Terraform 入门：专家指南和 AWS CDK 专家指南 AWS CloudFormation

AWS 规范性指导

AWS 规范性指导: Terraform 入门：专家指南和 AWS CDK 专家指南 AWS CloudFormation

Table of Contents

Terraform 入门：AWS CDK 和 AWS 专家指南 CloudFormation

Steven Guggenheimer，Amazon Web Services ()AWS

2024 年 3 月（文档历史记录）

如果您仅在配置云资源方面的经验属于以下领域 AWS，那么除了AWS Cloud Development Kit (AWS CDK)和AWS CloudFormation之外，您使用基础设施即代码 (IaC) 工具的经验可能有限。实际上，类似的工具，例如Hashicorp Terraform，你可能完全不熟悉。但是，你进入云之旅的深度越深，遇到Terraform的不可避免性就越大。熟悉其核心概念绝对对你有利。

虽然 Terraform、 AWS CDK、和 CloudFormation 实现了相似的目标并有许多共同的核心概念，但还是有许多不同之处。如果你是第一次接触 Terraform，你可能还没有为这些差异做好准备。毕竟 AWS CDK ， CloudFormation堆栈都位于其中 AWS 账户，因此，通过这种方式，它们与它们维护的大多数资源有着直接的关系。Terraform 不基于任何单一云提供商的环境。这使它能够灵活地支持各种不同的提供商，但它必须维护来自远程位置的资源。

本指南有助于揭开Terraform背后的核心概念的神秘面纱，以帮助您应对遇到的任何Iac挑战。它重点介绍 Terraform 如何使用提供者、模块和状态文件等概念来配置资源。它还将 Terraform 的概念与 AWS CDK 和 CloudFormation 执行类似操作的方式进行了对比。

> **Note**
>
> AWS CDK 可帮助开发人员使用编程编码语言部署 CloudFormation 堆栈。运行后`cdk synth`，您的代码将转换为 CloudFormation模板。从那时起， AWS CDK 和之间的过程是相同的 CloudFormation。为了简洁起见，本指南通常用 CloudFormation 术语来提及 AWS Iac流程，但比较同样适合。 AWS CDK

CloudFormation 和 Terraform 术语

在将 Terraform 与 AWS CDK 和进行比较时 CloudFormation，由于描述它们时使用的术语不一致，因此很难协调 IaC 的核心概念。以下是这些术语以及本指南将如何引用它们：

- 堆栈 — 堆栈是 IaC，它部署到 CI/CD 管道中，可作为单个单元进行跟踪。尽管这个术语很常见 CloudFormation，但Terraform并没有真正使用这个术语。Terraform 堆栈是一个已部署的根模块，

包含其所有子模块。但是，为了避免与“模块”一词混淆，本指南使用术语堆栈来描述这两个工具的单一部署。

- 状态 -状态是 IaC 部署堆栈中当前跟踪的所有资源及其当前配置。如了解 Terraform 状态和后端本节所述，Terraform 更多地使用了“状态”一词。 CloudFormation这是因为在 Terraform 中维护状态更加明显，但是跟踪和更新状态同样重要。 CloudFormation
- IaC 文件 — IaC 文件是包含基础设施即代码 (IaC) 语言的单个文件。 CloudFormation 指单个 CloudFormation 文件作为模板。但是，Terraform 中的模板和模板文件完全不同。等同于 Terraform 中 CloudFormation 模板的被称为配置文件。为了最大限度地减少本指南中的混乱，术语文件或 IaC 文件用于指代 CloudFormation 模板和 Terraform 配置文件。

下表比较了用于 CloudFormation 和 Terraform 的术语。此表的目的是显示相似之处。这些不是 one-to-one 比较。 CloudFormation 和 Terraform 之间的每个概念至少略有不同。本指南的相关部分对概念进行了深入的解释。

CloudFormation 术语	Terraform 术语	本指南的章节
CDK 接口（比如 i Bucket）	数据来源	了解 Terraform 数据源
变更套装	规划	了解 Terraform 模块
条件函数	条件表达式	了解 Terraform 函数、表达式和元参数
`DependsOn` 属性	`depends_on` 元论点	了解 Terraform 函数、表达式和元参数
内置函数	函数	了解 Terraform 函数、表达式和元参数
模块	模块	了解 Terraform 模块
输出	输出值	了解 Terraform 变量、局部值和输出
参数	Variables	了解 Terraform 变量、局部值和输出
注册表	提供商	了解 Terraform 提供商

CloudFormation 术语	Terraform 术语	本指南的章节
模板	配置文件	全部

了解 Terraform 资源

两者兼 AWS CloudFormation 而有之 Terraform 的主要原因是云资源的创建和维护。但是云资源到底是什么？ CloudFormation资源和 Terraform 资源是一回事吗？ 答案是... 是的和不是。为了说明这一点，本指南提供了一个示例，说明如何使用 CloudFormation 然后使用 Terraform 创建亚马逊简单存储服务 (Amazon S3) 存储桶。

以下 CloudFormation 代码示例创建了一个示例 Amazon S3 存储桶。

```
{
    "myS3Bucket": {
        "Type": "AWS::S3::Bucket",
        "Properties": {
            "BucketName": "my-s3-bucket",
            "BucketEncryption": {
                "ServerSideEncryptionConfiguration": [
                    {
                        "ServerSideEncryptionByDefault": {
                            "SSEAlgorithm": "AES256"
                        }
                    }
                ]
            },
            "PublicAccessBlockConfiguration": {
                "BlockPublicAcls": true,
                "BlockPublicPolicy": true,
                "IgnorePublicAcls": true,
                "RestrictPublicBuckets": true
            },
            "VersioningConfiguration": {
                "Status": "Enabled"
            }
        }
    }
}
```

以下 Terraform 代码示例创建了一个相同的 Amazon S3 存储桶。

```
resource "aws_s3_bucket" "myS3Bucket" {
  bucket = "my-s3-bucket"
}
```

```
resource "aws_s3_bucket_server_side_encryption_configuration" "bucketencryption" {
  bucket = aws_s3_bucket.myS3Bucket.id
  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}

resource "aws_s3_bucket_public_access_block" "publicaccess" {
  bucket                  = aws_s3_bucket.myS3Bucket.id
  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}

resource "aws_s3_bucket_versioning" "versioning" {
  bucket = aws_s3_bucket.myS3Bucket.id
  versioning_configuration {
    status = "Enabled"
  }
}
```

对于 Terraform，提供者定义资源，然后开发人员声明和配置这些资源。提供者是本指南将在下一节中讨论的概念。Terraform 示例为 S3 存储桶的多个设置创建了完全独立的资源。为设置创建单独的资源不一定是 Terraform Prov AWS ider 处理 AWS 资源的典型方式。但是，这个例子显示了一个重要的区别。虽然 CloudFormation 资源是由CloudFormation资源规范严格定义的，但 Terraform 没有这样的要求。在 Terraform 中，资源的概念更加模糊。

尽管在定义单一资源的确切护栏方面，这些工具可能有所不同，但一般而言，云资源是指存在于云中的任何特定实体，可以创建、更新或删除。因此，不管涉及多少资源，前面的两个示例都创建了完全相同的东西，其中的设置完全相同 AWS 账户。

了解 Terraform 提供商

在 Terraform 中，提供者是一个与云提供商、第三方工具和其他 API 交互的插件。要将 Terraform 与 AWS Terraform 配合使用，您可以使用与资源交互的AWS 提供程序。 AWS

如果你从未使用过AWS CloudFormation 注册表将第三方扩展整合到部署堆栈中，那么 Terraform 提供商可能需要一些时间来适应。因为 CloudFormation 是原生的 AWS，所以默认情况下， AWS 资源提供者已经存在了。另一方面，Terraform 没有单一的默认提供者，因此无法假设给定资源的起源。这意味着需要在 Terraform 配置文件中声明的第一件事就是资源的去向以及它们将如何到达那里。

这种区别给 Terraform 增加了一层额外的复杂性，这是不存在的。 CloudFormation但是，这种复杂性增加了灵活性。您可以在单个 Terraform 模块中声明多个提供者，然后创建的底层资源可以作为同一部署层的一部分相互交互。

这在很多方面都很有用。提供商不一定必须是单独的云提供商。提供商可以代表任何云资源来源。例如，以亚马逊 Elastic Kubernetes Service（亚马逊 EKS）为例。在配置 Amazon EKS 集群时，您可能需要使用 Helm 图表来管理第三方扩展，并使用 Kubernetes 本身来管理容器资源。因为 AWSHelm 和 Kubernetes 都有自己的 Terraform 提供程序，所以你可以同时配置和集成这些资源，然后在它们之间传递值。

在以下 Terraform 的代码示例中， AWS 提供程序创建了一个 Amazon EKS 集群，然后将生成的 Kubernetes 配置信息传递给 Helm 和 Kubernetes 提供商。

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 4.33.0"
    }

    helm = {
      source  = "hashicorp/helm"
      version = "2.12.1"
    }

    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = "2.26.0"
    }
  }
  required_version = ">= 1.2.0"
```

```
}

provider "aws" {
  region = "us-west-2"
}

resource "aws_eks_cluster" "example_0" {
  name     = "example_0"
  role_arn = aws_iam_role.cluster_role.arn
  vpc_config {
    endpoint_private_access = true
    endpoint_public_access  = true
    subnet_ids              = var.subnet_ids
  }
}

locals {
  host        = aws_eks_cluster.example_0.endpoint
  certificate = base64decode(aws_eks_cluster.example_0.certificate_authority.data)
}

provider "helm" {
  kubernetes {
    host                   = local.host
    cluster_ca_certificate = local.certificate
    # exec allows for an authentication command to be run to obtain user
    # credentials rather than having them stored directly in the file
    exec {
      api_version = "client.authentication.k8s.io/v1beta1"
      args        = ["eks", "get-token", "--cluster-name",
 aws_eks_cluster.example_0.name]
      command     = "aws"
    }
  }
}

provider "kubernetes" {
  host                   = local.host
  cluster_ca_certificate = local.certificate
  exec {
    api_version = "client.authentication.k8s.io/v1beta1"
    args        = ["eks", "get-token", "--cluster-name",
 aws_eks_cluster.example_0.name]
    command     = "aws"
```

```
  }
}
```

就这两个 IaC 工具而言，提供商需要权衡取舍。Terraform 完全依赖位于外部的提供程序包，这些软件包是推动其部署的引擎。 CloudFormation 内部支持所有主要 AWS 进程。使用 CloudFormation，只有当你想加入第三方扩展时，才需要担心第三方提供商。每种方法都有其优点和缺点。哪一个最适合你超出了本指南的范围，但是在评估这两种工具时，记住其中的区别很重要。

使用 Terraform 别名

在 Terraform 中，您可以将自定义配置传递给每个提供程序。那么，如果你想在同一个模块中使用多个提供商配置怎么办？ 在这种情况下，你必须使用别名。 别名可帮助您选择在每个资源或每个模块级别上使用哪个提供商。当您有同一个提供程序的多个实例时，您可以使用别名来定义非默认实例。例如，您的默认提供商实例可能是特定的 AWS 区域，但您可以使用别名来定义备用区域。

以下 Terraform 示例显示了如何使用别名在不同的存储桶中配置存储桶。 AWS 区域提供商的默认区域为`us-west-2`，但您可以使用 east 别名在中配置资源`us-east-2`。

```
provider "aws" {
  region = "us-west-2"
}

provider "aws" {
  alias  = "east"
  region = "us-east-2"
}

resource "aws_s3_bucket" "myWestS3Bucket" {
  bucket = "my-west-s3-bucket"
}

resource "aws_s3_bucket" "myEastS3Bucket" {
  provider = aws.east
  bucket   = "my-east-s3-bucket"
}
```

当你使用和`provider`元参数时，如前面的示例所示，你可以为特定资源指定不同的提供者配置。`alias`在单个堆栈 AWS 区域 中以多个方式配置资源仅仅是个开始。别名提供者在很多方面都非常方便。

例如，一次配置多个 Kubernetes 集群是很常见的。别名可以帮助您配置其他 Helm 和 Kubernetes 提供程序，这样您就可以针对不同的 Amazon EKS 资源以不同的方式使用这些第三方工具。以下 Terraform 代码示例说明了如何使用别名来执行此任务。

```
resource "aws_eks_cluster" "example_0" {
  name     = "example_0"
  role_arn = aws_iam_role.cluster_role.arn
  vpc_config {
    endpoint_private_access = true
    endpoint_public_access  = true
    subnet_ids              = var.subnet_ids[0]
  }
}

resource "aws_eks_cluster" "example_1" {
  name     = "example_1"
  role_arn = aws_iam_role.cluster_role.arn
  vpc_config {
    endpoint_private_access = true
    endpoint_public_access  = true
    subnet_ids              = var.subnet_ids[1]
  }
}

locals {
  host         = aws_eks_cluster.example_0.endpoint
  certificate  = base64decode(aws_eks_cluster.example_0.certificate_authority.data)
  host1        = aws_eks_cluster.example_1.endpoint
  certificate1 = base64decode(aws_eks_cluster.example_1.certificate_authority.data)
}

provider "helm" {
  kubernetes {
    host                   = local.host
    cluster_ca_certificate = local.certificate
    exec {
      api_version = "client.authentication.k8s.io/v1beta1"
      args        = ["eks", "get-token", "--cluster-name",
 aws_eks_cluster.example_0.name]
      command     = "aws"
    }
  }
}
```

```
provider "helm" {
  alias = "helm1"
  kubernetes {
    host                   = local.host1
    cluster_ca_certificate = local.certificate1
    exec {
      api_version = "client.authentication.k8s.io/v1beta1"
      args        = ["eks", "get-token", "--cluster-name",
 aws_eks_cluster.example_1.name]
      command     = "aws"
    }
  }
}

provider "kubernetes" {
  host                   = local.host
  cluster_ca_certificate = local.certificate
  exec {
    api_version = "client.authentication.k8s.io/v1beta1"
    args        = ["eks", "get-token", "--cluster-name",
 aws_eks_cluster.example_0.name]
    command     = "aws"
  }
}

provider "kubernetes" {
  alias                  = "kubernetes1"
  host                   = local.host1
  cluster_ca_certificate = local.certificate1
  exec {
    api_version = "client.authentication.k8s.io/v1beta1"
    args        = ["eks", "get-token", "--cluster-name",
 aws_eks_cluster.example_1.name]
    command     = "aws"
  }
}
```

了解 Terraform 模块

在基础设施即代码 (IaC) 领域，模块是一个独立的代码块，它被隔离并打包在一起以供重复使用。模块的概念是 Terraform 开发中不可避免的方面。有关更多信息，请参阅 Terraform 文档中的模块。 AWS CloudFormation 还支持模块。有关更多信息，请参阅 AWS 云运营和迁移博客中的AWS CloudFormation 模块简介。

Terraform 和 Terraform 中模块的主要区别在 CloudFormation 于， CloudFormation 模块是通过使用特殊的资源类型 () `AWS::CloudFormation::ModuleVersion` 导入的。在 Terraform 中，每个配置都至少有一个模块，称为根模块。主.tf 文件或 Terraform 配置文件中的文件中的 Terraform 资源被视为位于根模块中。然后，根模块可以调用其他模块以包含在堆栈中。以下示例显示了一个根模块使用开源 eks 模块配置亚马逊 Elastic Kubernetes Service (Amazon EKS) 集群。

```
terraform {
  required_providers {
    helm = {
      source  = "hashicorp/helm"
      version = "2.12.1"
    }
  }
  required_version = ">= 1.2.0"
}

module "eks" {
  source  = "terraform-aws-modules/eks/aws"
  version = "20.2.1"
  vpc_id  = var.vpc_id
}

provider "helm" {
  kubernetes {
    host                   = module.eks.cluster_endpoint
    cluster_ca_certificate =
 base64decode(module.eks.cluster_certificate_authority_data)
  }
}
```

您可能已经注意到，上面的配置文件不包括 AWS 提供程序。这是因为模块是独立的，可以包含自己的提供者。由于 Terraform 提供程序是全球性的，因此可以在根模块中使用子模块中的提供程序。但是，并非所有模块值都是如此。默认情况下，模块中的其他内部值仅限于该模块，需要声明为输出才能在

根模块中访问。您可以利用开源模块来简化堆栈中的资源创建。例如，eks 模块所做的不仅仅是配置 EKS 集群，它还提供了一个功能齐全的 Kubernetes 环境。只要eks模块配置符合您的需求，使用它可以省去编写数十行额外代码的麻烦。

调用模块

你在 Terraform 部署期间运行的两个主要 Terraform CLI 命令是 terraform init 和 terra form apply。该`terraform init`命令执行的默认步骤之一是找到所有子模块，并将它们作为依赖项导入到`.terraform/modules`目录中。在开发过程中，无论何时添加新的外部源模块，都必须在使用命令之前重新初始化。`apply`当你听到有人提及 Terraform 模块时，它指的是这个目录中的软件包。严格来说，你在代码中声明的模块就是调用模块，所以在实践中，module 关键字调用实际的模块，该模块存储为依赖项。

这样，调用模块可以更简洁地代表部署时要替换的完整模块。你可以利用这个想法，在堆栈中创建自己的模块，使用任何你想要的标准来强制资源的逻辑分离。请记住，这样做的最终目标应该是降低堆栈的复杂性。由于在模块之间共享数据需要您从模块内部输出数据，因此有时过于依赖模块会使事情变得过于复杂。

根模块

由于每个 Terraform 配置都至少有一个模块，因此检查你最常处理的模块（根模块）的模块属性会有所帮助。每当你处理 Terraform 项目时，根模块都包含顶级目录中的所有`.tf`（或`.tf.json`）文件。当你在该顶级目录`terraform apply`中运行时，Terraform 会尝试运行它在那里找到的所有`.tf`文件。除非在其中一个顶级配置文件中调用子目录中的任何文件，否则这些文件都将被忽略。

这为你的代码结构提供了一定的灵活性。这也是为什么将 Terraform 部署称为模块而不是文件更准确的原因，因为单个进程中可能涉及多个文件。对于最佳实践，Terraform 推荐了一种标准模块结构。但是，如果您要将任何`.tf`文件放在顶层目录中，它将与其余文件一起运行。实际上，模块中的所有顶级`.tf`文件都是在你运行时部署的`terraform apply`。那么 Terraform 首先运行哪个文件呢？ 这个问题的答案非常重要。

在初始化之后和堆栈部署之前，Terraform 会执行一系列步骤。首先，分析现有配置，然后创建依赖关系图。依赖关系图决定了需要哪些资源以及应按什么顺序分配这些资源。例如，包含在其他资源中引用的属性的资源将在其依赖资源之前进行处理。同样，使用`depends_on`参数显式声明依赖关系的资源将在它们指定的资源之后进行处理。在可能的情况下，Terraform 可以实现并行性并同时处理非依赖资源。在部署之前，你可以使用 terraform graph 命令查看依赖关系图。

创建依赖关系图后，Terraform 会确定部署期间需要做什么。它将依赖关系图与最新的状态文件进行比较。此过程的结果称为计划，它非常像 CloudFormation变更集。你可以使用 terraform plan 命令查看当前的计划。

作为最佳实践，建议尽可能接近标准模块结构。如果您的配置文件变得太长而无法有效管理，而逻辑分隔可以简化管理，则可以将代码分散到多个文件中。请记住依赖关系图和计划流程的工作原理，以使您的堆栈尽可能高效地运行。

了解 Terraform 状态和后端

基础设施即代码 (IaC) 中最重要的概念之一是状态的概念。IaC 服务保持状态，这允许您在 IaC 文件中声明资源，而无需在每次部署时都重新创建该资源。IaC 文件记录了部署结束时所有资源的状态，这样它就可以将该状态与下一次部署中声明的目标状态进行比较。因此，如果当前状态包含`my-s3-bucket`名为的 Amazon Simple Storage Service (Amazon S3) 存储桶，并且传入的更改也包含相同的存储桶，则新流程将对现有存储桶应用发现的所有更改，而不是尝试创建全新的存储桶。

下表提供了常规 IaC 状态过程的示例。

当前状态	目标状态	操作
没有命名的 S3 存储桶 `my-s3-bucket`	名为 S3 存储桶 `my-s3-bucket`	创建名为的 S3 存储桶 `my-s3-bucket`
`my-s3-bucket` 未配置存储桶版本控制	`my-s3-bucket` 未配置存储桶版本控制	无需操作
`my-s3-bucket` 未配置存储桶版本控制	`my-s3-bucket` 配置了存储桶版本控制	配置`my-s3-bucket` 为使用存储桶版本控制
`my-s3-bucket` 配置了存储桶版本控制	没有命名的 S3 存储桶 `my-s3-bucket`	正在尝试删除 `my-s3-bucket`

要了解 AWS CloudFormation 和Terraform跟踪状态的不同方式，重要的是要记住这两个工具之间的第一个基本区别：托管在内部 AWS 云，而Terraform本质上 CloudFormation 是远程的。这一事实 CloudFormation 允许在内部维持状态。你可以进入 CloudFormation 控制台查看给定堆栈的事件历史记录，但 CloudFormation 服务本身会为你强制执行状态规则。

在给定资源下 CloudFormation 运行的三种模式是`Create`、`Update`和`Delete`。当前模式是根据上次部署中发生的情况确定的，否则无法对其产生影响。你也许可以手动更新 CloudFormation 资源以影响所确定的模式，但你不能向其传递一条写 CloudFormation 着 “对于这个资源，在`Create`模式下操作” 的命令。

由于 Terraform 未托管在中 AWS 云，因此维护状态的过程必须更具可配置性。因此，Terraform 状态保存在自动生成的状态文件中。Terraform 开发者必须比处理状态更直接地处理状态。CloudFormation要记住的重要一点是，跟踪状态对两个工具同样重要。

默认情况下，Terraform 状态文件存储在本地运行 Terraform 堆栈的主目录的顶层。如果你在本地开发环境中运行该`terraform apply`命令，你可以看到 Terraform 生成了它用来实时维护状态的 terraform.tfstat e 文件。不管好坏，这让你对Terraform状态的控制比你在Terraform中的状态要多得多。 CloudFormation虽然你永远不应该直接更新状态文件，但你可以运行几个 Terraform CLI 命令来更新两次部署之间的状态。例如，terraform 导入允许您将在 Terraform 之外创建的资源添加到部署堆栈中。相反，你可以通过运行 terraform state rm 将资源从状态中移除。

Terraform 需要将其状态存储在某个地方，这一事实导致了另一个不适用于后端的概念。CloudFormationTerraform 后端是 Terraform 堆栈在部署后存储其状态文件的地方。这也是它期望在新部署开始时找到状态文件的地方。如上所述，在本地运行堆栈时，可以在顶级本地目录中保存 Terraform 状态的副本。这被称为本地后端。

在为持续集成和持续部署 (CI/CD) 环境进行开发时，本地状态文件通常包含在 .gitignore 文件中，以使其不受版本控制。然后管道中就没有本地状态文件了。为了正常工作，该管道阶段需要在某处找到正确的状态文件。这就是为什么 Terraform 配置文件通常包含后端块的原因。后端块向 Terraform 堆栈表明，它需要在自己的顶级目录之外寻找其他地方才能找到状态文件。

Terraform 后端几乎可以位于任何地方：亚马逊 S3 存储桶、API 终端节点，甚至是远程 Ter raform 工作空间。以下是存储在 Amazon S3 存储桶中的 Terraform 后端的示例。

```
terraform {
  backend "s3" {
    bucket = "my-s3-bucket"
    key    = "state-file-folder"
    region = "us-east-1"
  }
}
```

为了避免在 Terraform 配置文件中存储敏感信息，后端还支持部分配置。在前面的示例中，配置中不存在访问存储桶所需的凭证。凭证可以从环境变量中获取，也可以通过其他方式获取，例如 AWS Secrets Manager。有关更多信息，请参阅使用 AWS Secrets Manager 和 HashiCorp Terraform 保护敏感数据。

常见的后端场景是在本地环境中用于测试目的的本地后端。terraform.tfstat e 文件包含在 .gitignore 文件中，因此不会将其推送到远程存储库。然后，CI/CD 管道中的每个环境都将维护自己的后端。在这种情况下，可能有多个开发人员可以访问此远程状态，因此您需要保护状态文件的完整性。如果多个部署同时运行并更新状态，则状态文件可能会损坏。因此，在非本地后端的情况下，状态文件通常在部署期间被锁定。

了解 Terraform 数据源

部署堆栈通常依赖来自先前现有资源的数据。大多数 IaC 工具都有一种方法可以导入由其他流程创建的资源。这些导入的资源通常是只读的（尽管 IAM 角色是一个明显的例外），用于访问堆栈中资源所需的数据。 AWS CloudFormation 允许导入资源，但是通过查看. 可以更好地解释这个想法 AWS Cloud Development Kit (AWS CDK)。

可 AWS CDK 帮助开发人员使用现有的编程语言生成 CloudFormation模板。 AWS CDK 操作的最终结果是在中导入资源 CloudFormation。但是，与使用的语法可以更 AWS CDK 轻松地与 Terraform 进行比较。以下是使用导入资源的示例 AWS CDK。

```
const importedBucket: IBucket = Bucket.fromBucketAttributes(
    scope,
    "imported-bucket",
    {
        bucketName: "My_S3_Bucket"
    }
);
```

导入的资源通常是通过在用于创建相同类型的新资源的同一个类上调用静态方法来创建的。调用new Bucket(...将创建新资源，调用Bucket.fromBucketAttributes(...导入现有资源。您可以将存储桶属性的子集传递到函数中，这样他们 AWS CDK 就可以找到正确的存储桶。但是，另一个区别是，创建新存储桶会返回该Bucket类的完整实例，其中包含所有可用的属性和方法。导入资源会返回IBucket，该类型仅包含Bucket必须具备的属性。尽管您可以从外部堆栈导入资源，但使用该资源的选项是有限的。

在 Terraform 中，使用数据源可以实现类似的目标。大多数定义的 Terraform 资源旁边都有附带的数据源。以下是 Terraform S3 存储桶资源及其相应数据源的示例。

```
# S3 Bucket resource:
resource "aws_s3_bucket" "My_S3_Bucket" {
  bucket = "My_S3_Bucket"
}

# S3 Bucket data source:
data "aws_s3_bucket" "My_S3_Bucket" {
  bucket = "My_S3_Bucket"
}
```

这两项之间的唯一区别是名称前缀。如数据源的文档所示，可以传递给数据源的参数少于资源。这是因为资源使用这些参数来声明新 S3 存储桶的所有属性，而数据源只需要足够的信息来唯一地识别和导入现有资源的数据。

Terraform 资源和数据源的语法相似可能很方便，但也可能存在问题。对于新手 Terraform 开发者来说，在配置中不小心使用了数据源而不是资源是很常见的。Terraform 数据源始终是只读的。您可以使用它们代替相应的资源进行读取操作（例如向其他资源提供 ID 名称）。但是，你不能将它们用于写入操作，这会从根本上改变底层资源的某些方面。因此，您可以将 Terraform 数据源视为底层资源的克隆版本。

与之前的 AWS CDK iBucket 示例类似，数据源对于只读场景非常有用。如果您需要从现有资源获取数据，但不需要在堆栈中维护该资源，请使用数据源。这方面的一个很好的例子是，当你创建使用账户默认 VPC 的 Amazon EC2 实例时。由于该 VPC 已经存在，因此您只需提取其数据即可。以下代码示例展示了如何使用数据来识别目标 VPC。

```
data "aws_vpc" "default" {
  default = true
}

resource "aws_instance" "instance1" {
  ami           = "ami-123456"
  instance_type = "t2.micro"
  subnet_id     = data.aws_vpc.default.main_route_table_id
}
```

了解 Terraform 变量、局部值和输出

变量允许在代码块中使用占位符，从而增强了代码的灵活性。每当重复使用代码时，变量可以表示不同的值。Terraform 通过其模块化范围来区分其变量类型。输入变量是可以注入模块的外部值，输出值是可以在外部共享的内部值，局部值始终保持在其原始范围内。

Variables

AWS CloudFormation 使用参数来表示自定义值，这些值可以从一个堆栈部署到下一个堆栈部署进行设置和重置。同样，Terraform 使用输入变量或变量。变量可以在 Terraform 配置文件中的任何地方声明，并且通常使用所需的数据类型或默认值进行声明。以下所有三个表达式都是有效的 Terraform 变量声明。

```
variable "thing_i_made_up" {
  type = string
}

variable "random_number" {
  default = 5
}

variable "dogs" {
  type = list(object({
    name  = string
    breed = string
  }))

  default = [
    {
      name  = "Sparky",
      breed = "poodle"
    }
  ]
}
```

要在配置中访问 Sparky 的品种，你可以使用变量`var.dogs[0].breed`。如果变量没有默认值且未被归类为可为空，则必须为每次部署设置该变量的值。否则，可以选择为变量设置一个新值。在根模块中，可以在命令行、环境变量或 terrafor m.tfv ars 文件中将当前变量值设置为环境变量。以下示例说明如何在 terraform.tfvars 文件中输入变量值，该文件存储在模块的顶级目录中。

```
# terraform.tfvars
dogs = [
   {
      name  = "Sparky",
      breed = "poodle"
   },
   {
      name  = "Fluffy",
      breed = "chihuahua"
   }
]

random_number = 7

thing_i_made_up = "Kabibble"
```

在此示例dogs中，terraform.tfvars 文件的值将覆盖变量声明中的默认值。如果您在子模块中声明变量，则可以直接在模块声明块中设置变量值，如以下示例所示。

```
module "my_custom_module" {
  source        = "modulesource/custom"
  version       = "0.0.1"
  random_number = 8
}
```

声明变量时可以使用的其他一些参数包括：

- `sensitive`— 将其设置为`true`防止变量值暴露在 Terraform 进程输出中。
- `nullable`— 将其设置为`true`允许变量没有值。这对于未设置默认值的变量来说很方便。
- `description`— 将变量的描述添加到堆栈的元数据中。
- `validation`— 为变量设置验证规则。

Terraform 变量最方便的方面之一是能够在变量声明中添加一个或多个验证对象。您可以使用验证对象添加一个条件，即变量必须通过，否则部署将失败。您也可以设置一条自定义错误消息，使其在违反条件时显示。

例如，你正在设置一个 Terraform 配置文件，你的团队成员将运行该文件。在部署堆栈之前，团队成员需要创建一个 terraform.tfvars 文件来设置重要的配置值。为了提醒他们，你可以做类似以下的事情。

```
variable "important_config_setting" {
```

```
  type = string

  validation {
    condition     = length(var.important_config_setting) > 0
    error_message = "Don't forget to create the terraform.tfvars file!"
  }

  validation {
    condition     = substr(var.important_config_setting, 0, 7) == "prefix-"
    error_message = "Remember that the value always needs to start with 'prefix-'"
  }
}
```

如本示例所示，您可以在单个变量中设置多个条件。Terraform 仅显示失败条件的错误消息。通过这种方式，您可以对变量值强制执行各种规则。如果变量值导致管道故障，您将确切地知道原因。

本地值

如果模块中存在您想要别名的值，请使用`locals`关键字，而不是声明一个永远不会更新的默认变量。顾名思义，一个`locals`区块包含的术语在内部限定于该特定模块。如果要转换字符串值，例如通过在变量值中添加前缀以用于资源名称，则使用本地值可能是一个不错的解决方案。单个`locals`块可以为你的模块声明所有本地值，如以下示例所示。

```
locals {
  moduleName    = "My Module"
  localConfigId = concat("prefix-", var.important_config_setting)
}
```

请记住，当你访问该值时，`locals`关键字会变成单数，例如`local.LocalConfigId`。

输出值

如果 Terraform 输入变量就像 CloudFormation 参数一样，那么你可以说 Terraform 的输出值就像输出一样。CloudFormation两者都用于公开部署堆栈中的值。但是，由于 Terraform 模块在工具结构中更加根深蒂固，因此 Terraform 输出值也用于将模块中的值公开给父模块或其他子模块，即使这些模块都在同一个部署堆栈中。如果您正在构建两个自定义模块，而第一个模块需要访问第二个模块的 ID 值，则需要将以下`output`块添加到第二个模块。

```
output "module_id" {
```

```
  value = local.module_id
}
Then in the first module you could use it like this:
module "first_module" {
  source = "path/to/first/module"
}

resource "example_resource" "example_resource_name" {
  module_id = module.first_module.module_id
}
```

由于 Terraform 输出值可以在同一个堆栈中使用，因此您也可以在output块中使用该sensitive属性来抑制该值显示在堆栈输出中。此外，方output块可以像变量使用precondition块一样使用validation块：以确保变量遵循一组特定的规则。这有助于确保在继续部署之前，模块中的所有值都按预期存在。

```
output "important_config_setting" {
  value = var.important_config_setting

  precondition {
    condition     = length(var.important_config_setting) > 0
    error_message = "You forgot to create the terraform.tfvars file again."
  }
}
```

了解 Terraform 函数、表达式和元参数

对使用声明式配置文件而不是常见编程语言的 IaC 工具的一种批评是，它们使实现自定义编程逻辑变得更加困难。在 Terraform 配置中，这个问题是通过使用函数、表达式和元参数来解决的。

函数

使用代码来配置基础架构的最大优势之一是能够存储常见的工作流程并一次又一次地重复使用它们，通常每次都传递不同的参数。Terraform 函数与 AWS CloudFormation 内部函数类似，尽管它们的语法与编程语言中函数的调用方式更为相似。在本指南的示例中，你可能已经注意到一些 Terraform 函数，例如 substr、concat、lengt h 和 base64decode。 CloudFormation 与内部函数一样，Terraform 有一系列可在您的配置中使用的内置函数。例如，如果特定的资源属性采用一个非常大的 JSON 对象，而直接粘贴到文件中效率低下，则可以将该对象放在 .json 文件中，然后使用 Terraform 函数对其进行访问。在以下示例中，该`file`函数以字符串形式返回文件内容，然后该`jsondecode`函数将其转换为对象类型。

```
resource "example_resource" "example_resource_name" {
  json_object = jsondecode(file("/path/to/file.json"))
}
```

Expressions

Terraform 还允许使用条件表达式，条件表达式与 CloudFormation `condition`函数类似，不同之处在于它们使用更传统的三元运算符语法。在以下示例中，两个表达式返回的结果完全相同。第二个例子是 Terraform 所说的 s pl at 表达式。星号会让 Terraform 循环浏览列表并仅使用每个项目的`id`属性来创建一个新列表。

```
resource "example_resource" "example_resource_name" {
  boolean_value  = var.value ? true : false
  numeric_value  = var.value > 0 ? 1 : 0
  string_value   = var.value == "change_me" ? "New value" : var.value
  string_value_2 = var.value != "change_me" ? var.value : "New value"
}
There are two ways to express for loops in a Terraform configuration:
resource "example_resource" "example_resource_name" {
  list_value   = [for object in var.ids : object.id]
  list_value_2 = var.ids[*].id
```

```
}
```

元参数

在前面的代码示例中，`list_value`和`list_value_2`被称为参数。你可能已经熟悉其中一些元参数了。Terraform 还有一些元参数，它们的作用就像参数一样，但有一些额外的功能：

- de pends_on 元参数与该属性非常相似。 CloudFormation DependsOn
- 提供者元参数允许您同时使用多个提供程序配置。
- 生命周期元参数允许您自定义资源设置，类似于中的删除和删除策略。 CloudFormation

其他元参数允许将函数和表达式功能直接添加到资源中。例如，count 元参数是同时创建多个相似资源的有用机制。以下示例演示了如何在不使用`count`元参数的情况下创建两个亚马逊弹性容器服务 (Amazon EKS) 集群。

```
resource "aws_eks_cluster" "example_0" {
  name     = "example_0"
  role_arn = aws_iam_role.cluster_role.arn
  vpc_config {
    endpoint_private_access = true
    endpoint_public_access  = true
    subnet_ids              = var.subnet_ids[0]
  }
}

resource "aws_eks_cluster" "example_1" {
  name     = "example_1"
  role_arn = aws_iam_role.cluster_role.arn
  vpc_config {
    endpoint_private_access = true
    endpoint_public_access  = true
    subnet_ids              = var.subnet_ids[1]
  }
}
```

以下示例演示如何使用`count`元参数创建两个 Amazon EKS 集群。

```
resource "aws_eks_cluster" "clusters" {
  count    = 2
  name     = "cluster_${count.index}"
```

```
  role_arn = aws_iam_role.cluster_role.arn
  vpc_config {
    endpoint_private_access = true
    endpoint_public_access  = true
    subnet_ids              = var.subnet_ids[count.index]
  }
}
```

要为每个单位命名，您可以访问资源块中的列表索引，网址为count.index。但是，如果你想创建多个稍微复杂一点的类似资源，该怎么办？ 这就是 for_ each 元参数的用武之地。for_each元参数与非常相似count，不同之处在于你传入的是列表或对象，而不是数字。Terraform 为列表或对象的每个成员创建一个新资源。它与您设置时类似count = length(list)，不同之处在于您可以访问列表的内容而不是循环索引。

这既适用于项目列表，也适用于单个对象。以下示例将创建两个以id-0和id-1为其的资源 IDs。

```
variable "ids" {
  default = [
    { id = "id-0" },
    { id = "id-1" },
  ]
}

resource "example_resource" "example_resource_name" {
  # If your list fails, you might have to call "toset" on it to convert it to a set
  for_each = toset(var.ids)
  id       = each.value
}
```

以下示例还将创建两个资源，一个用于贵宾犬 Sparky，另一个用于吉娃娃 Fluffy。

```
variable "dogs" {
  default = {
    poodle    = "Sparky"
    chihuahua = "Fluffy"
  }
}

resource "example_resource" "example_resource_name" {
  for_each = var.dogs
  breed    = each.key
  name     = each.value
```

```
}
```

就像你可以使用 count.index 访问 count.index 中的循环索引一样，你可以使用每个对象访问 for_each 循环中每个项目的键和值。由于 for_each 会迭代列表和对象，因此跟踪每个键和值可能会有些混乱。下表显示了使用 for_each 元参数的不同方式以及如何在每次迭代时引用这些值。

示例	**for_each** 类型	第一次迭代	第二次迭代
A	`["poodle", "chihuahua"]`	`each.key = "poodle"` `each.value = null`	`each.key = "chihuahua"` `each.value = null`
B	`[ { type = "poodle", name = "Sparky" }, { type = "chihuahua", name = "Fluffy" } ]`	`each.key = { type = "poodle", name = "Sparky" }` `each.value = null`	`each.key = { type = "chihuahua", name = "Fluffy" }` `each.value = null`
C	`{ poodle = "Sparky",`	`each.key = "poodle"`	`each.key = "chihuahua"`

示例	**for_each** 类型	第一次迭代	第二次迭代
	chihuahua = "Fluffy" }	each.value = "Sparky"	each.value = "Fluffy"

D	{
	
	dogs = {
	
	poodle = "Sparky",
	
	chihuahua = "Fluffy"
	
	},
	
	cats = {
	
	persian = "Felix",
	
	burmese = "Morris"
	
	}
	
	}
		each.key = "dogs"
		
		each.value = {
		
		poodle = "Sparky",
		
		chihuahua = "Fluffy"
		
		}
			each.key = "cats"
			
			each.value = {
			
			persian = "Felix",
			
			burmese = "Morris"
			
			}

示例	**for_each** 类型	第一次迭代	第二次迭代

E	{
	
	dogs = [
	
	{
	
	type =
	"poodle",
	
	name = "Sparky"
	
	},
	
	{
	
	type = "chihuahu
	a",
	
	name = "Fluffy"
	
	}
	
	],
	
	cats = [
	
	{
	
	type = "persian"
	,
	
	name = "Felix"
	
	},
	
	{
	
	type = "burmese"
	,
	
	name = "Morris"
		each.key = "dogs"
		
		each.value = [
		
		{
		
		type =
		"poodle",
		
		name = "Sparky"
		
		},
		
		{
		
		type = "chihuahu
		a",
		
		name = "Fluffy"
		
		}
		
		]
			each.key = "cats"
			
			each.value = [
			
			{
			
			type = "persian"
			,
			
			name = "Felix"
			
			},
			
			{
			
			type = "burmese"
			,
			
			name = "Morris"
			
			}
			
			]

示例	for_each 类型	第一次迭代	第二次迭代
	}] }		

因此var.animals，如果等于行 E，则可以使用以下代码为每只动物创建一个资源。

```
resource "example_resource" "example_resource_name" {
  for_each = var.animals
  type     = each.key
  breeds   = each.value[*].type
  names    = each.value[*].name
}
```

或者，您可以使用以下代码为每只动物创建两个资源。

```
resource "example_resource" "example_resource_name" {
  for_each = var.animals.dogs
  type     = "dogs"
  breeds   = each.value.type
  names    = each.value.name
}

resource "example_resource" "example_resource_name" {
  for_each = var.animals.cats
  type     = "cats"
  breeds   = each.value.type
  names    = each.value.name
}
```

常见问题解答

我应该什么时候用 Terraform 来代替？ CloudFormation

通常，如果您的工作负载主要基于 AWS，则可以 AWS CloudFormation 提供 Terraform 无法比拟的原生支持级别。但是，如果您的工作负载包含不少第三方进程，或者它们分散在多个云提供商中，那么 Terraform是您可能需要考虑的工具。

我什么时候应该用 the AWS CDK 来代替 CloudFormation？

当你使用时 AWS Cloud Development Kit (AWS CDK)，你也在使用 CloudFormation。 AWS CDK 允许您使用通用编程语言生成 CloudFormation 模板。如果您对它们 AWS CDK 支持的任何编程语言有经验，则 AWS CDK 可以减少生成 CloudFormation 模板所需的时间。

有没有像这样的工具可以生成 Terraform 配置？ AWS CDK

相比之下 AWS CDK，适用于 Terraform 的 CDK (CDKTF) 使用相同的构造库来配置资源，使用相同的 jsii 引擎来支持多种编程语言。你可以用它来生成 Terraform 配置，就像 AWS CDK 生成 CloudFormation 模板一样。

如何了解有关 Terraform 的更多信息？

有关高级 Terraform 概念的更多信息，请参阅 T er raform 文档。它还描述了所有主要提供商和开源模块的组件。

相关资源

AWS 文档

- AWS CDK 文档
- AWS CloudFormation 文档
- Terraform：超越基础知识 AWS（AWS 博客文章）

其他资源

- 适用于 Terraform 的 CDK 文档
- Terraform 文档

附录：Terraform 属性访问示例

资源

```
resource "aws_s3_bucket" "myS3Bucket" {
     bucket = "my-s3-bucket"
}

bucketName = aws_s3_bucket.myS3Bucket.bucket
```

数据来源

```
data "aws_s3_bucket" "myS3Bucket" {
     bucket = "my-s3-bucket"
}

bucketName = data.aws_s3_bucket.myS3Bucket.bucket
```

模块

```
module "eks" {
    source = "terraform-aws-modules/eks/aws"
    version = "20.2.1"
}

vpc_id = module.eks.vpc_id
```

Variable

```
variable "my_variable" = {
   default = "dog"
}

animalType = var.my_variable
```

本地

```
locals {
  type = "dog"
}

animalType = local.type
```

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	—	2024 年 3 月 29 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：将您的本地 Oracle 数据库迁移到兼容 Amazon Aurora PostgreSQL 的版本。
- 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：在中将您的本地 Oracle 数据库迁移到适用于 Oracle 的亚马逊关系数据库服务 (Amazon RDS) AWS 云。
- 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：将您的客户关系管理 (CRM) 系统迁移到 Salesforce.com。
- 更换主机（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：在中的 EC2 实例上将您的本地 Oracle 数据库迁移到 Oracle AWS 云。
- 重新定位（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。您可以将服务器从本地平台迁移到同一平台的云服务。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

参见托管服务。

ACID

参见原子性、一致性、隔离性、耐久性。

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。与主动-被动迁移相比，它更灵活，但需要更多的工作。

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

一个 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

参见人工智能。

AIOps

参见人工智能操作。

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 (I AM) 文档 AWS中的 AB AC。

权威数据源

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

坏机器人

旨在破坏个人或组织或对其造成伤害的机器人。

BCP

参见业务连续性计划。

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。另请参见字节顺序。

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的应用程序版本（蓝色），在另一个环境中运行新的应用程序版本（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互动的软件应用程序。有些机器人是有用或有益的，例如在互联网上索引信息的网络爬虫。其他一些被称为恶意机器人的机器人旨在破坏个人或组织或对其造成伤害。

僵尸网络

被恶意软件感染并受单方（称为机器人牧民或机器人操作员）控制的机器人网络。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）。

破碎的玻璃通道

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 Well -Architected 指南中的 “实施破碎玻璃程序” 指示 AWS 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

参见AWS 云采用框架。

金丝雀部署

向最终用户缓慢而渐进地发布版本。当你有信心时，你可以部署新版本并全部替换当前版本。

CCoE

参见云卓越中心。

CDC

请参阅变更数据捕获。

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弹性。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

查看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 云 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。云计算通常与边缘计算技术相关。

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

组织迁移到以下阶段时通常会经历四个阶段 AWS 云：

- 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 landing zone、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 云 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

参见配置管理数据库。

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 管道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。

配置偏差

对于工作负载，配置会从预期状态发生变化。这可能会导致工作负载变得不合规，而且通常是渐进的，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

参见计算机视觉。

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-Architecte 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 云 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个人。

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

参见数据库定义语言。

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 子字段使用多层人工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。

委托管理员

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。此账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。

后

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

参见环境。

侦测性控制

一种安全控制，在事件发生后进行检测、记录日志和发出警报。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

在星型架构中，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)

您用来最大限度地减少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 Fr ame AWS work 中的 “工作负载灾难恢复：云端 AWS 恢复”。

DML

参见数据库操作语言。

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

参见灾难恢复。

漂移检测

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

参见开发价值流映射。

E

EDA

参见探索性数据分析。

EDI

参见电子数据交换。

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。与云计算相比，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)

组织之间自动交换业务文档。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密文的计算过程。

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

参见服务端点。

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

参见企业资源规划。

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和创建数据可视化得以执行。

F

事实表

星形架构中的中心表。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。

失败得很快

一种使用频繁和增量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 云，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

参见分支。

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。

少量提示

在要求法学硕士执行类似任务之前，向其提供少量示例，以演示该任务和所需的输出。这种技术是情境学习的应用，模型可以从提示中嵌入的示例（镜头）中学习。对于需要特定格式、推理或领域知识的任务，Few-shot 提示可能非常有效。另请参见零镜头提示。

FGAC

请参阅精细的访问控制。

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，它使用连续的数据复制，通过更改数据捕获在尽可能短的时间内迁移数据，而不是使用分阶段的方法。目标是将停机时间降至最低。

FM

参见基础模型。

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

人工智能模型的子集，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工件，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 AI。

地理封锁

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，而基于主干的工作流程是现代的首选方法。

金色影像

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配置软件，并有助于提高设备制造运营的速度、可扩展性和生产力。

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

一项高级规则，可帮助管理各组织单位的资源、策略和合规性 (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。侦测性防护机制会检测策略违规和合规性问题，并生成警报以进行修复。它们通过使用 AWS Config、、Amazon、 AWS Security Hub GuardDuty AWS Trusted Advisor、Amazon Inspector 和自定义 AWS Lambda 支票来实现。

H

HA

参见高可用性。

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。

抵制数据

从用于训练机器学习模型的数据集中扣留的一部分带有标签的历史数据。通过将模型预测与抵制数据进行比较，您可以使用抵制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。

hypercare 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 hypercare 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

IaC

参见基础设施即代码。

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 云 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。

IIoT

参见工业物联网。

不可变的基础架构

一种为生产工作负载部署新基础架构，而不是更新、修补或修改现有基础架构的模型。不可变基础架构本质上比可变基础架构更一致、更可靠、更可预测。有关更多信息，请参阅 Well-Architected Framework 中的使用不可变基础架构 AWS 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

该术语由克劳斯·施瓦布（Klaus Schwab）于2016年推出，指的是通过连接、实时数据、自动化、分析和人工智能/机器学习的进步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置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 (IIoT)

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 (IIoT) 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。

IoT

参见物联网。

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。

大型语言模型 (LLM)

一种基于大量数据进行预训练的深度学习 AI 模型。法学硕士可以执行多项任务，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

见 7 R。

小端序系统

一个先存储最低有效字节的系统。另请参见字节顺序。

LLM

参见大型语言模型。

下层环境

参见环境。

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

参见分支。

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。恶意软件的示例包括病毒、蠕虫、勒索软件、特洛伊木马、间谍软件和键盘记录器。

托管服务

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。亚马逊简单存储服务 (Amazon S3) Service 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、记录和控制在车间将原材料转化为成品的生产过程。

MAP

参见迁移加速计划。

机制

一个完整的过程，在此过程中，您可以创建工具，推动工具的采用，然后检查结果以进行调整。机制是一种在运行过程中自我增强和改进的循环。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

参见制造执行系统。

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： EC2 使用 AWS 应用程序迁移服务重新托管向 Amazon 的迁移。

迁移组合评测（MPA）

一种在线工具，可提供信息，用于验证迁移到的业务案例。 AWS 云 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

用于将工作负载迁移到的方法 AWS 云。有关更多信息，请参阅此词汇表中的 “7 R” 条目，并参阅调动组织以加快大规模迁移。

ML

参见机器学习。

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，请参阅中的应用程序现代化策略。 AWS 云

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，请参阅中的评估应用程序的现代化准备情况 AWS 云。

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

参见迁移组合评估。

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？”

可变基础架构

一种用于更新和修改现有生产工作负载基础架构的模型。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

请参阅源站访问控制。

OAI

参见源访问身份。

OCM

参见组织变更管理。

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

参见运营集成。

OLA

参见运营层协议。

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

参见开放流程通信-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)

一份问题清单和相关的最佳实践，可帮助您理解、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 Well-Architecte AWS d Frame work 中的运营准备情况评估 (ORR)。

操作技术 (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

由此创建的跟踪 AWS CloudTrail ，用于记录组织 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

参见运营准备情况审查。

OT

参见运营技术。

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

查看个人身份信息。

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

参见可编程逻辑控制器。

PLM

参见产品生命周期管理。

policy

一个对象，可以在中定义权限（参见基于身份的策略）、指定访问条件（参见基于资源的策略）或定义组织中所有账户的最大权限 AWS Organizations （参见服务控制策略）。

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，则可以更轻松地实现微服务，并获得更好的性能和可扩展性。有关更多信息，请参阅在微服务中实现数据持久性。

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`或的查询条件`false`，通常位于子`WHERE`句中。

谓词下推

一种数据库查询优化技术，可在传输前筛选查询中的数据。这减少了必须从关系数据库检索和处理的数据量，并提高了查询性能。

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 文档中角色术语和概念中的主体。

通过设计保护隐私

一种在整个开发过程中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

一种安全控制措施，旨在防止部署不合规的资源。这些控件会在资源配置之前对其进行扫描。如果资源与控件不兼容，则不会对其进行配置。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)

在产品的整个生命周期中，从设计、开发和上市，到成长和成熟，再到衰落和移除，对产品进行数据和流程的管理。

生产环境

参见环境。

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。

提示链接

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，以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

一系列步骤，例如指令，用于访问 SQL 关系数据库系统中的数据。

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

参见负责任、负责、咨询、知情（RACI）。

RAG

请参见检索增强生成。

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

参见负责任、负责、咨询、知情（RACI）。

RCAC

请参阅行和列访问控制。

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。

重新架构师

见 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

见 7 R。

区域

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 AWS 区域 您的账户可以使用的账户。

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

见 7 R。

版本

在部署过程中，推动生产环境变更的行为。

搬迁

见 7 R。

更换平台

见 7 R。

回购

见 7 R。

故障恢复能力

应用程序抵御中断或从中断中恢复的能力。在中规划弹性时，高可用性和灾难恢复是常见的考虑因素。 AWS 云有关更多信息，请参阅AWS 云 弹性。

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

见 7 R。

退休

见 7 R。

检索增强生成（RAG）

一种生成式人工智能技术，其中法学硕士在生成响应之前引用其训练数据源之外的权威数据源。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据的过程。

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

参见恢复点目标。

RTO

参见恢复时间目标。

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 Management Console 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

参见监督控制和数据采集。

SCP

参见服务控制政策。

secret

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 Secret s Manager 密钥中有什么？ 在 Secrets Manager 文档中。

安全性源于设计

一种在整个开发过程中考虑安全性的系统工程方法。

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。安全控制主要有四种类型：预防性、侦测、响应式和主动式。

安全加固

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

一种预定义和编程的操作，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 实例或轮换证书。

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。

服务级别指示器 (SLI)

对服务性能方面的衡量，例如其错误率、可用性或吞吐量。

服务级别目标 (SLO)

代表服务运行状况的目标指标，由服务级别指标衡量。

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

参见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

参见服务级别协议。

SLI

参见服务级别指标。

SLO

参见服务级别目标。

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，请参阅中的分阶段实现应用程序现代化的方法。 AWS 云

恶作剧

参见单点故障。

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交易数据或测量数据，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。

监控和数据采集 (SCADA)

在制造业中，一种使用硬件和软件来监控有形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。你可以使用 Amazon S CloudWatch ynthetics 来创建这些测试。

系统提示符

一种向法学硕士提供上下文、说明或指导方针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

tags

键值对，用作组织资源的元数据。 AWS 标签可帮助您管理、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

参见环境。

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，该服务可以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。有关更多信息，请参阅量化深度学习系统中的不确定性指南。

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

参见环境。

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

一个 SQL 函数，用于对一组以某种方式与当前记录相关的行进行计算。窗口函数对于处理任务很有用，例如计算移动平均线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。

蠕虫

参见一次写入，多读。

WQF

参见AWS 工作负载资格框架。

一次写入，多次读取 (WORM)

一种存储模型，它可以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，但他们无法对其进行更改。这种数据存储基础架构被认为是不可变的。

Z

零日漏洞利用

一种利用未修补漏洞的攻击，通常是恶意软件。

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。

零镜头提示

向法学硕士提供执行任务的说明，但没有示例（镜头）可以帮助指导任务。法学硕士必须使用其预先训练的知识来处理任务。零镜头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 few-shot 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。