应用程序仅将 Amazon ECS 与 Fargate 配合使用，因此这些 L3 构造提供了便利，可以为开发者节省时间，为客户节省金钱。

资源扩展

某些使用案例要求资源具有特定的默认设置，而这些设置不是 L2 构造的原生设置。在堆栈级别，这可以通过使用[方面](#)来处理，但为 L2 构造赋予新默认值的另一种便捷方法是扩展第 2 层。由于构造是继承了 Construct 类别的任何类别，而 L2 构造又扩展了该类别，因此您也可以直接扩展 L2 构造来创建 L3 构造。

这对于支持客户单一需求的自定义业务逻辑特别有用。假设一家公司有一个存储库，其中存储了其所有 AWS Lambda 函数代码，这些代码位于名为 `src/lambda` 的单个目录中，并且大多数 Lambda 函数每次都重复使用相同的运行时和处理程序名称。您可以创建一个新的 L3 构造，而不必在每次配置新的 Lambda 函数时都配置代码路径：

```
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
    });
  }
}
```

然后，您可以按如下方式替换存储库中任何位置的 L2 Function 构造：

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

默认值允许您在单行上创建新的 Lambda 函数，且 L3 构造已设置，因此您仍然可以在需要时覆盖默认属性。

当您只想为现有 L2 构造添加新的默认值时，直接扩展 L2 构造效果最佳。如果您还需要其他自定义逻辑，最好扩展 Construct 类别。其原因源于在构造函数内调用的 super 方法。在可扩展其他类别的类别中，super 方法用于调用父类别的构造函数，并且这必须是构造函数内发生的第一件事。这意味着只有在创建了原始 L2 构造之后，才能对传递的参数或其他自定义逻辑进行任何操作。如果您需要在实例化 L2 构造之前执行任何自定义逻辑，则最好遵循之前在[资源交互](#)部分中概述的模式。

自定义资源

[自定义资源](#)是 CloudFormation 中的一项强大功能，可让您通过堆栈部署期间激活的 Lambda 函数运行自定义逻辑。当您在部署期间需要任何不受 CloudFormation 直接支持的流程时，都可以使用自定义资源来实现。AWS CDK 提供的类别还允许您以编程方式创建自定义资源。通过在 L3 构造函数内使用自定义资源，您几乎可以用任何东西制作一个构造。

使用 Amazon CloudFront 的优势之一是其强大的全球缓存功能。如果您想手动重置该缓存，以便您的网站立即反映对您的来源所做的新更改，则可以使用 [CloudFront 失效](#)。然而，失效是指在 CloudFront 分发上运行的进程，而不是 CloudFront 分发的属性。它们可以随时创建并应用于现有分发，因此它们并非预调配和部署过程的固有组成部分。

在这种情况下，您可能希望在每次更新分发来源后创建并运行失效。由于有自定义资源，您可以创建如下所示的 L3 构造：

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

export class CloudFrontInvalidation extends Construct {
```


AWS CDK 使用自定义资源 L3 构造的另一个很好的示例是 [S3 存储桶部署](#)。由此 L3 构造的构造函数内的自定义资源创建的 Lambda 函数添加了 CloudFormation 无法处理的功能：它在 S3 存储桶中添加和更新对象。如果没有 S3 存储桶部署，您将无法将内容放入作为堆栈一部分刚刚创建的 S3 存储桶中，这将非常不方便。

AWS CDK 消除编写大量 CloudFormation 语法需求的最佳示例是以下基本 S3BucketDeployment：

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

相比之下，要实现同样的功能，您需要编写以下 CloudFormation 代码：

```
"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": [
      "f888a9d977f0b5bdbbc04a1f8f07520ede6e00d4051b9a6a250860a1700924f26.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 行与 241 行差异很大！这只是利用第 3 层自定义堆栈所能实现的功能的一个示例。

最佳实践

L1 构造

- 您无法始终避免直接使用 L1 构造，但应尽可能避免这样做。如果特定的 L2 构造不支持您的边缘情况，您可以探索这两个选项，而不是直接使用 L1 构造：
 - 访问 `defaultChild`：如果您需要的 CloudFormation 属性在 L2 构造中不可用，则可以使用 `L2Construct.node.defaultChild` 访问底层 L1 构造。您可以通过此属性访问 L1 构造的任何公共属性来更新它们，而不必费心自己创建 L1 构造。
 - 使用属性覆盖：如果您要更新的属性不是公开的，该怎么办？允许 AWS CDK 实现 CloudFormation 模板所有功能的终极解决方案是使用每个 L1 构造中都可用的方法：[addPropertyOverride](#)。通过将 CloudFormation 属性名称和值直接传递给此方法，您可以在 CloudFormation 模板级别上操作堆栈。

L2 构造

- 记得利用 L2 构造经常提供的帮助程序方法。在第 2 层中，您不必在实例化时传递每个属性。L2 帮助程序方法可以使资源预调配变得更加方便，尤其是在需要条件逻辑时。最方便的帮助程序方法之一是从 [Grant](#) 类派生出来的。这个类不是直接使用的，但是许多 L2 构造使用它来提供帮助程序方法，使权限更容易实现。例如，如果您想授予 L2 Lambda 函数访问某个 L2 S3 存储桶的权限，则可以调用 `s3Bucket.grantReadWrite(lambdaFunction)` 而不是创建新的角色和策略。

L3 构造

- 虽然当您想让堆栈更具可重复使用性和可自定义时，L3 构造可能非常方便，但我们建议您谨慎使用它们。考虑您需要哪种类型的 L3 构造，或者您是否需要 L3 构造：
 - 如果您不直接与 AWS 资源交互，则通常更适合创建帮助程序类而不是扩展该 `Construct` 类。这是因为默认情况下，该 `Construct` 类会执行许多操作，只有在您直接与 AWS 资源交互时才需要执行这些操作。因此，如果您不需要执行这些操作，则避免这些操作会更有效。
 - 如果您确定创建新的 L3 构造是合适的，那么在大多数情况下，您需要直接扩展该 `Construct` 类。仅在想要更新其他 L2 构造的默认属性时，才扩展该构造。如果涉及其他 L2 构造或自定义逻辑，请直接扩展 `Construct`，并实例化构造函数中的所有资源。

常见问题解答

如果不了解图层，我 AWS CDK 就不能使用吗？

您绝对可以。但是，与大多数强大的工具一样，您对它的了解 AWS CDK 越多，功能就越强大。学习各层 AWS CDK 的交互方式可以将理解提升到一个新的水平，这有助于简化堆栈部署，远远超出仅凭基础 AWS CDK 知识所能做的事情。

我能否像从 L2 创建 L3 构造一样，从 L1 创建 L2 构造？

如果资源已经有 L2 构造，我们建议您使用该构造，并在第 3 层进行自定义。这是因为已经有大量研究投入到寻找针对特定资源配置现有 L2 构造的最佳方法。然而，有几种 L1 构造的 L2 构造尚不存在。在这种情况下，我们鼓励您创建自己的 L2 构造，并通过成为 AWS CDK 开源库的贡献者与他人共享。您可以在 AWS CDK 的[贡献准则](#)中找到入门所需的一切。

哪些 AWS 资源还没有官方的 L2 结构？

[没有 L2 结构的 AWS 资源数量每天都在减少，但是如果您有兴趣帮助为其中一个资源创建 L2 结构，请访问 API 参考。AWS CDK](#)在左窗格中，查看资源列表。名称旁边有上标 1 的资源没有官方的 L2 构造。

我能用它们支持的任何语言制作 L2 或 L3 结构吗？AWS CDK

AWS CDK 支持多种编程语言，包括 TypeScript、Python JavaScript、Java、C# 和 Go。您可以使用编译成相关语言的 AWS CDK 代码来创建您的个人 L3 结构。但是，如果要为本机构造做出贡献 AWS CDK 或创建原生 AWS CDK 构造，则必须使用 TypeScript。这是因为 TypeScript 它是唯一的本土语言 AWS CDK。其他语言的 AWS CDK 版本是通过使用名为的 AWS 库从本机 TypeScript 代码构建的[JSii](#)。

在哪里可以找到 AWS CDK 之外的现有 L3 构造？

这里有太多地方可以分享，但你可以在 S [AWS olutions Constructs 网站](#)和 [Construct Hub AWS CDK](#)部分找到许多最受欢迎的建筑。

资源

- [AWS CDK API 参考](#)
- [AWS CloudFormation 资源规范](#)
- [AWS CDK 构造文档](#)
- [AWS CDK 抽象和逃生舱口](#)
- [Leverage L2 constructs to reduce the complexity of your AWS CDK application](#) (AWS 博客文章)
- [AWS CloudFormation 自定义资源](#)
- [AWS 解决方案构造](#)
- [构造中心](#)
- [AWS CDK 示例](#) (GitHub 存储库)

文档历史记录

下表介绍了本指南的一些重要更改。如果您希望收到有关未来更新的通知，可以订阅 [RSS 源](#)。

变更	说明	日期
初次发布	—	2023 年 12 月 4 日

AWS 规范性指导词汇表

以下是 AWS 规范性指导提供的策略、指南和模式中的常用术语。若要推荐词条，请使用术语表末尾的提供反馈链接。

数字

7 R

将应用程序迁移到云中的 7 种常见迁移策略。这些策略以 Gartner 于 2011 年确定的 5 R 为基础，包括以下内容：

- **重构/重新架构**：充分利用云原生功能来提高敏捷性、性能和可扩展性，以迁移应用程序并修改其架构。这通常涉及到移植操作系统和数据库。示例：将本地 Oracle 数据库迁移到 Amazon Aurora PostgreSQL 兼容版。
- **更换平台**：将应用程序迁移到云中，并进行一定程度的优化，以利用云功能。示例：将本地 Oracle 数据库迁移到 AWS Cloud 中的 Amazon Relational Database Service (Amazon RDS) for Oracle。
- **重新购买**：转换到其他产品，通常是从传统许可转向 SaaS 模式。示例：将客户关系管理 (CRM) 系统迁移到 Salesforce.com。
- **重新托管 (直接迁移)**：将应用程序迁移到云中，无需进行任何更改即可利用云功能。示例：将本地 Oracle 数据库迁移到 AWS Cloud 中 EC2 实例上的 Oracle。
- **重新放置 (虚拟机监控器级直接迁移)**：将基础设施迁移到云中，无需购买新硬件、重写应用程序或修改现有操作。您将服务器从本地平台迁移到同一平台的云服务中。示例：将 Microsoft Hyper-V 应用程序迁移到 AWS。
- **保留 (重访)**：将应用程序保留在源环境中。其中可能包括需要进行重大重构的应用程序，并且您希望将工作推迟到以后，以及您希望保留的遗留应用程序，因为迁移它们没有商业上的理由。
- **停用**：停用或删除源环境中不再需要的应用程序。

A

ABAC

请参阅[基于属性的访问控制](#)。

抽象服务

请参阅[托管服务](#)。

ACID

请参阅[原子性、一致性、隔离性、持久性](#)。

主动-主动迁移

一种数据库迁移方法，在这种方法中，源数据库和目标数据库保持同步（通过使用双向复制工具或双写操作），两个数据库都在迁移期间处理来自连接应用程序的事务。这种方法支持小批量、可控的迁移，而不需要一次性割接。它比[主动-被动迁移](#)更灵活，但工作量更大。

主动-被动迁移

一种数据库迁移方法，在这种方法中，源数据库和目标数据库保持同步，但在将数据复制到目标数据库时，只有源数据库处理来自连接应用程序的事务。目标数据库在迁移期间不接受任何事务。

聚合函数

一种 SQL 函数，它对一组行进行操作并计算该组的单个返回值。聚合函数的示例包括 SUM 和 MAX。

AI

请参阅[人工智能](#)。

AIOps

请参阅[人工智能运营](#)。

匿名化

永久删除数据集中个人信息的过程。匿名化可以帮助保护个人隐私。匿名化数据不再被视为个人数据。

反模式

一种用于解决反复出现的问题的常用解决方案，而在这类问题中，此解决方案适得其反、无效或不如替代方案有效。

应用程序控制

一种安全方法，仅允许使用经批准的应用程序，以帮助保护系统免受恶意软件的侵害。

应用程序组合

有关组织使用的每个应用程序的详细信息的集合，包括构建和维护该应用程序的成本及其业务价值。这些信息是[产品组合发现和分析过程](#)的关键，有助于识别需要进行迁移、现代化和优化的应用程序并确定其优先级。

人工智能 (AI)

计算机科学领域致力于使用计算技术执行通常与人类相关的认知功能，例如学习、解决问题和识别模式。有关更多信息，请参阅[什么是人工智能？](#)

人工智能操作 (AIOps)

使用机器学习技术解决运营问题、减少运营事故和人为干预以及提高服务质量的过程。有关如何在 AIOps AWS 迁移策略中使用的更多信息，请参阅[操作集成指南](#)。

非对称加密

一种加密算法，使用一对密钥，一个公钥用于加密，一个私钥用于解密。您可以共享公钥，因为它不用于解密，但对私钥的访问应受到严格限制。

原子性、一致性、隔离性、持久性 (ACID)

一组软件属性，即使在出现错误、电源故障或其他问题的情况下，也能保证数据库的数据有效性和操作可靠性。

基于属性的访问权限控制 (ABAC)

根据用户属性（如部门、工作角色和团队名称）创建精细访问权限的做法。有关更多信息，请参阅 AWS Identity and Access Management (IAM) [文档](#) [AWS 中的 AB AC](#)。

权威数据来源

存储主要数据版本的位置，被认为是最可靠的信息源。您可以将数据从权威数据来源复制到其他位置，以便处理或修改数据，例如对数据进行匿名化、编辑或假名化。

可用区

中的一个不同位置 AWS 区域，不受其他可用区域故障的影响，并向同一区域中的其他可用区提供低成本、低延迟的网络连接。

AWS 云采用框架 (AWS CAF)

该框架包含指导方针和最佳实践 AWS，可帮助组织制定高效且有效的计划，以成功迁移到云端。AWS CAF 将指导分为六个重点领域，称为视角：业务、人员、治理、平台、安全和运营。业务、人员和治理角度侧重于业务技能和流程；平台、安全和运营角度侧重于技术技能和流程。例如，人

员角度针对的是负责人力资源 (HR)、人员配置职能和人员管理的利益相关者。从这个角度来看，AWS CAF 为人员发展、培训和沟通提供了指导，以帮助组织为成功采用云做好准备。有关更多信息，请参阅 [AWS CAF 网站](#) 和 [AWS CAF 白皮书](#)。

AWS 工作负载资格框架 (AWS WQF)

一种评估数据库迁移工作负载、推荐迁移策略和提供工作估算的工具。AWS WQF 包含在 AWS Schema Conversion Tool (AWS SCT) 中。它用来分析数据库架构和代码对象、应用程序代码、依赖关系和性能特征，并提供评测报告。

B

恶意机器人

一种旨在扰乱或伤害个人或组织的[机器人](#)。

BCP

请参阅[业务连续性计划](#)。

行为图

一段时间内资源行为和交互的统一交互式视图。您可以使用 Amazon Detective 的行为图来检查失败的登录尝试、可疑的 API 调用和类似的操作。有关更多信息，请参阅 Detective 文档中的[行为图中的数据](#)。

大端序系统

一个先存储最高有效字节的系统。另请参阅[字节顺序](#)。

二进制分类

一种预测二进制结果 (两个可能的类别之一) 的过程。例如，您的 ML 模型可能需要预测诸如“该电子邮件是否为垃圾邮件？”或“这个产品是书还是汽车？”之类的问题

bloom 筛选条件

一种概率性、内存高效的数据结构，用于测试元素是否为集合的成员。

蓝/绿部署

一种部署策略，您可以创建两个独立但完全相同的环境。在一个环境中运行当前应用程序版本 (蓝色)，在另一个环境中运行新应用程序版本 (绿色)。此策略可帮助您在影响最小的情况下快速回滚。

自动程序

一种通过互联网运行自动任务并模拟人类活动或交互的软件应用程序。有些机器人是有用或有益的，例如在互联网上索引信息的 Web 爬网程序。还有一些被称为恶意机器人的机器人，其目的是扰乱或伤害个人或组织。

僵尸网络

被[恶意软件](#)感染并受单方（称为僵尸网络控制者或僵尸网络操作者）控制的[僵尸网络](#)。僵尸网络是最著名的扩展机器人及其影响力的机制。

分支

代码存储库的一个包含区域。在存储库中创建的第一个分支是主分支。您可以从现有分支创建新分支，然后在新分支中开发功能或修复错误。为构建功能而创建的分支通常称为功能分支。当功能可以发布时，将功能分支合并回主分支。有关更多信息，请参阅[关于分支](#)（GitHub 文档）。

紧急（break-glass）访问

在特殊情况下，通过批准的流程，用户 AWS 账户 可以快速访问他们通常没有访问权限的内容。有关更多信息，请参阅 AWS Well-Architected Guidance 中的 [Implement break-glass procedures](#) 指示器。

棕地策略

您环境中的现有基础设施。在为系统架构采用棕地策略时，您需要围绕当前系统和基础设施的限制来设计架构。如果您正在扩展现有基础设施，则可以将棕地策略和[全新](#)策略混合。

缓冲区缓存

存储最常访问的数据的内存区域。

业务能力

企业如何创造价值（例如，销售、客户服务或营销）。微服务架构和开发决策可以由业务能力驱动。有关更多信息，请参阅[在 AWS 上运行容器化微服务](#)白皮书中的[围绕业务能力进行组织](#)部分。

业务连续性计划（BCP）

一项计划，旨在应对大规模迁移等破坏性事件对运营的潜在影响，并使企业能够快速恢复运营。

C

CAF

请参阅 [AWS 云采用框架](#)。

金丝雀部署

缓慢而渐进地向最终用户发布版本。当您确信无误后，即可部署新版本，并完全替换当前版本。

CCoE

请参阅[云卓越中心](#)。

CDC

请参阅[更改数据捕获](#)。

更改数据捕获 (CDC)

跟踪数据来源 (如数据库表) 的更改并记录有关更改的元数据的过程。您可以将 CDC 用于各种目的，例如审计或复制目标系统中的更改以保持同步。

混沌工程

故意引入故障或破坏性事件来测试系统的韧性。您可以使用 [AWS Fault Injection Service \(AWS FIS\)](#) 来执行实验，对您的 AWS 工作负载施加压力并评估其响应。

CI/CD

请参阅[持续集成和持续交付](#)。

分类

一种有助于生成预测的分类流程。分类问题的 ML 模型预测离散值。离散值始终彼此不同。例如，一个模型可能需要评估图像中是否有汽车。

客户端加密

在目标 AWS 服务 收到数据之前，对数据进行本地加密。

云卓越中心 (CCoE)

一个多学科团队，负责推动整个组织的云采用工作，包括开发云最佳实践、调动资源、制定迁移时间表、领导组织完成大规模转型。有关更多信息，请参阅 AWS Cloud 企业战略博客上的 [CCoE 帖子](#)。

云计算

通常用于远程数据存储和 IoT 设备管理的云技术。云计算通常连接到[边缘计算](#)技术。

云运营模型

在 IT 组织中，一种用于构建、完善和优化一个或多个云环境的运营模型。有关更多信息，请参阅[构建您的云运营模型](#)。

云采用阶段

组织迁移到 AWS Cloud 中时通常会经历四个阶段：

- 项目 - 出于概念验证和学习目的，开展一些与云相关的项目
- 基础 — 进行基础投资以扩大云采用率（例如，创建着陆区、定义 CCo E、建立运营模型）
- 迁移 - 迁移单个应用程序
- 重塑 - 优化产品和服务，在云中创新

Stephen Orban 在 AWS Cloud 企业战略博客的博客文章 [《云优先之旅和采用阶段》](#) 中定义了这些阶段。有关它们与 AWS 迁移策略的关系的信息，请参阅 [迁移准备指南](#)。

CMDB

请参阅 [配置管理数据库](#)。

代码存储库

通过版本控制过程存储和更新源代码和其他资产（如文档、示例和脚本）的位置。常见的云存储库包括 GitHub 或 Bitbucket Cloud。每个版本的代码都称为一个分支。在微服务结构中，每个存储库都专门用于一个功能。单个 CI/CD 管线可以使用多个存储库。

冷缓存

一种空的、填充不足或包含过时或不相关数据的缓冲区缓存。这会影响性能，因为数据库实例必须从主内存或磁盘读取，这比从缓冲区缓存读取要慢。

冷数据

很少访问的数据，且通常是历史数据。查询此类数据时，通常可以接受慢速查询。将这些数据转移到性能较低且成本更低的存储层或类别可以降低成本。

计算机视觉 (CV)

一种 [AI](#) 领域，它使用机器学习来分析和提取数字图像和视频等视觉格式中的信息。例如，Amazon SageMaker AI 为 CV 提供了图像处理算法。

配置偏移

对于工作负载而言，一种偏离预期状态的配置更改。这可能会导致工作负载变得不合规，且通常是渐进的，不是故意的。

配置管理数据库 (CMDB)

一种存储库，用于存储和管理有关数据库及其 IT 环境的信息，包括硬件和软件组件及其配置。您通常在迁移的产品组合发现和分析阶段使用来自 CMDB 的数据。

合规性包

一系列 AWS Config 规则和补救措施，您可以汇编这些规则和补救措施，以自定义您的合规性和安全性检查。您可以使用 YAML 模板将一致性包作为单个实体部署在 AWS 账户 和区域或整个组织中。有关更多信息，请参阅 AWS Config 文档中的 [一致性包](#)。

持续集成和持续交付 (CI/CD)

自动执行软件发布过程的源代码、构建、测试、暂存和生产阶段的过程。CI/CD 通常被描述为管道。CI/CD 可以帮助您实现流程自动化、提高生产力、提高代码质量和更快地交付。有关更多信息，请参阅[持续交付的优势](#)。CD 也可以表示持续部署。有关更多信息，请参阅[持续交付与持续部署](#)。

CV

请参阅[计算机视觉](#)。

D

静态数据

网络中静止的数据，例如存储中的数据。

数据分类

根据网络中数据的关键性和敏感性对其进行识别和分类的过程。它是任何网络安全风险管理策略的关键组成部分，因为它可以帮助您确定对数据的适当保护和保留控制。数据分类是 Well-Architected AWS d Framework 中安全支柱的一个组成部分。有关详细信息，请参阅[数据分类](#)。

数据漂移

生产数据与用来训练机器学习模型的数据之间的有意义差异，或者输入数据随时间推移的有意义变化。数据漂移可能降低机器学习模型预测的整体质量、准确性和公平性。

传输中数据

在网络中主动移动的数据，例如在网络资源之间移动的数据。

数据网格

一种架构框架，可提供分布式、去中心化的数据所有权以及集中式管理和治理。

数据最少化

仅收集并处理绝对必要数据的原则。在中进行数据最小化 AWS Cloud 可以降低隐私风险、成本和分析碳足迹。

数据边界

AWS 环境中的一组预防性防护措施，可帮助确保只有可信身份才能访问来自预期网络的可信资源。有关更多信息，请参阅在[上构建数据边界](#)。AWS

数据预处理

将原始数据转换为 ML 模型易于解析的格式。预处理数据可能意味着删除某些列或行，并处理缺失、不一致或重复的值。

数据溯源

在数据的整个生命周期跟踪其来源和历史的过程，例如数据如何生成、传输和存储。

数据主体

正在收集和处理其数据的人。

数据仓库

一种支持商业智能（例如分析）的数据管理系统。数据仓库通常包含大量历史数据，通常用于查询和分析。

数据库定义语言（DDL）

在数据库中创建或修改表和对象结构的语句或命令。

数据库操作语言（DML）

在数据库中修改（插入、更新和删除）信息的语句或命令。

DDL

请参阅[数据库定义语言](#)。

深度融合

组合多个深度学习模型进行预测。您可以使用深度融合来获得更准确的预测或估算预测中的不确定性。

深度学习

一个 ML 子字段使用多层神经网络来识别输入数据和感兴趣的目标变量之间的映射。

defense-in-depth

一种信息安全方法，经过深思熟虑，在整个计算机网络中分层实施一系列安全机制和控制措施，以保护网络及其中数据的机密性、完整性和可用性。当你采用这种策略时 AWS，你会在 AWS

Organizations 结构的不同层面添加多个控件来帮助保护资源。例如，一种 defense-in-depth 方法可以结合多因素身份验证、网络分段和加密。

委派管理员

在中 AWS Organizations，兼容的服务可以注册 AWS 成员帐户来管理组织的帐户并管理该服务的权限。此帐户被称为该服务的委托管理员。有关更多信息和兼容服务列表，请参阅 AWS Organizations 文档中[使用 AWS Organizations 的服务](#)。

部署

使应用程序、新功能或代码修复在目标环境中可用的过程。部署涉及在代码库中实现更改，然后在应用程序的环境中构建和运行该代码库。

开发环境

请参阅[环境](#)。

侦测性控制

一种安全控制，在事件发生后进行检测、记录日志和发出提醒。这些控制是第二道防线，提醒您注意绕过现有预防性控制的安全事件。有关更多信息，请参阅在 AWS 上实施安全控制中的[侦测性控制](#)。

开发价值流映射 (DVSM)

用于识别对软件开发生命周期中的速度和质量产生不利影响的限制因素并确定其优先级的流程。DVSM 扩展了最初为精益生产实践设计的价值流映射流程。其重点关注在软件开发过程中创造和转移价值所需的步骤和团队。

数字孪生

真实世界系统的虚拟再现，如建筑物、工厂、工业设备或生产线。数字孪生支持预测性维护、远程监控和生产优化。

维度表

[星型架构](#)中的一种较小的表，其中包含事实表中定量数据的数据属性。维度表属性通常是文本字段或行为类似于文本的离散数字。这些属性通常用于查询约束、筛选和结果集标注。

灾难

阻止工作负载或系统在其主要部署位置实现其业务目标的事件。这些事件可能是自然灾害、技术故障或人为操作的结果，例如无意的配置错误或恶意软件攻击。

灾难恢复 (DR)

您用来最大程度地减少由[灾难](#)造成的停机时间和数据丢失的策略和流程。有关更多信息，请参阅 Well-Architected Framework AWS work 中的“[工作负载灾难恢复：云端 AWS 恢复](#)”。

DML

请参阅[数据库操作语言](#)。

领域驱动设计

一种开发复杂软件系统的方法，通过将其组件连接到每个组件所服务的不断发展的领域或核心业务目标。Eric Evans 在其著作[领域驱动设计：软件核心复杂性应对之道](#) (Boston: Addison-Wesley Professional, 2003) 中介绍了这一概念。有关如何将领域驱动设计与 strangler fig 模式结合使用的信息，请参阅[使用容器和 Amazon API Gateway 逐步将原有的 Microsoft ASP.NET \(ASMX \) Web 服务现代化](#)。

DR

请参阅[灾难恢复](#)。

偏差检测

跟踪与基准配置的偏差。例如，您可以使用 AWS CloudFormation 来[检测系统资源中的偏差](#)，也可以使用 AWS Control Tower 来[检测着陆区中可能影响监管要求合规性的变化](#)。

DVSM

请参阅[开发价值流映射](#)。

E

EDA

请参阅[探索性数据分析](#)。

EDI

请参阅[电子数据交换](#)。

边缘计算

该技术可提高位于 IoT 网络边缘的智能设备的计算能力。与[云计算](#)比较时，边缘计算可以减少通信延迟并缩短响应时间。

电子数据交换 (EDI)

组织之间业务文件的自动交换。有关更多信息，请参阅[什么是电子数据交换](#)。

加密

一种将人类可读的纯文本数据转换为加密文字的计算流程。

加密密钥

由加密算法生成的随机位的加密字符串。密钥的长度可能有所不同，而且每个密钥都设计为不可预测且唯一。

字节顺序

字节在计算机内存中的存储顺序。大端序系统先存储最高有效字节。小端序系统先存储最低有效字节。

端点

请参阅[服务端点](#)。

端点服务

一种可以在虚拟私有云 (VPC) 中托管，与其他用户共享的服务。您可以使用其他 AWS 账户 或 AWS Identity and Access Management (IAM) 委托人创建终端节点服务，AWS PrivateLink 并向其授予权限。这些账户或主体可通过创建接口 VPC 端点来私密地连接到您的端点服务。有关更多信息，请参阅 Amazon Virtual Private Cloud (Amazon VPC) 文档中的[创建端点服务](#)。

企业资源规划 (ERP)

一种自动化和管理企业关键业务流程 (例如会计、[MES](#) 和项目管理) 的系统。

信封加密

用另一个加密密钥对加密密钥进行加密的过程。有关更多信息，请参阅 AWS Key Management Service (AWS KMS) 文档中的[信封加密](#)。

环境

正在运行的应用程序的实例。以下是云计算中常见的环境类型：

- 开发环境 — 正在运行的应用程序的实例，只有负责维护应用程序的核心团队才能使用。开发环境用于测试更改，然后再将其提升到上层环境。这类环境有时称为测试环境。
- 下层环境 — 应用程序的所有开发环境，比如用于初始构建和测试的环境。

- 生产环境 — 最终用户可以访问的正在运行的应用程序的实例。在 CI/CD 管道中，生产环境是最后一个部署环境。
- 上层环境 — 除核心开发团队以外的用户可以访问的所有环境。这可能包括生产环境、预生产环境和用户验收测试环境。

epic

在敏捷方法学中，有助于组织工作和确定优先级的功能类别。epics 提供了对需求和实施任务的总体描述。例如，AWS CAF 安全史诗包括身份和访问管理、侦探控制、基础设施安全、数据保护和事件响应。有关 AWS 迁移策略中 epics 的更多信息，请参阅[计划实施指南](#)。

ERP

请参阅[企业资源规划](#)。

探索性数据分析 (EDA)

分析数据集以了解其主要特征的过程。您收集或汇总数据，并进行初步调查，以发现模式、检测异常并检查假定情况。EDA 通过计算汇总统计数据 and 创建数据可视化得以执行。

F

事实表

[星型架构](#)中的中心表。它存储有关业务运营的定量数据。通常，事实表包含两种类型的列：包含度量的列和包含维度表外键的列。

快速失效机制

一种使用频繁且增量式的测试来缩短开发生命周期的理念。这是敏捷方法的关键部分。

故障隔离边界

在中 AWS Cloud，诸如可用区 AWS 区域、控制平面或数据平面之类的边界，它限制了故障的影响并有助于提高工作负载的弹性。有关更多信息，请参阅[AWS 故障隔离边界](#)。

功能分支

请参阅[分支](#)。

特征

您用来进行预测的输入数据。例如，在制造环境中，特征可能是定期从生产线捕获的图像。

特征重要性

特征对于模型预测的重要性。这通常表示为数值分数，可以通过各种技术进行计算，例如 Shapley 加法解释 (SHAP) 和积分梯度。有关更多信息，请参阅使用[机器学习模型的可解释性 AWS](#)。

功能转换

为 ML 流程优化数据，包括使用其他来源丰富数据、扩展值或从单个数据字段中提取多组信息。这使得 ML 模型能从数据中获益。例如，如果您将“2021-05-27 00:15:37”日期分解为“2021”、“五月”、“星期四”和“15”，则可以帮助学习与不同数据成分相关的算法学习精细模式。

少样本提示

在要求 [LLM](#) 执行类似任务之前，先向其提供少量示例，以演示任务和预期输出。此技术是上下文内学习的一种应用，其中模型可以从提示中嵌入的示例 (样本) 中学习。对于需要特定格式、推理或领域知识的任务，少样本提示可能非常有效。另请参阅[零样本提示](#)。

FGAC

请参阅[精细访问控制](#)。

精细访问控制 (FGAC)

使用多个条件允许或拒绝访问请求。

快闪迁移

一种数据库迁移方法，通过[更改数据捕获](#)使用连续数据复制，在极短的时间内迁移数据，而非使用分阶段方法。目标是将停机时间降至最低。

FM

请参阅[基础模型](#)。

基础模型 (FM)

一个大型深度学习神经网络，一直在广义和未标记数据的大量数据集上进行训练。FMs 能够执行各种各样的一般任务，例如理解语言、生成文本和图像以及用自然语言进行对话。有关更多信息，请参阅[什么是基础模型](#)。

G

生成式人工智能

[AI](#) 模型的一个子集，这些模型已经过大量数据训练，可以使用简单的文本提示来创建新的内容和构件，例如图像、视频、文本和音频。有关更多信息，请参阅[什么是生成式人工智能](#)。

地理阻止

请参阅[地理限制](#)。

地理限制 (地理阻止)

在 Amazon 中 CloudFront，一种阻止特定国家/地区的用户访问内容分发的选项。您可以使用允许列表或阻止列表来指定已批准和已禁止的国家/地区。有关更多信息，请参阅 CloudFront 文档[中的限制内容的地理分布](#)。

GitFlow 工作流程

一种方法，在这种方法中，下层和上层环境在源代码存储库中使用不同的分支。Gitflow 工作流程被认为是传统的工作流程，而[基于中继的工作流程](#)则是现代的、首选的方法。

黄金映像

系统或软件的快照，用作部署该系统或软件的新实例的模板。例如，在制造业中，黄金映像可用于在多个设备上预调配软件，并有助于提高设备制造操作的速度、可扩展性和生产效率。

全新策略

在新环境中缺少现有基础设施。在对系统架构采用全新策略时，您可以选择所有新技术，而不受对现有基础设施 (也称为[棕地](#)) 兼容性的限制。如果您正在扩展现有基础设施，则可以将棕地策略和全新策略混合。

防护机制

帮助管理各组织单位的资源、策略和合规性的高级规则 (OUs)。预防性防护机制会执行策略以确保符合合规性标准。它们是使用服务控制策略和 IAM 权限边界实现的。侦测性护栏会检测策略违规和合规性问题，并生成提醒以进行修复。它们通过使用 AWS Config、Amazon、AWS Security Hub CSPM GuardDuty AWS Trusted Advisor、Amazon Inspector 和自定义 AWS Lambda 支票来实现。

H

HA

请参阅[高可用性](#)。

异构数据库迁移

将源数据库迁移到使用不同数据库引擎的目标数据库 (例如，从 Oracle 迁移到 Amazon Aurora)。异构迁移通常是重新架构工作的一部分，而转换架构可能是一项复杂的任务。[AWS 提供了 AWS SCT](#) 来帮助实现架构转换。

高可用性 (HA)

在遇到挑战或灾难时，工作负载无需干预即可连续运行的能力。HA 系统旨在自动进行故障转移、持续提供良好性能，并以最小的性能影响处理不同负载和故障。

历史数据库现代化

一种用于实现运营技术 (OT) 系统现代化和升级以更好满足制造业需求的方法。历史数据库是一种用于收集和存储工厂中各种来源数据的数据库。

保留数据

从用于训练[机器学习](#)模型的数据集中保留的一部分标注的历史数据。通过将模型预测与保留数据进行比较，您可以使用保留数据来评估模型性能。

同构数据库迁移

将源数据库迁移到共享同一数据库引擎的目标数据库 (例如，从 Microsoft SQL Server 迁移到 Amazon RDS for SQL Server)。同构迁移通常是更换主机或更换平台工作的一部分。您可以使用本机数据库实用程序来迁移架构。

热数据

经常访问的数据，例如实时数据或近期的转化数据。这些数据通常需要高性能存储层或存储类别才能提供快速的查询响应。

修补程序

针对生产环境中关键问题的紧急修复。由于其紧迫性，修补程序通常是在典型的 DevOps 发布工作流程之外进行的。

hypercure 周期

割接之后，迁移团队立即管理和监控云中迁移的应用程序以解决任何问题的时间段。通常，这个周期持续 1-4 天。在 hypercure 周期结束时，迁移团队通常会将应用程序的责任移交给云运营团队。

我

laC

请参阅[基础设施即代码](#)。

基于身份的策略

附加到一个或多个 IAM 委托人的策略，用于定义他们在 AWS Cloud 环境中的权限。

空闲应用程序

90 天内平均 CPU 和内存使用率在 5% 到 20% 之间的应用程序。在迁移项目中，通常会停用这些应用程序或将其保留在本地。

IloT

请参阅[工业物联网](#)。

不可变基础设施

一种模型，可为生产工作负载部署新的基础设施，而不是更新、修补或修改现有基础设施。不可变基础设施本质上比[可变基础设施](#)更一致、更可靠、更可预测。有关更多信息，请参阅 AWS Well-Architected Framework 中的[使用不可变基础设施进行部署](#)最佳实践。

入站 (入口) VPC

在 AWS 多账户架构中，一种接受、检查和路由来自应用程序外部的网络连接的 VPC。[AWS 安全参考架构](#)建议设置您的网络帐户，包括入站、出站和检查，VPCs 以保护您的应用程序与更广泛的互联网之间的双向接口。

增量迁移

一种割接策略，在这种策略中，您可以将应用程序分成小部分进行迁移，而不是一次性完整割接。例如，您最初可能只将几个微服务或用户迁移到新系统。在确认一切正常后，您可以逐步迁移其他微服务或用户，直到停用遗留系统。这种策略降低了大规模迁移带来的风险。

工业 4.0

该术语由 [Klaus Schwab](#) 在 2016 年提出，指的是通过连接、实时数据、自动化、分析和 AI/ML 的进步来实现制造流程的现代化。

基础设施

应用程序环境中包含的所有资源和资产。

基础设施即代码 (IaC)

通过一组配置文件预调配和管理应用程序基础设施的过程。IaC 旨在帮助您集中管理基础设施、实现资源标准化和快速扩展，使新环境具有可重复性、可靠性和一致性。

工业物联网 (IloT)

在工业领域使用联网的传感器和设备，例如制造业、能源、汽车、医疗保健、生命科学和农业。有关更多信息，请参阅[制定工业物联网 \(IloT\) 数字化转型战略](#)。

检查 VPC

在 AWS 多账户架构中，一种集中式 VPC，用于管理对 VPCs（相同或不同 AWS 区域）、互联网和本地网络之间的网络流量的检查。[AWS 安全参考架构](#)建议设置您的网络帐户，包括入站、出站和检查，VPCs 以保护您的应用程序与更广泛的互联网之间的双向接口。

物联网 (IoT)

由带有嵌入式传感器或处理器的连接物理对象组成的网络，这些传感器或处理器通过互联网或本地通信网络与其他设备和系统进行通信。有关更多信息，请参阅[什么是 IoT ?](#)

可解释性

它是机器学习模型的一种特征，描述了人类可以理解模型的预测如何取决于其输入的程度。有关更多信息，请参阅使用[机器学习模型的可解释性 AWS](#)。

物联网

请参阅[物联网](#)。

IT 信息库 (ITIL)

提供 IT 服务并使这些服务符合业务要求的一套最佳实践。ITIL 是 ITSM 的基础。

IT 服务管理 (ITSM)

为组织设计、实施、管理和支持 IT 服务的相关活动。有关将云运营与 ITSM 工具集成的信息，请参阅[运营集成指南](#)。

ITIL

请参阅[IT 信息库](#)。

ITSM

请参阅[IT 服务管理](#)。

L

基于标签的访问控制 (LBAC)

强制访问控制 (MAC) 的一种实施方式，其中明确为用户和数据本身分配了安全标签值。用户安全标签和数据安全标签之间的交集决定了用户可以看到哪些行和列。

登录区

landing zone 是一个架构精良的多账户 AWS 环境，具有可扩展性和安全性。这是一个起点，您的组织可以从这里放心地在安全和基础设施环境中快速启动和部署工作负载和应用程序。有关登录区的更多信息，请参阅[设置安全且可扩展的多账户 AWS 环境](#)。

大语言模型 (LLM)

一种基于大量数据进行预训练的深度学习 [AI](#) 模型。LLM 可以执行多项任务，例如回答问题、总结文档、将文本翻译成其他语言以及完成句子。有关更多信息，请参阅[什么是 LLMs](#)。

大规模迁移

迁移 300 台或更多服务器。

LBAC

请参阅[基于标签的访问控制](#)。

最低权限

授予执行任务所需的最低权限的最佳安全实践。有关更多信息，请参阅 IAM 文档中的[应用最低权限许可](#)。

直接迁移

请参阅 [7 R](#)。

小端序系统

一个先存储最低有效字节的系统。另请参阅[字节顺序](#)。

LLM

请参阅[大型语言模型](#)。

下层环境

请参阅[环境](#)。

M

机器学习 (ML)

一种使用算法和技术进行模式识别和学习的人工智能。ML 对记录的数据 (例如物联网 (IoT) 数据) 进行分析和学习，以生成基于模式的统计模型。有关更多信息，请参阅[机器学习](#)。

主分支

请参阅[分支](#)。

恶意软件

旨在危害计算机安全或隐私的软件。恶意软件可能会破坏计算机系统、泄露敏感信息或获得未经授权的访问权限。恶意软件的示例包括病毒、蠕虫、勒索软件、木马、间谍软件和键盘记录器。

托管式服务

AWS 服务 它 AWS 运行基础设施层、操作系统和平台，您可以访问端点来存储和检索数据。Amazon Simple Storage Service (Amazon S3) 和 Amazon DynamoDB 就是托管服务的示例。这些服务也称为抽象服务。

制造执行系统 (MES)

一种软件系统，用于跟踪、监控、记录和控制将原材料转化为成品的生产过程。

MAP

请参阅[迁移加速计划](#)。

机制

一个完整的过程，您可以在其中创建工具，推动工具的采用，然后检查结果以进行调整。机制是一种在运作过程中自我强化和改善的循环。有关更多信息，请参阅在 Well-Architect AWS ed 框架中[构建机制](#)。

成员账户

AWS 账户 除属于组织中的管理账户之外的所有账户 AWS Organizations。一个账户一次只能是一个组织的成员。

MES

请参阅[制造执行系统](#)。

消息队列遥测传输 (MQTT)

一种基于发布/订阅模式的轻量级 machine-to-machine (M2M) 通信协议，适用于资源受限的物联网设备。

微服务

一种小型的独立服务，通过明确的定义进行通信 APIs ，通常由小型的独立团队拥有。例如，保险系统可能包括映射到业务能力（如销售或营销）或子域（如购买、理赔或分析）的微服务。微服务

的好处包括敏捷、灵活扩展、易于部署、可重复使用的代码和恢复能力。有关更多信息，请参阅[使用 AWS 无服务器服务集成微服务](#)。

微服务架构

一种使用独立组件构建应用程序的方法，这些组件将每个应用程序进程作为微服务运行。这些微服务使用轻量级通过定义明确的接口进行通信。APIs 该架构中的每个微服务都可以更新、部署和扩展，以满足对应用程序特定功能的需求。有关更多信息，请参阅[在上实现微服务](#)。AWS

迁移加速计划 (MAP)

AWS 该计划提供咨询支持、培训和服务，以帮助组织为迁移到云奠定坚实的运营基础，并帮助抵消迁移的初始成本。MAP 提供了一种以系统的方式执行遗留迁移的迁移方法，以及一套用于自动执行和加速常见迁移场景的工具。

大规模迁移

将大部分应用程序组合分波迁移到云中的过程，在每一波中以更快的速度迁移更多应用程序。本阶段使用从早期阶段获得的最佳实践和经验教训，实施由团队、工具和流程组成的迁移工厂，通过自动化和敏捷交付简化工作负载的迁移。这是[AWS 迁移策略](#)的第三阶段。

迁移工厂

跨职能团队，通过自动化、敏捷的方法简化工作负载迁移。迁移工厂团队通常包括运营、业务分析师和所有者、迁移工程师、开发人员和冲刺 DevOps 领域的专业人员。20% 到 50% 的企业应用程序组合由可通过工厂方法优化的重复模式组成。有关更多信息，请参阅本内容集中[有关迁移工厂的讨论](#)和[云迁移工厂指南](#)。

迁移元数据

有关完成迁移所需的应用程序和服务器器的信息。每种迁移模式都需要一套不同的迁移元数据。迁移元数据的示例包括目标子网、安全组和 AWS 账户。

迁移模式

一种可重复的迁移任务，详细列出了迁移策略、迁移目标以及所使用的迁移应用程序或服务。示例：使用 AWS 应用程序迁移服务重新托管向 Amazon EC2 的迁移。

迁移组合评测 (MPA)

一种在线工具，提供了用于验证迁移到 AWS Cloud 的业务案例的信息。MPA 提供了详细的组合评测（服务器规模调整、定价、TCO 比较、迁移成本分析）以及迁移计划（应用程序数据分析和数据收集、应用程序分组、迁移优先级排序和波次规划）。所有 AWS 顾问和 APN 合作伙伴顾问均可免费使用[MPA 工具](#)（需要登录）。

迁移准备情况评测 (MRA)

使用 AWS CAF 深入了解组织的云就绪状态、确定优势和劣势以及制定行动计划以缩小已发现差距的过程。有关更多信息，请参阅[迁移准备指南](#)。MRA 是 [AWS 迁移策略](#) 的第一阶段。

迁移策略

将工作负载迁移到 AWS Cloud 的方法。有关更多信息，请参见术语表中的 [7 R](#) 词条，以及[动员您的组织以加快大规模迁移](#)。

ML

请参阅[机器学习](#)。

现代化

将过时的（原有的或单体）应用程序及其基础设施转变为云中敏捷、弹性和高度可用的系统，以降低成本、提高效率和利用创新。有关更多信息，请参阅[在 AWS Cloud 中实现应用程序现代化的策略](#)。

现代化准备情况评估

一种评估方式，有助于确定组织应用程序的现代化准备情况；确定收益、风险和依赖关系；确定组织能够在多大程度上支持这些应用程序的未来状态。评估结果是目标架构的蓝图、详细说明现代化进程发展阶段和里程碑的路线图以及解决已发现差距的行动计划。有关更多信息，请参阅[在 AWS Cloud 中评估应用程序的现代化准备情况](#)。

单体应用程序 (单体式)

作为具有紧密耦合进程的单个服务运行的应用程序。单体应用程序有几个缺点。如果某个应用程序功能的需求激增，则必须扩展整个架构。随着代码库的增长，添加或改进单体应用程序的功能也会变得更加复杂。若要解决这些问题，可以使用微服务架构。有关更多信息，请参阅[将单体分解为微服务](#)。

MPA

请参阅[迁移组合评测](#)。

MQTT

请参阅[消息队列遥测传输](#)。

多分类器

一种帮助为多个类别生成预测（预测两个以上结果之一）的过程。例如，ML 模型可能会询问“这个产品是书、汽车还是手机？”或“此客户最感兴趣什么类别的产品？”

可变基础设施

一种用于更新和修改生产工作负载的现有基础设施的模型。为了提高一致性、可靠性和可预测性，Well-Architect AWS ed Framework 建议使用[不可变基础设施](#)作为最佳实践。

O

OAC

请参阅[来源访问控制](#)。

OAI

请参阅[来源访问身份](#)。

OCM

请参阅[组织变革管理](#)。

离线迁移

一种迁移方法，在这种方法中，源工作负载会在迁移过程中停止运行。这种方法会延长停机时间，通常用于小型非关键工作负载。

OI

请参阅[运营集成](#)。

OLA

请参阅[运营级别协议](#)。

在线迁移

一种迁移方法，在这种方法中，源工作负载无需离线即可复制到目标系统。在迁移过程中，连接工作负载的应用程序可以继续运行。这种方法的停机时间为零或最短，通常用于关键生产工作负载。

OPC-UA

请参阅[开放流程通信 – 统一架构](#)。

开放流程通信 – 统一架构 (OPC-UA)

一种用于工业自动化的 machine-to-machine (M2M) 通信协议。OPC-UA 提供了一个包含数据加密、身份验证和授权方案的互操作性标准。

运营级别协议 (OLA)

一项协议，阐明了 IT 职能部门承诺相互交付的内容，以支持服务水平协议 (SLA)。

运营准备情况审查 (ORR)

一份问题核对清单和关联的最佳实践，可帮助您了解、评估、预防或缩小事件和可能的故障的范围。有关更多信息，请参阅 [AWS Well-Architected Framework 中的运营准备情况审查 \(ORR \)](#)。

运营技术 (OT)

与物理环境配合使用以控制工业运营、设备和基础设施的硬件和软件系统。在制造业中，OT 和信息技术 (IT) 系统的集成是[工业 4.0](#) 转型的关键重点。

运营整合 (OI)

在云中实现运营现代化的过程，包括就绪计划、自动化和集成。有关更多信息，请参阅[运营整合指南](#)。

组织跟踪

由 AWS CloudTrail 此创建的跟踪记录组织 AWS 账户 中所有人的所有事件 AWS Organizations。该跟踪是在每个 AWS 账户 中创建的，属于组织的一部分，并跟踪每个账户的活动。有关更多信息，请参阅 CloudTrail 文档中的[为组织创建跟踪](#)。

组织变革管理 (OCM)

一个从人员、文化和领导力角度管理重大、颠覆性业务转型的框架。OCM 通过加快变革采用、解决过渡问题以及推动文化和组织变革，帮助组织为新系统和战略做好准备和过渡。在 AWS 迁移策略中，该框架被称为人员加速，因为云采用项目需要变更的速度。有关更多信息，请参阅 [OCM 指南](#)。

来源访问控制 (OAC)

在中 CloudFront，一个增强的选项，用于限制访问以保护您的亚马逊简单存储服务 (Amazon S3) 内容。OAC 全部支持所有 S3 存储桶 AWS 区域、使用 AWS KMS (SSE-KMS) 进行服务器端加密，以及对 S3 存储桶的动态PUT和DELETE请求。

来源访问身份 (OAI)

在中 CloudFront，一个用于限制访问权限以保护您的 Amazon S3 内容的选项。当您使用 OAI 时，CloudFront 会创建一个 Amazon S3 可以对其进行身份验证的委托人。经过身份验证的委托人只能通过特定 CloudFront 分配访问 S3 存储桶中的内容。另请参阅 [OAC](#)，其中提供了更精细和增强的访问控制。

ORR

请参阅[运营准备情况审查](#)。

OT

请参阅[运营技术](#)。

出站 (出口) VPC

在 AWS 多账户架构中，一种处理从应用程序内部启动的网络连接的 VPC。[AWS 安全参考架构](#)建议设置您的网络帐户，包括入站、出站和检查，VPCs 以保护您的应用程序与更广泛的互联网之间的双向接口。

P

权限边界

附加到 IAM 主体的 IAM 管理策略，用于设置用户或角色可以拥有的最大权限。有关更多信息，请参阅 IAM 文档中的[权限边界](#)。

个人身份信息 (PII)

直接查看其他相关数据或与之配对时可用于合理推断个人身份的信息。PII 的示例包括姓名、地址和联系信息。

PII

请参阅[个人身份信息](#)。

playbook

一套预定义的步骤，用于捕获与迁移相关的工作，例如在云中交付核心运营功能。playbook 可以采用脚本、自动化运行手册的形式，也可以是操作现代化环境所需的流程或步骤的摘要。

PLC

请参阅[可编程逻辑控制器](#)。

PLM

请参阅[产品生命周期管理](#)。

policy

一个对象，可以定义权限（请参阅[基于身份的策略](#)）、指定访问条件（请参阅[基于资源的策略](#)）或定义 AWS Organizations 的组织中所有账户的最大权限（请参阅[服务控制策略](#)）。

多语言持久性

根据数据访问模式和其他要求，独立选择微服务的数据存储技术。如果您的微服务采用相同的数据存储技术，它们可能会遇到实现难题或性能不佳。如果微服务使用最适合其需求的数据存储，则可以更轻松实现微服务，并获得更好的性能和可扩展性。

组合评测

一个发现、分析和确定应用程序组合优先级以规划迁移的过程。有关更多信息，请参阅[评估迁移准备情况](#)。

谓词

返回 true 或 false 的查询条件，通常位于 WHERE 子句中。

谓词下推

一种数据库查询优化技术，可在传输之前筛选查询中的数据。这将减少从关系数据库检索和处理的数据量，并提高查询性能。

预防性控制

一种安全控制，旨在防止事件发生。这些控制是第一道防线，帮助防止未经授权的访问或对网络的意外更改。有关更多信息，请参阅在 AWS 上实施安全控制中的[预防性控制](#)。

主体

中 AWS 可以执行操作和访问资源的实体。此实体通常是 IAM 角色的根用户或用户。AWS 账户有关更多信息，请参阅 IAM 文档中的[角色术语和概念](#)中的主体。

隐私设计

一种在整个开发过程中都考虑隐私的系统工程方法。

私有托管区

一个容器，其中包含有关您希望 Amazon Route 53 如何响应针对一个或多个 VPCs 域名及其子域名的 DNS 查询的信息。有关更多信息，请参阅 Route 53 文档中的[私有托管区的使用](#)。

主动控制

一种[安全控制](#)，旨在防止部署不合规资源。这些控制会在资源预置之前对其进行扫描。如果资源与控制不兼容，则不会预置它。有关更多信息，请参阅 AWS Control Tower 文档中的[控制参考指南](#)，并参见在上实施安全[控制中的主动控制](#) AWS。

产品生命周期管理 (PLM)

对产品在其整个生命周期内的数据和流程的管理，从设计、开发和发布，到增长和成熟，再到衰退和淘汰。

生产环境

请参阅[环境](#)。

可编程逻辑控制器 (PLC)

在制造业中，一种高度可靠、适应性强的计算机，用于监控机器并实现制造过程自动化。

提示串接

使用一个 [LLM](#) 提示的输出作为下一个提示的输入，以生成更好的响应。该技术用于将复杂的任务分解为子任务，或者迭代地完善或扩展初步响应。它有助于提高模型响应的准确性和相关性，并允许获得更精细的个性化结果。

假名化

用占位符值替换数据集中个人标识符的过程。假名化可以帮助保护个人隐私。假名化数据仍被视为个人数据。

publish/subscribe (pub/sub)

一种支持微服务间异步通信的模式，可提高可扩展性和响应能力。例如，在基于微服务的 [MES](#) 中，微服务可以将事件消息发布到其他微服务可以订阅的频道。系统可以在不更改发布服务的情况下添加新的微服务。

Q

查询计划

一系列用于访问 SQL 关系数据库系统中的数据的步骤，类似于指令。

查询计划回归

当数据库服务优化程序选择的最佳计划不如数据库环境发生特定变化之前时。这可能是由统计数据、约束、环境设置、查询参数绑定更改和数据库引擎更新造成的。

R

RACI 矩阵

请参阅[责任、问责、咨询和知情 \(RACI \)](#)。

RAG

请参阅[检索增强生成](#)。

勒索软件

一种恶意软件，旨在阻止对计算机系统或数据的访问，直到付款为止。

RASCI 矩阵

请参阅[责任、问责、咨询和知情 \(RACI \)](#)。

RCAC

请参阅[行列访问控制](#)。

只读副本

用于只读目的的数据库副本。您可以将查询路由到只读副本，以减轻主数据库的负载。

重新架构

请参阅 [7 R](#)。

恢复点目标 (RPO)

自上一个数据恢复点以来可接受的最长时间。这决定了从上一个恢复点到服务中断之间可接受的数据丢失情况。

恢复时间目标 (RTO)

服务中断和服务恢复之间可接受的最大延迟。

重构

请参阅 [7 R](#)。

Region

地理区域内的 AWS 资源集合。每一个 AWS 区域 都相互隔离，彼此独立，以提供容错、稳定性和弹性。有关更多信息，请参阅[指定您的账户可以使用的 AWS 区域](#)。

回归

一种预测数值的 ML 技术。例如，要解决“这套房子的售价是多少？”的问题 ML 模型可以使用线性回归模型，根据房屋的已知事实（如建筑面积）来预测房屋的销售价格。

重新托管

请参阅 [7 R](#)。

版本

在部署过程中，推动生产环境变更的行为。

重新放置

请参阅 [7 R](#)。

更换平台

请参阅 [7 R](#)。

重新购买

请参阅 [7 R](#)。

韧性

应用程序抵御中断或从中断中恢复的能力。在 AWS Cloud 中规划韧性时，[高可用性](#)和[灾难恢复](#)是常见的考虑因素。有关更多信息，请参阅 [AWS Cloud 韧性](#)。

基于资源的策略

一种附加到资源的策略，例如 AmazonS3 存储桶、端点或加密密钥。此类策略指定了允许哪些主体访问、支持的操作以及必须满足的任何其他条件。

责任、问责、咨询和知情 (RACI) 矩阵

定义参与迁移活动和云运营的所有各方的角色和责任的矩阵。矩阵名称源自矩阵中定义的责任类型：负责 (R)、问责 (A)、咨询 (C) 和知情 (I)。支持 (S) 类型是可选的。如果包括支持，则该矩阵称为 RASCI 矩阵，如果将其排除在外，则称为 RACI 矩阵。

响应性控制

一种安全控制，旨在推动对不良事件或偏离安全基线的情况进行修复。有关更多信息，请参阅在 AWS 上实施安全控制中的[响应性控制](#)。

保留

请参阅 [7 R](#)。

停用

请参阅 [7 R](#)。

检索增强生成 (RAG)

一种[生成式人工智能](#)技术，其中 [LLM](#) 在生成响应之前引用其训练数据来源之外的权威数据来源。例如，RAG 模型可以对组织的知识库或自定义数据执行语义搜索。有关更多信息，请参阅[什么是 RAG](#)。

轮换

定期更新[密钥](#)以使攻击者更难访问凭证的过程。

行列访问控制 (RCAC)

使用已定义访问规则的基本、灵活的 SQL 表达式。RCAC 由行权限和列掩码组成。

RPO

请参阅[恢复点目标](#)。

RTO

请参阅[恢复时间目标](#)。

运行手册

执行特定任务所需的一套手动或自动程序。它们通常是为了简化重复性操作或高错误率的程序而设计的。

S

SAML 2.0

许多身份提供商 (IdPs) 使用的开放标准。此功能支持联合单点登录 (SSO)，因此用户无需在 IAM 中为组织中的所有人创建用户即可登录 AWS 管理控制台 或调用 AWS API 操作。有关基于 SAML 2.0 的联合身份验证的更多信息，请参阅 IAM 文档中的[关于基于 SAML 2.0 的联合身份验证](#)。

SCADA

请参阅[监督控制和数据采集](#)。

SCP

请参阅[服务控制策略](#)。

机密密钥

在中 AWS Secrets Manager，您以加密形式存储的机密或受限信息，例如密码或用户凭证。它由密钥值及其元数据组成。密钥值可以是二进制、单个字符串或多个字符串。有关更多信息，请参阅 Secrets Manager 文档中的[什么是 Amazon Secrets Manager 密钥？](#)。

安全设计

一种在整个开发过程中都考虑安全的系统工程方法。

安全控制

一种技术或管理防护机制，可防止、检测或降低威胁行为体利用安全漏洞的能力。安全控制有以下四种类型：[预防性](#)、[检测性](#)、[响应性](#)和[主动性](#)。

安全固化

缩小攻击面，使其更能抵御攻击的过程。这可能包括删除不再需要的资源、实施授予最低权限的最佳安全实践或停用配置文件中不必要的功能等操作。

安全信息和事件管理 (SIEM) 系统

结合了安全信息管理 (SIM) 和安全事件管理 (SEM) 系统的工具和服务。SIEM 系统会收集、监控和分析来自服务器、网络、设备和其他来源的数据，以检测威胁和安全漏洞，并生成警报。

安全响应自动化

一种预定义的程序化操作，旨在自动响应或修复安全事件。这些自动化可作为[侦探或响应式](#)安全控制措施，帮助您实施 AWS 安全最佳实践。自动响应操作的示例包括修改 VPC 安全组、修补 Amazon EC2 实例或轮换凭证。

服务器端加密

由接收数据的人在目的地对数据 AWS 服务 进行加密。

服务控制策略 (SCP)

一种策略，用于集中控制组织中所有账户的权限 AWS Organizations。SCPs 定义防护措施或限制管理员可以委托给用户或角色的操作。您可以使用 SCPs 允许列表或拒绝列表来指定允许或禁止哪些服务或操作。有关更多信息，请参阅 AWS Organizations 文档中的[服务控制策略](#)。

服务端点

的入口点的 URL AWS 服务。您可以使用端点，通过编程方式连接到目标服务。有关更多信息，请参阅 AWS 一般参考 中的[AWS 服务 端点](#)。

服务水平协议 (SLA)

一份协议，阐明了 IT 团队承诺向客户交付的内容，比如服务正常运行时间和性能。

服务水平指示器 (SLI)

对服务性能方面的衡量，例如错误率、可用性或吞吐量。

服务水平目标 (SLO)

代表服务运行状况的目标指标，由[服务水平指示器](#)衡量。

责任共担模式

描述您在云安全与合规方面共同承担 AWS 的责任的模型。AWS 负责云的安全，而您则负责云中的安全。有关更多信息，请参阅[责任共担模式](#)。

SIEM

请参阅[安全信息和事件管理系统](#)。

单点故障 (SPOF)

应用程序的单个关键组件出现故障，可能会中断系统。

SLA

请参阅[服务水平协议](#)。

SLI

请参阅[服务水平指示器](#)。

SLO

请参阅[服务水平目标](#)。

split-and-seed 模型

一种扩展和加速现代化项目的模式。随着新功能和产品发布的定义，核心团队会拆分以创建新的产品团队。这有助于扩展组织的能力和服务，提高开发人员的工作效率，支持快速创新。有关更多信息，请参阅[在 AWS Cloud 中实现应用程序现代化的分阶段方法](#)。

SPOF

请参阅[单点故障](#)。

星型架构

一种数据库组织结构，它使用一个大型事实表来存储事务数据或测量数据，并使用一个或多个较小的维度表来存储数据属性。此结构专为在[数据仓库](#)中使用或用于商业智能目的而设计。

strangler fig 模式

一种通过逐步重写和替换系统功能直至可以停用原有的系统来实现单体系统现代化的方法。这种模式用无花果藤作为类比，这种藤蔓成长为一棵树，最终战胜并取代了宿主。该模式是由 [Martin Fowler](#) 提出的，作为重写单体系统时管理风险的一种方法。有关如何应用此模式的示例，请参阅[使用容器和 Amazon API Gateway 逐步将原有的 Microsoft ASP.NET \(ASMX \) Web 服务现代化](#)。

子网

您的 VPC 内的一个 IP 地址范围。子网必须位于单个可用区中。

监督控制和数据采集 (SCADA)

在制造业中，一种使用硬件和软件来监控实物资产和生产操作的系统。

对称加密

一种加密算法，它使用相同的密钥来加密和解密数据。

综合测试

以模拟用户交互的方式测试系统，以检测潜在问题或监控性能。您可以使用 [Amazon S CloudWatch ynthetic](#) 来创建这些测试。

系统提示

一种为 [LLM](#) 提供上下文、说明或准则以指导其行为的技术。系统提示有助于设置上下文并制定与用户交互的规则。

T

标签

键值对，用作组织资源的元数据。AWS 标签有助于您管理、识别、组织、搜索和筛选 资源。有关更多信息，请参阅[标记您的 AWS 资源](#)。

目标变量

您在监督式 ML 中尝试预测的值。这也被称为结果变量。例如，在制造环境中，目标变量可能是产品缺陷。

任务列表

一种通过运行手册用于跟踪进度的工具。任务列表包含运行手册的概述和要完成的常规任务列表。对于每项常规任务，它包括预计所需时间、所有者和进度。

测试环境

请参阅[环境](#)。

训练

为您的 ML 模型提供学习数据。训练数据必须包含正确答案。学习算法在训练数据中查找将输入数据属性映射到目标（您希望预测的答案）的模式。然后输出捕获这些模式的 ML 模型。然后，您可以使用 ML 模型对不知道目标的新数据进行预测。

中转网关

一个网络传输中心，可用于将您的网络 VPCs 和本地网络互连。有关更多信息，请参阅 AWS Transit Gateway 文档中的[什么是公交网关](#)。

基于中继的工作流程

一种方法，开发人员在功能分支中本地构建和测试功能，然后将这些更改合并到主分支中。然后，按顺序将主分支构建到开发、预生产和生产环境。

可信访问权限

向您指定的服务授予权限，该服务可代表您在其账户中执行任务。AWS Organizations 当需要服务相关的角色时，受信任的服务会在每个账户中创建一个角色，为您执行管理任务。有关更多信息，请参阅 AWS Organizations 文档中的[AWS Organizations 与其他 AWS 服务一起使用](#)。

优化

更改训练过程的各个方面，以提高 ML 模型的准确性。例如，您可以通过生成标签集、添加标签，并在不同的设置下多次重复这些步骤来优化模型，从而训练 ML 模型。

双披萨团队

一个小 DevOps 团队，你可以用两个披萨来喂食。双披萨团队的规模可确保在软件开发过程中充分协作。

U

不确定性

这一概念指的是不精确、不完整或未知的信息，这些信息可能会破坏预测式 ML 模型的可靠性。不确定性有两种类型：认知不确定性是由有限的、不完整的数据造成的，而偶然不确定性是由数据中固有的噪声和随机性导致的。

无差别任务

也称为繁重工作，即创建和运行应用程序所必需的工作，但不能为最终用户提供直接价值或竞争优势。无差别任务的示例包括采购、维护和容量规划。

上层环境

请参阅[环境](#)。

V

vacuum 操作

一种数据库维护操作，包括在增量更新后进行清理，以回收存储空间并提高性能。

版本控制

跟踪更改的过程和工具，例如存储库中源代码的更改。

VPC 对等连接

两者之间的连接 VPCs，允许您使用私有 IP 地址路由流量。有关更多信息，请参阅 Amazon VPC 文档中的[什么是 VPC 对等连接](#)。

漏洞

损害系统安全的软件缺陷或硬件缺陷。

W

热缓存

一种包含经常访问的当前相关数据的缓冲区缓存。数据库实例可以从缓冲区缓存读取，这比从主内存或磁盘读取要快。

暖数据

不常访问的数据。查询此类数据时，通常可以接受中速查询。

窗口函数

一种对与当前记录有某种关联的一组行执行计算的 SQL 函数。窗口函数对于处理任务很有用，例如计算移动平均值或根据当前行的相对位置访问行的值。

工作负载

一系列资源和代码，它们可以提供商业价值，如面向客户的应用程序或后端过程。

工作流

迁移项目中负责一组特定任务的职能小组。每个工作流都是独立的，但支持项目中的其他工作流。例如，组合工作流负责确定应用程序的优先级、波次规划和收集迁移元数据。组合工作流将这些资产交付给迁移工作流，然后迁移服务器和应用程序。

WORM

请参阅[一次写入多次读取](#)。

WQF

请参阅[AWS 工作负载资格鉴定框架](#)。

一次写入多次读取 (WORM)

一种存储模型，可一次写入数据并防止数据被删除或修改。授权用户可以根据需要多次读取数据，但无法对其进行更改。此数据存储基础设施被认为[不可变](#)。

Z

零日漏洞利用

一种利用[零日漏洞](#)的攻击，通常为恶意软件。

零日漏洞

生产系统中不可避免的缺陷或漏洞。威胁主体可能利用这种类型的漏洞攻击系统。开发人员经常因攻击而意识到该漏洞。

零样本提示

为[LLM](#)提供执行任务的说明，但没有可以帮助指导的示例（样本）。LLM 必须使用预先训练的知识来处理任务。零样本提示的有效性取决于任务的复杂性和提示的质量。另请参阅[少样本提示](#)。

僵尸应用程序

平均 CPU 和内存使用率低于 5% 的应用程序。在迁移项目中，通常会停用这些应用程序。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。