Terraform 入門： AWS CDK 和 AWS CloudFormation 專家指引

AWS 方案指引

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS 方案指引: Terraform 入門： AWS CDK 和 AWS CloudFormation 專家指引

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

簡介 1
CloudFormation 和地形術語 1
資源 3
提供者 5
使用地形別名 7
模組 10
呼叫模組 11
根模塊 11
狀態和後端 13
資料來源 15
變數、區域值和輸出 17
Variables 17
局部值 19
輸出值 19
函數、表達式和中繼引數 21
函數 21
表達式 21
中繼引數 22
常見問答集 28
我什麼時候應該使用地形而不是？ CloudFormation 28
我應該什麼時候使用 AWS CDK 而不是 CloudFormation？ 28
有沒有像生成 Terraform 配置 AWS CDK 的工具？ 28
如何進一步了解地形？ 28
相關資源 29
AWS 文件 29
其他資源 29
附錄：地形屬性訪問實例 30
資源 30
資料來源 30
模組 30
變數 30
區域 31
文件歷史紀錄 32
詞彙表 33

..... 33
A 33
B 36
C 37
D 40
E 43
F 45
G 46
H 47
I 48
L 50
M 51
O 55
P 57
Q 59
R 60
S 62
T 65
U 66
V 67
W 67
Z 68
..... lxix

開始使用地形表單：適用於 AWS CDK 和 AWS 專家的指導 CloudFormation

史蒂芬·古根海默, Amazon Web Services ()AWS

2024 年三月 (文件歷史記錄)

如果您在佈建雲端資源方面的經驗完全位於的範圍內 AWS，那麼您可能對AWS Cloud Development Kit (AWS CDK)和AWS CloudFormation之外的基礎結構即程式碼 (IaC) 工具的經驗有限。實際上，類似的工具（例如 Hashicorp 地形）可能對您來說完全不熟悉。但是，您進入雲端旅程越深，遇到 Terraform 就越難免。熟悉其核心概念將是您的優勢。

雖然 Terraform AWS CDK, 和 CloudFormation 實現類似的目標和共享許多核心概念, 有相當多的差異. 如果您是第一次接近地形，您可能不會為這些差異做好準備。畢竟 AWS CDK ， CloudFormation堆棧都基於其中 AWS 帳戶，因此以這種方式，它們與它們維護的大多數資源有直接關係。Terraform 並非以任何單一雲端供應商的環境為基礎。這使得它具有支持各種不同提供程序的靈活性，但它必須維護從遠程位置的資源。

本指南有助於揭開 Terraform 背後的核心概念，以幫助您處理任何 IaC 挑戰。它著重於 Terraform 如何使用概念（例如提供者，模塊和狀態文件）來佈建資源。它還使 Terraform 概念與如何執行類似 CloudFormation 操作進 AWS CDK 行對比。

> **Note**
>
> AWS CDK 可協助開發人員使用程式設計編碼語言部署 CloudFormation 堆疊。執行之後`cdk synth`，您的程式碼會轉換成 CloudFormation範本。從那時起，和之間的過程是相同 AWS CDK 的 CloudFormation。為了簡潔起見，本指南通常指的是 AWS IaC 過程，但比較對於. CloudFormation AWS CDK

CloudFormation 和地形術語

當與 AWS CDK 和比較 Terraform 時 CloudFormation，協調 IaC 的核心概念可能是困難的，因為用於描述它們不一致的術語。以下是這些術語以及本指南將如何參考它們：

- 堆棧 - 堆棧是部署到 CI/CD 管道中並可以作為單個單元進行跟踪的 IaC。雖然這個術語在中很常見 CloudFormation，但 Terraform 並沒有真正使用這個術語。Terraform 堆棧是一個部署的根模塊，其所有子模塊。不過，為了避免與模組一詞混淆，本指南使用術語堆疊來描述兩種工具的單一部署。

- 狀態 -狀態是 IaC 部署堆棧中所有當前跟踪的資源及其當前配置。如了解 Terraform 狀態和後端本節所述，地形使用術語狀態超過。 CloudFormation這是因為維護狀態在 Terraform 中更加明顯，但是跟踪和更新狀態對於而言同樣重要。 CloudFormation
- IaC 文件- IaC 文件是包含基礎設施代碼（IaC）語言的單個文件。 CloudFormation 將單一 CloudFormation 檔案稱為範本。但是，Terraform 中的模板和模板文件完全不同。相當於 Terraform 中的 CloudFormation 模板稱為配置文件。為了最大限度地減少本指南中的混淆，術語文件或 IaC 文件用於指示 CloudFormation 模板和 Terraform 配置文件。

下表比較用於 CloudFormation 和地形的術語。此表格的目的是顯示相似之處。這些不是 one-to-one 比較。每個概念 CloudFormation 和地形之間至少略有不同。本指南的相關章節將深入解釋概念。

CloudFormation 任期	地形術語	本指南章節
CDK 介面 (例如 i Bucket)	資料來源	了解地形資料來源
變更集	計畫	了解地形模塊
條件函數	條件式運算式	了解 Terraform 函數、表達式和中繼引數
DependsOn 屬性	depends_on 元參數	了解 Terraform 函數、表達式和中繼引數
內部函數	函數	了解 Terraform 函數、表達式和中繼引數
模組	模組	了解地形模塊
輸出	輸出值	瞭解地形變數、區域值和輸出
參數	Variables	瞭解地形變數、區域值和輸出
登錄檔	提供者	了解地形提供者
範本	組態檔案	全部

了解 Terraform 資源

同時存在 AWS CloudFormation 和 Terraform 的主要原因是雲端資源的建立和維護。但是，什麼是雲端資源？ CloudFormation 資源和 Terraform 資源是否相同？ 答案是...是和否。為了說明這一點，本指南提供使用 CloudFormation 和 Terraform 建立 Amazon Simple Storage Service (Amazon S3) 儲存貯體的範例。

下列 CloudFormation 程式碼範例會建立範例 Amazon S3 儲存貯體。

```
{
    "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"
            }
        }
    }
}
```

下列 Terraform 程式碼範例會建立相同的 Amazon S3 儲存貯體。

```
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"
  }
}
```

對於 Terraform，供應商會定義資源，然後開發人員宣告並設定這些資源。供應商是本指南在下一節討論的概念。Terraform 範例會為數個 S3 儲存貯體的設定建立完全獨立的資源。為設定建立個別資源不一定是 Terraform AWS 提供者處理 AWS 資源的典型方式。不過，此範例顯示重要的區別。雖然 CloudFormation 資源規格嚴格定義 CloudFormation 資源，但 Terraform 沒有這類需求。在 Terraform 中，資源的概念更模糊。

雖然工具在定義單一資源的確切護欄方面可能不同，但通常來說，雲端資源是存在於雲端中且可建立、更新或刪除的任何特定實體。因此，無論涉及多少資源，上述兩個範例都會在 中建立完全相同的設定 AWS 帳戶。

了解地形提供者

在 Terraform 中，提供程序是與雲提供商，第三方工具和其他 API 進行交互的插件。若要搭配使用 Terraform AWS，您可以使用與資源互動的AWS 提供者。 AWS

如果您從未使用AWS CloudFormation 登錄將第三方擴充功能併入部署堆疊中，則 Terraform 提供者可能需要一些習慣。因為 CloudFormation 是原生的 AWS，所以預設情況下， AWS 資源提供者已經存在。另一方面，Terraform 沒有單一的默認提供程序，因此不能假設給定資源的來源。這意味著需要在 Terraform 配置文件中聲明的第一件事就是資源的去向以及它們將如何到達那裡。

這種區別為不存在的 Terraform 增加了一層額外的複雜性。 CloudFormation但是，這種複雜性提供了更高的靈活性 您可以在單一 Terraform 模組中宣告多個提供者，然後建立的基礎資源可以作為相同部署層的一部分互動。

這可以在許多方面有用。供應商不一定要針對單獨的雲提供商。提供者可以代表雲端資源的任何來源。例如，以 Amazon Elastic Kubernetes Service（Amazon EKS）。佈建 Amazon EKS 叢集時，您可能想要使用 Helm 圖表來管理第三方擴充功能，並使用 Kubernetes 本身來管理網繭資源。由 AWS於 Helm 和 Kubernetes 都有自己的 Terraform 提供者，因此您可以同時佈建和整合這些資源，然後在這些資源之間傳遞值。

在 Terraform 的下列程式碼範例中， AWS 提供者會建立 Amazon EKS 叢集，然後產生的 Kubernetes 組態資訊會傳遞給 Helm 和 Kubernetes 提供者。

```
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"
```

```
  }
}
```

當涉及到兩個 IaC 工具時，有一個關於提供者的權衡。Terraform 完全依賴外部位於提供程序包，這些提供程序包是驅動其部署的引擎。 CloudFormation 內部支持所有主要 AWS 流程。使用 CloudFormation，僅當您想合併第三方擴展時，才需要擔心第三方提供商。每種方法都有優點和缺點。哪一種適合您超出了本指南的範圍，但是在評估兩種工具時記住差異很重要。

使用地形別名

在 Terraform 中，您可以將自定義配置傳遞給每個提供程序。那麼，如果您想在同一個模塊中使用多個提供程序配置怎麼辦？ 在這種情況下，您必須使用別名。 別名可協助您選取要在每個資源或每個模組層級使用的提供者。當您有多個相同提供者的執行個體時，您可以使用別名來定義非預設執行個體。例如，您的預設提供者執行個體可能是特定的 AWS 區域，但您可以使用別名來定義替代區域。

下列 Terraform 範例會示範如何使用別名來佈建不同的值區。 AWS 區域提供者的預設區域為`us-west-2`，但您可以使用 east 別名在中佈建資源`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"
}
```

如前面的範例所示，當您`alias`搭配中`provider`繼引數使用時，您可以為特定資源指定不同的提供者組態。在單一堆疊 AWS 區域 中佈建多個資源只是個開始。別名提供者在許多方面都非常方便。

例如，一次佈建多個 Kubernetes 叢集是很常見的。別名可協助您設定其他 Helm 和 Kubernetes 提供者，以便您可以針對不同的 Amazon EKS 資源以不同的方式使用這些第三方工具。下列 Terraform 程式碼範例說明如何使用別名來執行此工作。

```
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"
  }
}
```

了解地形模塊

在基礎結構即程式碼 (IaC) 的領域中，模組是隔離並封裝在一起以供重複使用的獨立程式碼區塊。模塊的概念是 Terraform 開發的一個不可避免的方面。如需詳細資訊，請參閱 Terraform 文件中的模組。 AWS CloudFormation 還支持模塊。如需詳細資訊，請參閱 AWS 雲端作業和移轉部落格中的簡介 AWS CloudFormation 模組。

Terraform 中模塊之間的主要區別在於 CloudFormation 模塊 CloudFormation 是通過使用特殊的資源類型（ ）`AWS::CloudFormation::ModuleVersion`導入的。在 Terraform 中，每個配置都至少有一個模塊，稱為根模塊。位於 main.tf 檔案或 Terraform 組態檔案中的檔案中的 Terraform 資源會被視為位於根模組中。然後根模塊可以調用其他模塊以包含在堆棧中。下列範例顯示使用開放原始碼 eks 模組佈建 Amazon Elastic Kubernetes Service (Amazon EKS) 叢集的根模組。

```
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)
  }
}
```

您可能已經注意到上述配置文件不包括 Pro AWS vider。這是因為模塊是獨立的，可以包含自己的提供程序。由於 Terraform 提供者是全域的，因此子模組的提供者可以在根模組中使用。儘管如此，所有模塊值都不是如此。默認情況下，模塊中的其他內部值僅限於該模塊，並且需要聲明為輸出才能在根模塊

中訪問。您可以利用開放原始碼模組來簡化堆疊中的資源建立作業。例如，eks 模組的功能不僅僅是佈建 EKS 叢集，還可以佈建一個功能完整的 Kubernetes 環境。使用它可以節省您編寫數十行額外的代碼，前提是 eks 模塊配置適合您的需求。

呼叫模組

您在 Terraform 部署期間執行的兩個主要 Terraform CLI 命令是地形初始化和地形套用。該`terraform init`命令執行的默認步驟之一是找到所有子模塊，並將它們作為依賴項導入到目`.terraform/modules`錄中。在開發過程中，無論何時添加新的外部來源模塊，都必須在使用`apply`命令之前重新初始化。當您聽到對 Terraform 模塊的引用時，它指的是此目錄中的軟件包。嚴格來說，您在代碼中聲明的模塊是調用模塊，因此在實踐中，module 關鍵字調用實際模塊，該模塊被存儲為依賴關係。

通過這種方式，調用模塊作為在部署發生時要替換的完整模塊的更簡潔的代表。您可以通過在堆棧中創建自己的模塊來利用這個想法，通過使用任何您想要的標準來強制執行資源的邏輯分離。請記住，這樣做的最終目標應該是降低堆棧複雜性。由於在模塊之間共享數據需要您從模塊內部輸出數據，因此有時過於依賴模塊可能會使事情過於複雜化。

根模塊

由於每個 Terraform 配置至少都有一個模塊，因此它可以幫助您檢查您將要處理的模塊的模塊屬性：根模塊。每當您在 Terraform 項目上工作時，根模塊都包含頂級目錄中的所有`.tf`（或`.tf.json`）文件。當您`terraform apply`在該頂級目錄中運行時，Terraform 會嘗試運行它在那裡找到的每個`.tf`文件。除非在其中一個頂層組態檔案中呼叫子目錄中的任何檔案，否則會略過這些檔案。

這在構建代碼方面提供了一些靈活性。這也是為什麼將 Terraform 部署引用為模塊而不是文件更準確的原因，因為單個過程可能涉及多個文件。Terraform 建議使用標準模組結構以取得最佳實務。但是，如果您要將任何文`.tf`件放在頂級目錄中，它將與其餘文件一起運行。事實上，模組中的所有頂層`.tf`檔案都會在您執行時部署`terraform apply`。那麼地形首先運行哪個文件？ 這個問題的答案非常重要。

Terraform 在初始化之後和堆棧部署之前執行了一系列步驟。首先，分析現有的組態，然後建立相依性圖形。依賴關係圖確定要求什麼樣的資源以及它們應該被解決的順序。例如，包含在其他資源中參照之屬性的資源，會在其相依資源之前處理。同樣地，使用`depends_on`參數明確宣告相依性的資源，也會在資源指定的資源之後進行處理。如果可能的話，Terraform 可以實現並行性並同時處理非依賴資源。您可以使用地形圖形命令在部署之前查看依賴關係圖。

建立相依性圖表之後，Terraform 會決定部署期間需要執行的動作。它的依賴關係圖與最新的狀態文件進行比較。此流\ 'b5\ '7b 的結果稱為「計劃」，非常類似於 CloudFormation變更集。您可以使用地形平面圖指令來查看目前的計畫。

作為最佳實踐，建議盡可能接近標準模塊結構。如果您的組態檔案變得太長而無法有效管理，而邏輯分離可能會簡化管理，您可以將程式碼分散到多個檔案中。請記住相依性圖表和計劃流程的運作方式，讓您的堆疊盡可能有效率地執行。

了解 Terraform 狀態和後端

基礎設施即程式碼 (IaC) 中最重要的概念之一是狀態的概念。IaC 服務會維持 狀態，可讓您在 IaC 檔案中宣告資源，而不必在每次部署時重新建立。IaC 檔案會在部署結束時記錄所有資源的狀態，以便將該狀態與目標狀態進行比較，如下次部署中所宣告。因此，如果目前狀態包含名為 的 Amazon Simple Storage Service (Amazon S3) 儲存貯體，`my-s3-bucket`而傳入的變更也包含相同的儲存貯體，則新程序會將找到的任何變更套用至現有儲存貯體，而不是嘗試建立新的儲存貯體。

下表提供一般 IaC 狀態程序的範例。

目前狀態	目標狀態	動作
沒有名為 的 S3 儲存貯體 `my-s3-bucket`	名為 的 S3 儲存貯體 `my-s3-bucket`	建立名為 的 S3 儲存貯體 `my-s3-bucket`
`my-s3-bucket` 未設定儲存貯體版本控制	`my-s3-bucket` 未設定儲存貯體版本控制	無動作
`my-s3-bucket` 未設定儲存貯體版本控制	`my-s3-bucket` 已設定儲存貯體版本控制	`my-s3-bucket` 將 設定為具有儲存貯體版本控制
`my-s3-bucket` 已設定儲存貯體版本控制	沒有名為 的 S3 儲存貯體 `my-s3-bucket`	嘗試刪除 `my-s3-bucket`

若要了解 AWS CloudFormation 和 Terraform 追蹤狀態的不同方式，請務必記住兩個工具之間的第一個基本差異：CloudFormation 託管在 內部 AWS 雲端，而 Terraform 基本上是遠端的。此事實可讓 CloudFormation 在內部維持狀態。您可以前往 CloudFormation 主控台並檢視指定堆疊的事件歷史記錄，但 CloudFormation 服務本身會為您強制執行狀態規則。

CloudFormation 在指定資源下操作的三種模式為 `Create`、 `Update`和 `Delete`。目前模式會根據上次部署中發生的情況來決定，否則無法受到影響。您或許可以手動更新 CloudFormation 資源，以影響決定哪個模式，但您無法將命令傳遞給 CloudFormation，其中指出「對於此資源，請在 `Create` 模式下操作」。

由於 Terraform 不在 中託管 AWS 雲端，因此維護狀態的程序必須更可設定。因此，Terraform 狀態會維持在自動產生的狀態檔案中。與 CloudFormation 相比，Terraform 開發人員必須更直接地處理狀態。要記住的重要事項是，追蹤狀態對這兩個工具同樣重要。

根據預設，Terraform 狀態檔案會儲存在本機執行 Terraform 堆疊的主目錄最上層。如果您從本機開發環境執行 `terraform apply`命令，您可以看到 Terraform 產生用於即時維護狀態的 terraform.tfstate 檔案。為了更好或更差，這可讓您比在 CloudFormation 中更能控制 Terraform 中的狀態。雖然您不應該直接更新狀態檔案，但您可以執行數個 Terraform CLI 命令，以便在部署之間更新狀態。例如，terraform import 可讓您將 Terraform 外部建立的資源新增至部署堆疊。相反地，您可以透過執行 terraform 狀態 rm 從狀態移除資源。

Terraform 需要將其狀態存放在某個位置，這會導致另一個不適用於 CloudFormation 的概念：後端。Terraform 後端是 Terraform 堆疊在部署後存放其狀態檔案的位置。這也是它預期在新部署開始時尋找狀態檔案的位置。當您在本機執行堆疊時，如上所述，您可以將 Terraform 狀態的副本保留在最上層本機目錄中。這稱為本機後端。

開發持續整合和持續部署 (CI/CD) 環境時，本機狀態檔案通常包含在 .gitignore 檔案中，以防止版本控制。然後，管道中不存在任何本機狀態檔案。為了正常運作，管道階段需要在某個位置找到正確的狀態檔案。 這就是 Terraform 組態檔案通常包含後端區塊的原因。後端區塊會向 Terraform 堆疊指出，除了自己的頂層目錄之外，還需要尋找某個位置才能尋找狀態檔案。

Terraform 後端幾乎可以位於任何地方：Amazon S3 儲存貯體、API 端點，甚至是遠端 Terraform 工作區。以下是存放在 Amazon S3 儲存貯體中的 Terraform 後端範例。

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

為了避免在 Terraform 組態檔案中存放敏感資訊，後端也支援部分組態。在先前的範例中，存取儲存貯體所需的登入資料不會出現在組態中。登入資料可以從環境變數或使用其他方式取得，例如 AWS Secrets Manager。如需詳細資訊，請參閱使用 AWS Secrets Manager 和 HashiCorp Terraform 保護敏感資料。

常見的後端案例是在本機環境中用於測試目的的本機後端。terraform.tfstate 檔案包含在 .gitignore 檔案中，因此不會推送到遠端儲存庫。然後，CI/CD 管道中的每個環境都會維護自己的後端。在此案例中，多個開發人員可能可以存取此遠端狀態，因此您想要保護狀態檔案的完整性。如果多個部署同時執行並更新狀態，狀態檔案可能會損毀。因此，在具有非本機後端的情況下，狀態檔案通常會在部署期間鎖定。

了解地形資料來源

部署堆疊依賴先前現有資源的資料是很常見的。大多數 IaC 工具都有一種導入由其他進程創建的資源的方法。這些匯入的資源通常是唯讀的 (雖然 IAM 角色是明顯的例外)，並且用來存取堆疊中資源所需的資料。 AWS CloudFormation 允許導入資源，但這個想法可以通過查看 AWS Cloud Development Kit (AWS CDK).

AWS CDK 可協助開發人員使用現有的程式設計語言產生 CloudFormation範本。 AWS CDK 作業的最終結果是中的匯入資源 CloudFormation。但是，與 AWS CDK 使用的語法可以更輕鬆地與 Terraform 進行比較。以下是使用匯入資源的範例 AWS CDK。

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

導入的資源通常是通過在您用來創建相同類型的新資源的同一個類上調用靜態方法來創建的。調用new Bucket(...將創建一個新的資源，並調用Bucket.fromBucketAttributes(...導入一個現有的。您可以將值區屬性的子集傳遞給函數， AWS CDK 以便找到正確的值區。不過，另一個不同之處在於，建立新值區會傳回Bucket類別的完整執行個體，其中包含所有可用的屬性和方法。匯入資源會傳回一個IBucket，此類型僅包含Bucket必須具有的屬性。雖然您可以從外部堆棧導入資源，但是您可以使用它執行的操作的選項是有限的。

在 Terraform 中，類似的目標是通過使用數據源來實現。大多數定義的 Terraform 資源都有一個隨附的資料來源。以下是 Terraform S3 儲存貯體資源後跟其對應資料來源的範例。

```
# 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"
}
```

這兩個項目之間的唯一區別是名稱前綴。如資料來源的文件所示，可傳遞至資料來源的可用參數少於資源。這是因為資源使用這些參數來宣告新 S3 儲存貯體的所有屬性，而資料來源只需要足夠的資訊才能唯一識別和匯入現有資源的資料。

Terraform 資源的語法與資料來源之間的相似性可能很方便，但也可能會有問題。對於新手來說，Terraform 開發人員通常會在其配置中意外使用數據源而不是資源。地形資料來源永遠是唯讀的。您可以使用它們來取代讀取動作的對應資源 (例如向其他資源提供 ID 名稱)。但是，您不能將它們用於編寫操作，這從根本上改變了基礎資源的某些方面。因此，您可以將 Terraform 資料來源視為基礎資源的複製版本。

與之前的 AWS CDK iBucket 範例類似，資料來源適用於唯讀案例。如果您需要從現有資源取得資料，但不需要在堆疊中維護該資源，請使用資料來源。這方面的一個很好的例子是當您創建使用該帳戶的默認 VPC 的 Amazon EC2 實例時。由於 VPC 已經存在，您只需要提取其資料即可。下列程式碼範例顯示如何使用資料來識別目標 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
}
```

瞭解地形變數、區域值和輸出

變數可讓程式碼區塊內的預留位置加強程式碼彈性。每當重複使用代碼時，變量可以代表不同的值。地形通過它們的模塊化範圍它的變量類型之間的區別。輸入變量是可以注入到模塊中的外部值，輸出值是可以在外部共享的內部值，並且局部值始終保持在其原始範圍內。

Variables

AWS CloudFormation 使用參數來表示可從一個堆疊部署設定和重設到下一個堆疊部署的自訂值。同樣地，地形使用輸入變數或變數。變量可以在 Terraform 配置文件中的任何地方聲明，並且通常使用所需的數據類型或默認值聲明。以下所有三個表達式都是有效的 Terraform 變量聲明。

```
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"
    }
  ]
}
```

要在配置中訪問斯帕克的品種，您可以使用該變量`var.dogs[0].breed`。如果變數沒有預設值且未分類為 Null，則必須為每個部署設定變數的值。否則，為變量設置新值是可選的。在根模組中，您可以在指令列、環境變數或 terrafor m.tfvars 檔案中設定目前的變數值。下列範例會示範如何在儲存在模組頂層目錄中的 terraform.tfvars 檔案中輸入變數值。

```
# terraform.tfvars
```

```
dogs = [
   {
      name  = "Sparky",
      breed = "poodle"
   },
   {
      name  = "Fluffy",
      breed = "chihuahua"
   }
]

random_number = 7

thing_i_made_up = "Kabibble"
```

dogs在此範例中，terraform.tfvars 檔案中的值會覆寫變數宣告中的預設值。如果您要在子模組中宣告變數，則可以直接在模組宣告區塊內設定變數值，如下列範例所示。

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

聲明變量時可以使用的其他一些參數包括：

- `sensitive`— 設定此項可`true`防止變數值暴露在 Terraform 流程輸出中。
- `nullable`— 將其設定為`true`允許變數沒有值。這對於未設置默認值的變量很方便。
- `description`— 將變數的描述新增至堆疊的中繼資料。
- `validation`— 設定變數的驗證規則。

Terraform 變量最方便的方面之一是能夠在變量聲明中添加一個或多個驗證對象。您可以使用驗證物件來新增變數必須通過的條件，否則部署失敗。您還可以設置自定義錯誤消息以在違反條件時顯示。

例如，您正在設定 Terraform 設定檔，讓團隊成員執行該檔案。在部署堆疊之前，團隊成員需要建立 terraform.tfvars 檔案來設定重要的組態值。為了提醒他們，您可以執行以下操作。

```
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-'"
  }
}
```

如此範例所示，您可以在單一變數內設定多個條件。地形只會顯示失敗狀況的錯誤訊息。通過這種方式，您可以在變量值上強制執行各種規則。如果變數值導致管線失敗，您將確切知道原因。

局部值

如果您想要別名的模組中有任何值，請使用`locals`關鍵字，而不是宣告永遠不會更新的預設變數。顧名思義，`locals`塊包含在內部範圍為該特定模塊的術語。如果您想要轉換字串值，例如在變數值中加入前置詞以供資源名稱使用，則使用本機值可能是個不錯的解決方案。單一`locals`區塊可以宣告模組的所有區域值，如下列範例所示。

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

請記住，當您訪問該值時，`locals`關鍵字變為單數，例如`local.LocalConfigId`。

輸出值

如果 Terraform 輸入變量就像 CloudFormation 參數，那麼你可以說 Terraform 輸出值就像輸出。CloudFormation兩者都用於公開部署堆疊中的值。但是，由於 Terraform 模組更根深蒂固於工具的結構中，因此 Terraform 輸出值也可用於將模組內的值公開給父模組或其他子模組，即使這些模組都位於相同的部署堆疊中。如果您正在構建兩個自定義模塊，並且第一個模塊需要訪問第二個模塊的 ID 值，那麼您需要將以下`output`塊添加到第二個模塊。

```
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
}
```

由於 Terraform 輸出值可以在同一堆疊中使用，因此您也可以使用output區塊中的sensitive屬性來抑制該值，使其不會顯示在堆疊輸出中。此外，區output塊可以使用precondition區塊的方式與變數使用validation區塊相同：以確保變數遵循特定的規則集。這有助於確保在繼續部署之前，模組內的所有值都如預期般存在。

```
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 函數、表達式和中繼引數

對使用宣告式組態檔案而非常見程式設計語言的 IaC 工具的批評是，它們使得實作自訂程式設計邏輯變得更加困難。在 Terraform 組態中，此問題是透過使用函數、表達式和中繼引數來解決。

函數

使用程式碼佈建基礎設施的絕佳優點之一是能夠儲存常見的工作流程，並不斷重複使用它們，通常每次都會傳遞不同的引數。Terraform 函數與 AWS CloudFormation 內部函數類似，雖然其語法更類似於以程式設計語言呼叫函數的方式。您可能已經在本指南的範例中注意到一些 Terraform 函數，例如 substr、concat、length 和 base64decode。如同具有內部函數的 CloudFormation，Terraform 具有一系列內建函數，可用於您的組態。例如，如果特定資源屬性採用非常大型的 JSON 物件，而該物件無法有效地直接貼到檔案中，您可以將物件放入 .json 檔案中，並使用 Terraform 函數來存取它。在下列範例中，`file`函數會以字串形式傳回檔案的內容，然後`jsondecode`函數將其轉換為物件類型。

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

表達式

Terraform 也允許條件式表達式，這些表達式類似於 CloudFormation `condition`函數，但它們使用較傳統的三元運算子語法。在下列範例中，兩個表達式會傳回完全相同的結果。第二個範例是 Terraform 呼叫潑濺表達式的方式。星號會導致 Terraform 循環瀏覽清單，並僅使用每個項目的 `id` 屬性來建立新的清單。

```
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
}
```

中繼引數

在先前的程式碼範例中，`list_value`和 `list_value_2` 稱為引數。您可能已經熟悉其中一些中繼引數。Terraform 也有幾個中繼引數，其作用就像引數，但具有一些額外的功能：

- depends_on 中繼引數非常類似於 CloudFormation DependsOn 屬性。
- 提供者中繼引數可讓您一次使用多個提供者組態。
- 生命週期中繼引數可讓您自訂資源設定，類似於 CloudFormation 中的移除和刪除政策。

其他中繼引數可讓函數和表達式功能直接新增至資源。例如，計數中繼引數是同時建立多個類似資源的實用機制。下列範例示範如何在不使用中繼引數的情況下建立兩個 Amazon Elastic Container Service (Amazon EKS) `count` 叢集。

```
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]
  }
}
```

下列範例示範如何使用`count`中繼引數來建立兩個 Amazon EKS 叢集。

```
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]
  }
}
```

若要為每個單位名稱命名，您可以在 存取資源區塊中的清單索引count.index。但是，如果您想要建立多個比較複雜的類似資源，該怎麼辦？ 這就是 for_each 中繼引數所在的位置。for_each 中繼引數非常類似 count，但您傳入清單或物件而非數字。Terraform 會為清單或物件的每個成員建立新的資源。它類似於您設定 count = length(list)，但您可以存取清單的內容，而不是迴圈索引。

這適用於項目清單或單一物件。下列範例會建立兩個具有 id-0和 id-1作為其 IDs的資源。

```
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
}
```

下列範例也會建立兩個資源，一個用於 Sparky 木桿，另一個用於 chihuahua Fluffy。

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

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

就像您可以使用 count.index 存取計數中的迴圈索引一樣，您可以使用每個物件存取 for_each 迴圈中每個項目的索引鍵和值。由於 for_each 會逐一查看清單和物件，因此每個索引鍵和值都可能有些難以追蹤。下表顯示您可以使用 for_each 中繼引數的不同方式，以及如何在每次反覆運算時參考這些值。

範例	**for_each** 類型	第一次反覆運算	第二次反覆運算
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.value = "Sparky"`	`each.key = "chihuahua"` `each.value = "Fluffy"`

範例	for_each 類型	第一次反覆運算	第二次反覆運算
	chihuahua = "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"
			
			}

範例	for_each 類型	第一次反覆運算	第二次反覆運算

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"
			
			}
			
			]

範例	for_each 類型	第一次反覆運算	第二次反覆運算
	}] }		

因此，如果 var.animals 等於資料列 E，您可以使用下列程式碼，為每個動物建立一個資源。

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

或者，您可以使用下列程式碼，為每個動物建立兩個資源。

```
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
}
```

常見問答集

我什麼時候應該使用地形而不是？ CloudFormation

一般而言，如果您的工作負載主要基於 AWS，則會 AWS CloudFormation 提供 Terraform 無法比對的原生支援層級。但是，如果您的工作負載包含不少第三方程序，或者它們分散在多個雲端提供者之間，則 Terraform 是您可能需要考慮的工具。

我應該什麼時候使用 AWS CDK 而不是 CloudFormation？

當您使用時 AWS Cloud Development Kit (AWS CDK)，您也在使用 CloudFormation. AWS CDK 可讓您使用通用程式設計語言來產生 CloudFormation 範本。如果您對 AWS CDK 支援的任何程式設計語言有經驗，則 AWS CDK 可以減少產生 CloudFormation 範本所需的時間。

有沒有像生成 Terraform 配置 AWS CDK 的工具？

相較於 AWS CDK，地形的 CDK (CDKTF) 使用相同的建構程式庫來佈建資源，而相同的 jsii 引擎來支援多種程式設計語言。您可以使用它來產生 Terraform 組態，方法與產 AWS CDK 生 CloudFormation 範本相同。

如何進一步了解地形？

如需進階地形概念的詳細資訊，請參閱 Ter raform 文件。它還描述了所有主要提供程序和開源模塊的組件。

相關資源

AWS 文件

- AWS CDK 文件
- AWS CloudFormation 文件
- 地形：超越基礎與 AWS（AWS 博客文章）

其他資源

- 用於地形文件的 CDK
- 地形文件

附錄：地形屬性訪問實例

資源

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

bucketName = aws_s3_bucket.myS3Bucket.bucket
```

資料來源

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

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

模組

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

vpc_id = module.eks.vpc_id
```

變數

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

animalType = var.my_variable
```

區域

```
locals {
  type = "dog"
}

animalType = local.type
```

文件歷史紀錄

下表描述了本指南的重大變更。如果您想收到有關未來更新的通知，可以訂閱 RSS 摘要。

變更	描述	日期
初次出版	—	2024年3月29 日

AWS 規範性指引詞彙表

以下是 AWS Prescriptive Guidance 提供的策略、指南和模式中常用的術語。若要建議項目，請使用詞彙表末尾的提供意見回饋連結。

數字

7 R

將應用程式移至雲端的七種常見遷移策略。這些策略以 Gartner 在 2011 年確定的 5 R 為基礎，包括以下內容：

- 重構/重新架構 – 充分利用雲端原生功能來移動應用程式並修改其架構，以提高敏捷性、效能和可擴展性。這通常涉及移植作業系統和資料庫。範例：將您的現場部署 Oracle 資料庫遷移至 Amazon Aurora PostgreSQL 相容版本。
- 平台轉換 (隨即重塑) – 將應用程式移至雲端，並引入一定程度的優化以利用雲端功能。範例：將您的現場部署 Oracle 資料庫遷移至 中的 Amazon Relational Database Service (Amazon RDS) for Oracle AWS 雲端。
- 重新購買 (捨棄再購買) – 切換至不同的產品，通常從傳統授權移至 SaaS 模型。範例：將您的客戶關係管理 (CRM) 系統遷移至 Salesforce.com。
- 主機轉換 (隨即轉移) – 將應用程式移至雲端，而不進行任何變更以利用雲端功能。範例：將您的現場部署 Oracle 資料庫遷移至 中 EC2 執行個體上的 Oracle AWS 雲端。
- 重新放置 (虛擬機器監視器等級隨即轉移) – 將基礎設施移至雲端，無需購買新硬體、重寫應用程式或修改現有操作。您可以將伺服器從內部部署平台遷移到相同平台的雲端服務。範例：將 Microsoft Hyper-V應用程式遷移至 AWS。
- 保留 (重新檢視) – 將應用程式保留在來源環境中。其中可能包括需要重要重構的應用程式，且您希望將該工作延遲到以後，以及您想要保留的舊版應用程式，因為沒有業務理由來進行遷移。
- 淘汰 – 解除委任或移除來源環境中不再需要的應用程式。

A

ABAC

請參閱屬性型存取控制。

抽象服務

請參閱 受管服務。

ACID

請參閱原子性、一致性、隔離性、耐久性。

主動-主動式遷移

一種資料庫遷移方法，其中來源和目標資料庫保持同步 (透過使用雙向複寫工具或雙重寫入操作)，且兩個資料庫都在遷移期間處理來自連接應用程式的交易。此方法支援小型、受控制批次的遷移，而不需要一次性切換。它更靈活，但比主動-被動遷移需要更多工作。

主動-被動式遷移

一種資料庫遷移方法，其中來源和目標資料庫會保持同步，但只有來源資料庫會在資料複寫至目標資料庫時處理來自連線應用程式的交易。目標資料庫在遷移期間不接受任何交易。

彙總函數

在一組資料列上運作的 SQL 函數，會計算群組的單一傳回值。彙總函數的範例包括 SUM和 MAX。

AI

請參閱人工智慧。

AIOps

請參閱人工智慧操作。

匿名化

永久刪除資料集中個人資訊的程序。匿名化有助於保護個人隱私權。匿名資料不再被視為個人資料。

反模式

經常用於重複性問題的解決方案，其中解決方案具有反生產力、無效或比替代解決方案更有效。

應用程式控制

一種安全方法，僅允許使用核准的應用程式，以協助保護系統免受惡意軟體攻擊。

應用程式組合

有關組織使用的每個應用程式的詳細資訊的集合，包括建置和維護應用程式的成本及其商業價值。此資訊是產品組合探索和分析程序的關鍵，有助於識別要遷移、現代化和優化的應用程式並排定其優先順序。

人工智慧 (AI)

電腦科學領域，致力於使用運算技術來執行通常與人類相關的認知功能，例如學習、解決問題和識別模式。如需詳細資訊，請參閱什麼是人工智慧？

人工智慧操作 (AIOps)

使用機器學習技術解決操作問題、減少操作事件和人工干預以及提高服務品質的程序。如需有關如何在 AWS 遷移策略中使用 AIOps 的詳細資訊，請參閱操作整合指南。

非對稱加密

一種加密演算法，它使用一對金鑰：一個用於加密的公有金鑰和一個用於解密的私有金鑰。您可以共用公有金鑰，因為它不用於解密，但對私有金鑰存取應受到高度限制。

原子性、一致性、隔離性、耐久性 (ACID)

一組軟體屬性，即使在出現錯誤、電源故障或其他問題的情況下，也能確保資料庫的資料有效性和操作可靠性。

屬性型存取控制 (ABAC)

根據使用者屬性 (例如部門、工作職責和團隊名稱) 建立精細許可的實務。如需詳細資訊，請參閱《 AWS Identity and Access Management (IAM) 文件》中的 ABAC for AWS。

授權資料來源

存放主要版本資料的位置，被視為最可靠的資訊來源。您可以將授權資料來源中的資料複製到其他位置，以處理或修改資料，例如匿名、修訂或假名化資料。

可用區域

中的不同位置 AWS 區域 ，可隔離其他可用區域中的故障，並提供相同區域中其他可用區域的低成本、低延遲網路連線。

AWS 雲端採用架構 (AWS CAF)

的指導方針和最佳實務架構 AWS ，可協助組織制定高效且有效的計劃，以成功地移至雲端。 AWS CAF 將指導方針組織到六個重點領域：業務、人員、治理、平台、安全和營運。業務、人員和控管層面著重於業務技能和程序；平台、安全和操作層面著重於技術技能和程序。例如，人員層面針對處理人力資源 (HR)、人員配備功能和人員管理的利害關係人。為此， AWS CAF 為人員開發、訓練和通訊提供指引，協助組織做好成功採用雲端的準備。如需詳細資訊，請參閱 AWS CAF 網站和 AWS CAF 白皮書。

AWS 工作負載資格架構 (AWS WQF)

評估資料庫遷移工作負載、建議遷移策略並提供工作預估值的工具。 AWS WQF 隨附於 AWS Schema Conversion Tool (AWS SCT)。它會分析資料庫結構描述和程式碼物件、應用程式程式碼、相依性和效能特性，並提供評估報告。

B

錯誤的機器人

旨在中斷或傷害個人或組織的機器人。

BCP

請參閱業務持續性規劃。

行為圖

資源行為的統一互動式檢視，以及一段時間後的互動。您可以將行為圖與 Amazon Detective 搭配使用來檢查失敗的登入嘗試、可疑的 API 呼叫和類似動作。如需詳細資訊，請參閱偵測文件中的行為圖中的資料。

大端序系統

首先儲存最高有效位元組的系統。另請參閱 Endianness。

二進制分類

預測二進制結果的過程 (兩個可能的類別之一)。例如，ML 模型可能需要預測諸如「此電子郵件是否是垃圾郵件？」等問題 或「產品是書還是汽車？」

Bloom 篩選條件

一種機率性、記憶體高效的資料結構，用於測試元素是否為集的成員。

藍/綠部署

一種部署策略，您可以在其中建立兩個不同但相同的環境。您可以在一個環境 （藍色） 中執行目前的應用程式版本，並在另一個環境 （綠色） 中執行新的應用程式版本。此策略可協助您快速復原，並將影響降至最低。

機器人

透過網際網路執行自動化任務並模擬人類活動或互動的軟體應用程式。有些機器人有用或有益，例如在網際網路上為資訊編製索引的 Web 爬蟲程式。有些其他機器人稱為惡意機器人，旨在中斷或傷害個人或組織。

殭屍網路

受到惡意軟體感染且受單一方控制之機器人的網路，稱為機器人繼承器或機器人運算子。殭屍網路是擴展機器人及其影響的最佳已知機制。

分支

程式碼儲存庫包含的區域。儲存庫中建立的第一個分支是主要分支。您可以從現有分支建立新分支，然後在新分支中開發功能或修正錯誤。您建立用來建立功能的分支通常稱為功能分支。當準備好發佈功能時，可以將功能分支合併回主要分支。如需詳細資訊，請參閱關於分支 (GitHub 文件)。

碎片存取

在特殊情況下，以及透過核准的程序，讓使用者快速存取他們通常無權存取 AWS 帳戶 的 。如需詳細資訊，請參閱 Well-Architected 指南中的 AWS 實作打破玻璃程序指標。

棕地策略

環境中的現有基礎設施。對系統架構採用棕地策略時，可以根據目前系統和基礎設施的限制來設計架構。如果正在擴展現有基礎設施，則可能會混合棕地和綠地策略。

緩衝快取

儲存最常存取資料的記憶體區域。

業務能力

業務如何創造價值 (例如，銷售、客戶服務或營銷)。業務能力可驅動微服務架構和開發決策。如需詳細資訊，請參閱在 AWS上執行容器化微服務白皮書的圍繞業務能力進行組織部分。

業務連續性規劃 (BCP)

一種解決破壞性事件 (如大規模遷移) 對營運的潛在影響並使業務能夠快速恢復營運的計畫。

C

CAF

請參閱AWS 雲端採用架構。

Canary 部署

版本對最終使用者的緩慢和增量版本。當您有信心時，您可以部署新版本並完全取代目前的版本。

CCoE

請參閱 Cloud Center of Excellence。

CDC

請參閱變更資料擷取。

變更資料擷取 (CDC)

追蹤對資料來源 (例如資料庫表格) 的變更並記錄有關變更的中繼資料的程序。您可以將 CDC 用於各種用途，例如稽核或複寫目標系統中的變更以保持同步。

混沌工程

故意引入故障或破壞性事件，以測試系統的彈性。您可以使用 AWS Fault Injection Service (AWS FIS) 執行實驗，為您的 AWS 工作負載帶來壓力，並評估其回應。

CI/CD

請參閱持續整合和持續交付。

分類

有助於產生預測的分類程序。用於分類問題的 ML 模型可預測離散值。離散值永遠彼此不同。例如，模型可能需要評估影像中是否有汽車。

用戶端加密

在目標 AWS 服務 接收資料之前，在本機加密資料。

雲端卓越中心 (CCoE)

一個多學科團隊，可推動整個組織的雲端採用工作，包括開發雲端最佳實務、調動資源、制定遷移時間表以及領導組織進行大規模轉型。如需詳細資訊，請參閱 AWS 雲端 企業策略部落格上的 CCoE 文章。

雲端運算

通常用於遠端資料儲存和 IoT 裝置管理的雲端技術。雲端運算通常連接到邊緣運算技術。

雲端操作模型

在 IT 組織中，用於建置、成熟和最佳化一或多個雲端環境的操作模型。如需詳細資訊，請參閱建置您的雲端操作模型。

採用雲端階段

組織在遷移至 時通常會經歷的四個階段 AWS 雲端：

- 專案 – 執行一些與雲端相關的專案以進行概念驗證和學習用途
- 基礎 – 進行基礎投資以擴展雲端採用 (例如，建立登陸區域、定義 CCoE、建立營運模型)

- 遷移 – 遷移個別應用程式
- 重塑 – 優化產品和服務，並在雲端中創新

這些階段由 Stephen Orban 在部落格文章 The Journey Toward Cloud-First 和 Enterprise Strategy 部落格上的採用階段中定義。 AWS 雲端 如需有關它們如何與 AWS 遷移策略關聯的資訊，請參閱遷移整備指南。

CMDB

請參閱組態管理資料庫。

程式碼儲存庫

透過版本控制程序來儲存及更新原始程式碼和其他資產 (例如文件、範例和指令碼) 的位置。常見的雲端儲存庫包括 GitHub或 Bitbucket Cloud。程式碼的每個版本都稱為分支。在微服務結構中，每個儲存庫都專用於單個功能。單一 CI/CD 管道可以使用多個儲存庫。

冷快取

一種緩衝快取，它是空的、未填充的，或者包含過時或不相關的資料。這會影響效能，因為資料庫執行個體必須從主記憶體或磁碟讀取，這比從緩衝快取讀取更慢。

冷資料

很少存取且通常是歷史資料的資料。查詢這類資料時，通常可接受慢查詢。將此資料移至效能較低且成本較低的儲存層或類別，可以降低成本。

電腦視覺 (CV)

使用機器學習從數位影像和影片等視覺化格式分析和擷取資訊的 AI 欄位。例如，Amazon SageMaker AI 提供 CV 的影像處理演算法。

組態偏離

對於工作負載，組態會從預期狀態變更。這可能會導致工作負載變得不合規，而且通常是漸進和無意的。

組態管理資料庫 (CMDB)

儲存和管理有關資料庫及其 IT 環境的資訊的儲存庫，同時包括硬體和軟體元件及其組態。您通常在遷移的產品組合探索和分析階段使用 CMDB 中的資料。

一致性套件

您可以組合的 AWS Config 規則和修補動作集合，以自訂您的合規和安全檢查。您可以使用 YAML 範本，將一致性套件部署為 AWS 帳戶 和 區域中或整個組織的單一實體。如需詳細資訊，請參閱 AWS Config 文件中的一致性套件。

持續整合和持續交付 (CI/CD)

自動化軟體發行程序的來源、建置、測試、暫存和生產階段的程序。CI/CD 通常被描述為管道。CI/CD 可協助您將程序自動化、提升生產力、改善程式碼品質以及加快交付速度。如需詳細資訊，請參閱持續交付的優點。CD 也可表示持續部署。如需詳細資訊，請參閱持續交付與持續部署。

CV

請參閱電腦視覺。

D

靜態資料

網路中靜止的資料，例如儲存中的資料。

資料分類

根據重要性和敏感性來識別和分類網路資料的程序。它是所有網路安全風險管理策略的關鍵組成部分，因為它可以協助您確定適當的資料保護和保留控制。資料分類是 AWS Well-Architected Framework 中安全支柱的元件。如需詳細資訊，請參閱資料分類。

資料偏離

生產資料與用於訓練 ML 模型的資料之間有意義的變化，或輸入資料隨時間有意義的變更。資料偏離可以降低 ML 模型預測的整體品質、準確性和公平性。

傳輸中的資料

在您的網路中主動移動的資料，例如在網路資源之間移動。

資料網格

架構架構，提供分散式、分散式資料擁有權與集中式管理。

資料最小化

僅收集和處理嚴格必要資料的原則。在 中實作資料最小化 AWS 雲端 可以降低隱私權風險、成本和分析碳足跡。

資料周邊

AWS 環境中的一組預防性防護機制，可協助確保只有信任的身分才能從預期的網路存取信任的資源。如需詳細資訊，請參閱在 上建置資料周邊 AWS。

資料預先處理

將原始資料轉換成 ML 模型可輕鬆剖析的格式。預處理資料可能意味著移除某些欄或列，並解決遺失、不一致或重複的值。

資料來源

在整個資料生命週期中追蹤資料的原始伺服器和歷史記錄的程序，例如資料的產生、傳輸和儲存方式。

資料主體

正在收集和處理其資料的個人。

資料倉儲

支援商業智慧的資料管理系統，例如 分析。資料倉儲通常包含大量歷史資料，通常用於查詢和分析。

資料庫定義語言 (DDL)

用於建立或修改資料庫中資料表和物件之結構的陳述式或命令。

資料庫處理語言 (DML)

用於修改 (插入、更新和刪除) 資料庫中資訊的陳述式或命令。

DDL

請參閱資料庫定義語言。

深度整體

結合多個深度學習模型進行預測。可以使用深度整體來獲得更準確的預測或估計預測中的不確定性。

深度學習

一個機器學習子領域，它使用多層人工神經網路來識別感興趣的輸入資料與目標變數之間的對應關係。

深度防禦

這是一種資訊安全方法，其中一系列的安全機制和控制項會在整個電腦網路中精心分層，以保護網路和其中資料的機密性、完整性和可用性。當您在 上採用此策略時 AWS，您可以在 AWS Organizations 結構的不同層新增多個控制項，以協助保護資源。例如，defense-in-depth方法可能會結合多重要素驗證、網路分割和加密。

委派的管理員

在 中 AWS Organizations，相容的服務可以註冊 AWS 成員帳戶來管理組織的帳戶，並管理該服務的許可。此帳戶稱為該服務的委派管理員。如需詳細資訊和相容服務清單，請參閱 AWS Organizations 文件中的可搭配 AWS Organizations運作的服務。

部署

在目標環境中提供應用程式、新功能或程式碼修正的程序。部署涉及在程式碼庫中實作變更，然後在應用程式環境中建置和執行該程式碼庫。

開發環境

請參閱 環境。

偵測性控制

一種安全控制，用於在事件發生後偵測、記錄和提醒。這些控制是第二道防線，提醒您注意繞過現有預防性控制的安全事件。如需詳細資訊，請參閱在 AWS上實作安全控制中的偵測性控制。

開發值串流映射 (DVSM)

一種程序，用於識別對軟體開發生命週期中的速度和品質造成負面影響的限制並排定優先順序。DVSM 擴展了最初專為精簡製造實務設計的價值串流映射程序。它著重於透過軟體開發程序建立和移動價值所需的步驟和團隊。

數位分身

真實世界系統的虛擬呈現，例如建築物、工廠、工業設備或生產線。數位分身支援預測性維護、遠端監控和生產最佳化。

維度資料表

在星星結構描述中，較小的資料表包含有關事實資料表中量化資料的資料屬性。維度資料表屬性通常是文字欄位或離散數字，其行為類似於文字。這些屬性通常用於查詢限制、篩選和結果集標記。

災難

防止工作負載或系統在其主要部署位置實現其業務目標的事件。這些事件可能是自然災難、技術故障或人為動作的結果，例如意外設定錯誤或惡意軟體攻擊。

災難復原 (DR)

您用來將災難造成的停機時間和資料遺失降至最低的策略和程序。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的 上的工作負載災難復原 AWS：雲端中的復原。

DML

請參閱資料庫處理語言。

領域驅動的設計

一種開發複雜軟體系統的方法，它會將其元件與每個元件所服務的不斷發展的領域或核心業務目標相關聯。Eric Evans 在其著作 Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston: Addison-Wesley Professional, 2003) 中介紹了這一概念。如需有關如何將領域驅動的設計與 strangler fig 模式搭配使用的資訊，請參閱使用容器和 Amazon API Gateway 逐步現代化舊版 Microsoft ASP.NET (ASMX) Web 服務。

DR

請參閱災難復原。

偏離偵測

追蹤與基準組態的偏差。例如，您可以使用 AWS CloudFormation 來偵測系統資源中的偏離，也可以使用 AWS Control Tower 來偵測登陸區域中可能影響控管要求合規性的變更。 https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-stack-drift.html

DVSM

請參閱開發值串流映射。

E

EDA

請參閱探索性資料分析。

EDI

請參閱電子資料交換。

邊緣運算

提升 IoT 網路邊緣智慧型裝置運算能力的技術。與雲端運算相比，邊緣運算可以減少通訊延遲並改善回應時間。

電子資料交換 (EDI)

在組織之間自動交換商業文件。如需詳細資訊，請參閱什麼是電子資料交換。

加密

將人類可讀取的純文字資料轉換為加密文字的運算程序。

加密金鑰

由加密演算法產生的隨機位元的加密字串。金鑰長度可能有所不同，每個金鑰的設計都是不可預測且唯一的。

端序

位元組在電腦記憶體中的儲存順序。大端序系統首先儲存最高有效位元組。小端序系統首先儲存最低有效位元組。

端點

請參閱 服務端點。

端點服務

您可以在虛擬私有雲端 (VPC) 中託管以與其他使用者共用的服務。您可以使用 建立端點服務，AWS PrivateLink 並將許可授予其他 AWS 帳戶 或 AWS Identity and Access Management (IAM) 委託人。這些帳戶或主體可以透過建立介面 VPC 端點私下連接至您的端點服務。如需詳細資訊，請參閱 Amazon Virtual Private Cloud (Amazon VPC) 文件中的建立端點服務。

企業資源規劃 (ERP)

一種系統，可自動化和管理企業的關鍵業務流程 （例如會計、MES 和專案管理）。

信封加密

使用另一個加密金鑰對某個加密金鑰進行加密的程序。如需詳細資訊，請參閱 AWS Key Management Service (AWS KMS) 文件中的信封加密。

環境

執行中應用程式的執行個體。以下是雲端運算中常見的環境類型：

- 開發環境 – 執行中應用程式的執行個體，只有負責維護應用程式的核心團隊才能使用。開發環境用來測試變更，然後再將開發環境提升到較高的環境。此類型的環境有時稱為測試環境。
- 較低的環境 – 應用程式的所有開發環境，例如用於初始建置和測試的開發環境。
- 生產環境 – 最終使用者可以存取的執行中應用程式的執行個體。在 CI/CD 管道中，生產環境是最後一個部署環境。
- 較高的環境 – 核心開發團隊以外的使用者可存取的所有環境。這可能包括生產環境、生產前環境以及用於使用者接受度測試的環境。

epic

在敏捷方法中，有助於組織工作並排定工作優先順序的功能類別。epic 提供要求和實作任務的高層級描述。例如， AWS CAF 安全概念包括身分和存取管理、偵測控制、基礎設施安全、資料保護和事件回應。如需有關 AWS 遷移策略中的 Epic 的詳細資訊，請參閱計畫實作指南。

ERP

請參閱企業資源規劃。

探索性資料分析 (EDA)

分析資料集以了解其主要特性的過程。您收集或彙總資料，然後執行初步調查以尋找模式、偵測異常並檢查假設。透過計算摘要統計並建立資料可視化來執行 EDA。

F

事實資料表

星狀結構描述中的中央資料表。它存放有關業務操作的量化資料。一般而言，事實資料表包含兩種類型的資料欄：包含度量的資料，以及包含維度資料表外部索引鍵的資料欄。

快速失敗

一種使用頻繁和增量測試來縮短開發生命週期的理念。這是敏捷方法的關鍵部分。

故障隔離界限

在 中 AWS 雲端，像是可用區域 AWS 區域、控制平面或資料平面等邊界會限制故障的影響，並有助於改善工作負載的彈性。如需詳細資訊，請參閱AWS 故障隔離界限。

功能分支

請參閱分支。

特徵

用來進行預測的輸入資料。例如，在製造環境中，特徵可能是定期從製造生產線擷取的影像。

功能重要性

特徵對於模型的預測有多重要。這通常表示為可以透過各種技術來計算的數值得分，例如 Shapley Additive Explanations (SHAP) 和積分梯度。如需詳細資訊，請參閱 的機器學習模型可解譯性 AWS。

特徵轉換

優化 ML 程序的資料，包括使用其他來源豐富資料、調整值、或從單一資料欄位擷取多組資訊。這可讓 ML 模型從資料中受益。例如，如果將「2021-05-27 00:15:37」日期劃分為「2021」、「五月」、「週四」和「15」，則可以協助學習演算法學習與不同資料元件相關聯的細微模式。

少量擷取提示

在要求 LLM 執行類似的任務之前，提供少量示範任務和所需輸出的範例。此技術是內容內學習的應用程式，其中模型會從內嵌在提示中的範例 (快照) 中學習。對於需要特定格式、推理或網域知識的任務，少量的提示非常有效。另請參閱零鏡頭提示。

FGAC

請參閱精細存取控制。

精細存取控制 (FGAC)

使用多個條件來允許或拒絕存取請求。

閃切遷移

一種資料庫遷移方法，透過變更資料擷取使用連續資料複寫，以盡可能在最短的時間內遷移資料，而不是使用分階段方法。目標是將停機時間降至最低。

FM

請參閱基礎模型。

基礎模型 (FM)

大型深度學習神經網路，已在廣義和未標記資料的大量資料集上進行訓練。FMs 能夠執行各種一般任務，例如了解語言、產生文字和影像，以及以自然語言交談。如需詳細資訊，請參閱什麼是基礎模型。

G

生成式 AI

已針對大量資料進行訓練的 AI 模型子集，可使用簡單的文字提示建立新的內容和成品，例如影像、影片、文字和音訊。如需詳細資訊，請參閱什麼是生成式 AI。

地理封鎖

請參閱地理限制。

地理限制 (地理封鎖)

Amazon CloudFront 中的選項，可防止特定國家/地區的使用者存取內容分發。您可以使用允許清單或封鎖清單來指定核准和禁止的國家/地區。如需詳細資訊，請參閱 CloudFront 文件中的限制內容的地理分佈。

Gitflow 工作流程

這是一種方法，其中較低和較高環境在原始碼儲存庫中使用不同分支。Gitflow 工作流程會被視為舊版，而以幹線為基礎的工作流程是現代、偏好的方法。

黃金影像

系統或軟體的快照，做為部署該系統或軟體新執行個體的範本。例如，在製造中，黃金映像可用於在多個裝置上佈建軟體，並有助於提高裝置製造操作的速度、可擴展性和生產力。

綠地策略

新環境中缺乏現有基礎設施。對系統架構採用綠地策略時，可以選擇所有新技術，而不會限制與現有基礎設施的相容性，也稱為棕地。如果正在擴展現有基礎設施，則可能會混合棕地和綠地策略。

防護機制

有助於跨組織單位 (OU) 來管控資源、政策和合規的高層級規則。預防性防護機制會強制執行政策，以確保符合合規標準。透過使用服務控制政策和 IAM 許可界限來將其實作。偵測性防護機制可偵測政策違規和合規問題，並產生提醒以便修正。它們是透過使用 AWS Config AWS Security Hub、Amazon GuardDuty、、Amazon Inspector AWS Trusted Advisor和自訂 AWS Lambda 檢查來實作。

H

HA

請參閱高可用性。

異質資料庫遷移

將來源資料庫遷移至使用不同資料庫引擎的目標資料庫 (例如，Oracle 至 Amazon Aurora)。異質遷移通常是重新架構工作的一部分，而轉換結構描述可能是一項複雜任務。AWS 提供有助於結構描述轉換的 AWS SCT。

高可用性 (HA)

在遇到挑戰或災難時，工作負載能夠在不介入的情況下持續運作。HA 系統的設計目的是自動容錯移轉、持續提供高品質的效能，以及處理不同的負載和故障，並將效能影響降至最低。

歷史現代化

一種方法，用於現代化和升級操作技術 (OT) 系統，以更好地滿足製造業的需求。歷史資料是一種資料庫，用於從工廠中的各種來源收集和存放資料。

保留資料

從用於訓練機器學習模型的資料集中保留的部分歷史標記資料。您可以使用保留資料，透過比較模型預測與保留資料來評估模型效能。

異質資料庫遷移

將您的來源資料庫遷移至共用相同資料庫引擎的目標資料庫 (例如，Microsoft SQL Server 至 Amazon RDS for SQL Server)。同質遷移通常是主機轉換或平台轉換工作的一部分。您可以使用原生資料庫公用程式來遷移結構描述。

熱資料

經常存取的資料，例如即時資料或最近的轉譯資料。此資料通常需要高效能儲存層或類別，才能提供快速的查詢回應。

修補程序

緊急修正生產環境中的關鍵問題。由於其緊迫性，通常會在典型 DevOps 發行工作流程之外執行修補程式。

超級護理期間

在切換後，遷移團隊在雲端管理和監控遷移的應用程式以解決任何問題的時段。通常，此期間的長度為 1-4 天。在超級護理期間結束時，遷移團隊通常會將應用程式的責任轉移給雲端營運團隊。

I

IaC

將基礎設施視為程式碼。

身分型政策

連接至一或多個 IAM 主體的政策，可定義其在 AWS 雲端 環境中的許可。

閒置應用程式

90 天期間 CPU 和記憶體平均使用率在 5% 至 20% 之間的應用程式。在遷移專案中，通常會淘汰這些應用程式或將其保留在內部部署。

IIoT

請參閱工業物聯網。

不可變的基礎設施

為生產工作負載部署新基礎設施的模型，而不是更新、修補或修改現有的基礎設施。不可變基礎設施本質上比可變基礎設施更一致、可靠且可預測。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的使用不可變基礎設施的部署最佳實務。

傳入 (輸入) VPC

在 AWS 多帳戶架構中，接受、檢查和路由來自應用程式外部之網路連線的 VPC。AWS 安全參考架構建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

增量遷移

一種切換策略，您可以在其中將應用程式分成小部分遷移，而不是執行單一、完整的切換。例如，您最初可能只將一些微服務或使用者移至新系統。確認所有項目都正常運作之後，您可以逐步移動其他微服務或使用者，直到可以解除委任舊式系統。此策略可降低與大型遷移關聯的風險。

工業 4.0

2016 年 Klaus Schwab 推出的術語，透過連線能力、即時資料、自動化、分析和 AI/ML 的進展，指製造程序的現代化。

基礎設施

應用程式環境中包含的所有資源和資產。

基礎設施即程式碼 (IaC)

透過一組組態檔案來佈建和管理應用程式基礎設施的程序。IaC 旨在協助您集中管理基礎設施，標準化資源並快速擴展，以便新環境可重複、可靠且一致。

工業物聯網 (IIoT)

在製造業、能源、汽車、醫療保健、生命科學和農業等產業領域使用網際網路連線的感測器和裝置。如需詳細資訊，請參閱建立工業物聯網 (IIoT) 數位轉型策略。

檢查 VPC

在 AWS 多帳戶架構中，集中式 VPC，可管理 VPCs 之間 （在相同或不同的 中 AWS 區域)、網際網路和內部部署網路之間的網路流量檢查。AWS 安全參考架構建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

物聯網 (IoT)

具有內嵌式感測器或處理器的相連實體物體網路，其透過網際網路或本地通訊網路與其他裝置和系統進行通訊。如需詳細資訊，請參閱什麼是 IoT？

可解釋性

機器學習模型的一個特徵，描述了人類能夠理解模型的預測如何依賴於其輸入的程度。如需詳細資訊，請參閱 的機器學習模型可解譯性 AWS。

IoT

請參閱物聯網。

IT 資訊庫 (ITIL)

一組用於交付 IT 服務並使這些服務與業務需求保持一致的最佳實務。ITIL 為 ITSM 提供了基礎。

IT 服務管理 (ITSM)

與組織的設計、實作、管理和支援 IT 服務關聯的活動。如需有關將雲端操作與 ITSM 工具整合的資訊，請參閱操作整合指南。

ITIL

請參閱 IT 資訊庫。

ITSM

請參閱 IT 服務管理。

L

標籤型存取控制 (LBAC)

強制存取控制 (MAC) 的實作，其中使用者和資料本身都會獲得明確指派的安全標籤值。使用者安全標籤和資料安全標籤之間的交集會決定使用者可以看到哪些資料列和資料欄。

登陸區域

登陸區域是架構良好的多帳戶 AWS 環境，可擴展且安全。這是一個起點，您的組織可以從此起點快速啟動和部署工作負載與應用程式，並對其安全和基礎設施環境充滿信心。如需有關登陸區域的詳細資訊，請參閱設定安全且可擴展的多帳戶 AWS 環境。

大型語言模型 (LLM)

預先訓練大量資料的深度學習 AI 模型。LLM 可以執行多個任務，例如回答問題、摘要文件、將文字翻譯成其他語言，以及完成句子。如需詳細資訊，請參閱什麼是 LLMs。

大型遷移

遷移 300 部或更多伺服器。

LBAC

請參閱標籤型存取控制。

最低權限

授予執行任務所需之最低許可的安全最佳實務。如需詳細資訊，請參閱 IAM 文件中的套用最低權限許可。

隨即轉移

請參閱 7 個 R。

小端序系統

首先儲存最低有效位元組的系統。另請參閱 Endianness。

LLM

請參閱大型語言模型。

較低的環境

請參閱 環境。

M

機器學習 (ML)

一種使用演算法和技術進行模式識別和學習的人工智慧。機器學習會進行分析並從記錄的資料 (例如物聯網 (IoT) 資料) 中學習，以根據模式產生統計模型。如需詳細資訊，請參閱機器學習。

主要分支

請參閱分支。

惡意軟體

旨在危及電腦安全或隱私權的軟體。惡意軟體可能會中斷電腦系統、洩露敏感資訊，或取得未經授權的存取。惡意軟體的範例包括病毒、蠕蟲、勒索軟體、特洛伊木馬程式、間諜軟體和鍵盤記錄器。

受管服務

AWS 服務 會 AWS 操作基礎設施層、作業系統和平台，而您會存取端點來存放和擷取資料。Amazon Simple Storage Service (Amazon S3) 和 Amazon DynamoDB 是受管服務的範例。這些也稱為抽象服務。

製造執行系統 (MES)

一種軟體系統，用於追蹤、監控、記錄和控制生產程序，將原物料轉換為現場成品。

MAP

請參閱遷移加速計劃。

機制

建立工具、推動工具採用，然後檢查結果以進行調整的完整程序。機制是在操作時強化和改善自身的循環。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的建置機制。

成員帳戶

除了屬於組織一部分的管理帳戶 AWS 帳戶 之外的所有 AWS Organizations。一個帳戶一次只能是一個組織的成員。

製造執行系統

請參閱製造執行系統。

訊息佇列遙測傳輸 (MQTT)

根據發佈/訂閱模式的輕量型machine-to-machine(M2M) 通訊協定，適用於資源受限的 IoT 裝置。

微服務

一種小型的獨立服務，它可透過定義明確的 API 進行通訊，通常由小型獨立團隊擁有。例如，保險系統可能包含對應至業務能力 (例如銷售或行銷) 或子領域 (例如購買、索賠或分析) 的微服務。微服務的優點包括靈活性、彈性擴展、輕鬆部署、可重複使用的程式碼和適應力。如需詳細資訊，請參閱使用無 AWS 伺服器服務整合微服務。

微服務架構

一種使用獨立元件來建置應用程式的方法，這些元件會以微服務形式執行每個應用程式程序。這些微服務會使用輕量型 API，透過明確定義的介面進行通訊。此架構中的每個微服務都可以進行更新、部署和擴展，以滿足應用程式特定功能的需求。如需詳細資訊，請參閱在 上實作微服務 AWS。

Migration Acceleration Program (MAP)

此 AWS 計畫提供諮詢支援、訓練和服務，以協助組織建立強大的營運基礎，以移至雲端，並協助抵銷遷移的初始成本。MAP 包括用於有條不紊地執行舊式遷移的遷移方法以及一組用於自動化和加速常見遷移案例的工具。

大規模遷移

將大部分應用程式組合依波次移至雲端的程序，在每個波次中，都會以更快的速度移動更多應用程式。此階段使用從早期階段學到的最佳實務和經驗教訓來實作團隊、工具和流程的遷移工廠，以透過自動化和敏捷交付簡化工作負載的遷移。這是 AWS 遷移策略的第三階段。

遷移工廠

可透過自動化、敏捷的方法簡化工作負載遷移的跨職能團隊。遷移工廠團隊通常包括營運、業務分析師和擁有者、遷移工程師、開發人員以及從事 Sprint 工作的 DevOps 專業人員。20% 至 50% 之間的企業應用程式組合包含可透過工廠方法優化的重複模式。如需詳細資訊，請參閱此內容集中的遷移工廠的討論和雲端遷移工廠指南。

遷移中繼資料

有關完成遷移所需的應用程式和伺服器的資訊。每種遷移模式都需要一組不同的遷移中繼資料。遷移中繼資料的範例包括目標子網路、安全群組和 AWS 帳戶。

遷移模式

可重複的遷移任務，詳細描述遷移策略、遷移目的地以及所使用的遷移應用程式或服務。範例：使用 AWS Application Migration Service 重新託管遷移至 Amazon EC2。

遷移組合評定 (MPA)

線上工具，提供驗證商業案例以遷移至 的資訊 AWS 雲端。MPA 提供詳細的組合評定 (伺服器適當規模、定價、總體擁有成本比較、遷移成本分析) 以及遷移規劃 (應用程式資料分析和資料收集、應用程式分組、遷移優先順序，以及波次規劃)。MPA 工具 （需要登入） 可供所有 AWS 顧問和 APN 合作夥伴顧問免費使用。

遷移準備程度評定 (MRA)

使用 AWS CAF 取得組織雲端整備狀態的洞見、識別優缺點，以及建立行動計劃以消除已識別差距的程序。如需詳細資訊，請參閱遷移準備程度指南。MRA 是 AWS 遷移策略的第一階段。

遷移策略

用來將工作負載遷移至 的方法 AWS 雲端。如需詳細資訊，請參閱本詞彙表中的 7 個 Rs 項目，並請參閱動員您的組織以加速大規模遷移。

機器學習 (ML)

請參閱機器學習。

現代化

將過時的 (舊版或單一) 應用程式及其基礎架構轉換為雲端中靈活、富有彈性且高度可用的系統，以降低成本、提高效率並充分利用創新。如需詳細資訊，請參閱《》中的現代化應用程式的策略 AWS 雲端。

現代化準備程度評定

這項評估可協助判斷組織應用程式的現代化準備程度；識別優點、風險和相依性；並確定組織能夠在多大程度上支援這些應用程式的未來狀態。評定的結果就是目標架構的藍圖、詳細說明現代化程序的開發階段和里程碑的路線圖、以及解決已發現的差距之行動計畫。如需詳細資訊，請參閱《》中的評估應用程式的現代化準備 AWS 雲端程度。

單一應用程式 (單一)

透過緊密結合的程序作為單一服務執行的應用程式。單一應用程式有幾個缺點。如果一個應用程式功能遇到需求激增，則必須擴展整個架構。當程式碼庫增長時，新增或改進單一應用程式的功能也會變得更加複雜。若要解決這些問題，可以使用微服務架構。如需詳細資訊，請參閱將單一體系分解為微服務。

MPA

請參閱遷移產品組合評估。

MQTT

請參閱訊息佇列遙測傳輸。

多類別分類

一個有助於產生多類別預測的過程 (預測兩個以上的結果之一)。例如，機器學習模型可能會詢問「此產品是書籍、汽車還是電話？」 或者「這個客戶對哪種產品類別最感興趣？」

可變基礎設施

更新和修改生產工作負載現有基礎設施的模型。為了提高一致性、可靠性和可預測性， AWS Well-Architected Framework 建議使用不可變的基礎設施作為最佳實務。

O

OAC

請參閱原始存取控制。

OAI

請參閱原始存取身分。

OCM

請參閱組織變更管理。

離線遷移

一種遷移方法，可在遷移過程中刪除來源工作負載。此方法涉及延長停機時間，通常用於小型非關鍵工作負載。

OI

請參閱 操作整合。

OLA

請參閱操作層級協議。

線上遷移

一種遷移方法，無需離線即可將來源工作負載複製到目標系統。連接至工作負載的應用程式可在遷移期間繼續運作。此方法涉及零至最短停機時間，通常用於關鍵的生產工作負載。

OPC-UA

請參閱開啟程序通訊 - 統一架構。

開放程序通訊 - 統一架構 (OPC-UA)

用於工業自動化的machine-to-machine(M2M) 通訊協定。OPC-UA 提供資料加密、身分驗證和授權機制的互通性標準。

操作水準協議 (OLA)

一份協議，闡明 IT 職能群組承諾向彼此提供的內容，以支援服務水準協議 (SLA)。

操作整備審查 (ORR)

問題和相關最佳實務的檢查清單，可協助您了解、評估、預防或減少事件和可能失敗的範圍。如需詳細資訊，請參閱 AWS Well-Architected Framework 中的操作準備審查 (ORR)。

操作技術 (OT)

使用實體環境控制工業操作、設備和基礎設施的硬體和軟體系統。在製造業中，整合 OT 和資訊技術 (IT) 系統是工業 4.0 轉型的關鍵重點。

操作整合 (OI)

在雲端中將操作現代化的程序，其中包括準備程度規劃、自動化和整合。如需詳細資訊，請參閱操作整合指南。

組織追蹤

建立的線索 AWS CloudTrail 會記錄 AWS 帳戶 組織中所有 的所有事件 AWS Organizations。在屬於組織的每個 AWS 帳戶 中建立此追蹤，它會跟蹤每個帳戶中的活動。如需詳細資訊，請參閱 CloudTrail 文件中的建立組織追蹤。

組織變更管理 (OCM)

用於從人員、文化和領導力層面管理重大、顛覆性業務轉型的架構。OCM 透過加速變更採用、解決過渡問題，以及推動文化和組織變更，協助組織為新系統和策略做好準備，並轉移至新系統和策略。在 AWS 遷移策略中，此架構稱為人員加速，因為雲端採用專案所需的變更速度。如需詳細資訊，請參閱 OCM 指南。

原始存取控制 (OAC)

CloudFront 中的增強型選項，用於限制存取以保護 Amazon Simple Storage Service (Amazon S3) 內容。OAC 支援所有 S3 儲存貯體中的所有伺服器端加密 AWS KMS (SSE-KMS) AWS 區域，以及對 S3 儲存貯體的動態PUT和DELETE請求。

原始存取身分 (OAI)

CloudFront 中的一個選項，用於限制存取以保護 Amazon S3 內容。當您使用 OAI 時，CloudFront 會建立一個可供 Amazon S3 進行驗證的主體。經驗證的主體只能透過特定 CloudFront 分發來存取 S3 儲存貯體中的內容。另請參閱 OAC，它可提供更精細且增強的存取控制。

ORR

請參閱操作整備審核。

OT

請參閱操作技術。

傳出 (輸出) VPC

在 AWS 多帳戶架構中，處理從應用程式內啟動之網路連線的 VPC。AWS 安全參考架構建議您使用傳入、傳出和檢查 VPC 來設定網路帳戶，以保護應用程式與更廣泛的網際網路之間的雙向介面。

P

許可界限

附接至 IAM 主體的 IAM 管理政策，可設定使用者或角色擁有的最大許可。如需詳細資訊，請參閱 IAM 文件中的許可界限。

個人身分識別資訊 (PII)

當直接檢視或與其他相關資料配對時，可用來合理推斷個人身分的資訊。PII 的範例包括名稱、地址和聯絡資訊。

PII

請參閱個人身分識別資訊。

手冊

一組預先定義的步驟，可擷取與遷移關聯的工作，例如在雲端中提供核心操作功能。手冊可以採用指令碼、自動化執行手冊或操作現代化環境所需的程序或步驟摘要的形式。

PLC

請參閱可程式設計邏輯控制器。

PLM

請參閱產品生命週期管理。

政策

可定義許可的物件（請參閱身分型政策）、指定存取條件（請參閱資源型政策），或定義組織中所有帳戶的最大許可 AWS Organizations（請參閱服務控制政策）。

混合持久性

根據資料存取模式和其他需求，獨立選擇微服務的資料儲存技術。如果您的微服務具有相同的資料儲存技術，則其可能會遇到實作挑戰或效能不佳。如果微服務使用最適合其需求的資料儲存，則可以更輕鬆地實作並達到更好的效能和可擴展性。如需詳細資訊，請參閱在微服務中啟用資料持久性。

組合評定

探索、分析應用程式組合並排定其優先順序以規劃遷移的程序。如需詳細資訊，請參閱評估遷移準備程度。

述詞

傳回 `true`或 的查詢條件`false`，通常位於 `WHERE`子句中。

述詞下推

一種資料庫查詢最佳化技術，可在傳輸前篩選查詢中的資料。這可減少必須從關聯式資料庫擷取和處理的資料量，並改善查詢效能。

預防性控制

旨在防止事件發生的安全控制。這些控制是第一道防線，可協助防止對網路的未經授權存取或不必要變更。如需詳細資訊，請參閱在 AWS上實作安全控制中的預防性控制。

委託人

中可執行動作和存取資源 AWS 的實體。此實體通常是 AWS 帳戶、IAM 角色或使用者的根使用者。如需詳細資訊，請參閱 IAM 文件中角色術語和概念中的主體。

設計隱私權

透過整個開發程序將隱私權納入考量的系統工程方法。

私有託管區域

一種容器，它包含有關您希望 Amazon Route 53 如何回應一個或多個 VPC 內的域及其子域之 DNS 查詢的資訊。如需詳細資訊，請參閱 Route 53 文件中的使用私有託管區域。

主動控制

旨在防止部署不合規資源的安全控制。這些控制項會在佈建資源之前對其進行掃描。如果資源不符合控制項，則不會佈建。如需詳細資訊，請參閱 AWS Control Tower 文件中的控制項參考指南，並參閱實作安全控制項中的主動控制項。 AWS

產品生命週期管理 (PLM)

管理產品整個生命週期的資料和程序，從設計、開發和啟動，到成長和成熟，再到拒絕和移除。

生產環境

請參閱 環境。

可程式設計邏輯控制器 (PLC)

在製造中，高度可靠、可調整的電腦，可監控機器並自動化製造程序。

提示鏈結

使用一個 LLM 提示的輸出做為下一個提示的輸入，以產生更好的回應。此技術用於將複雜任務分解為子任務，或反覆精簡或展開初步回應。它有助於提高模型回應的準確性和相關性，並允許更精細、個人化的結果。

擬匿名化

將資料集中的個人識別符取代為預留位置值的程序。假名化有助於保護個人隱私權。假名化資料仍被視為個人資料。

發佈/訂閱 (pub/sub)

一種模式，可啟用微服務之間的非同步通訊，以提高可擴展性和回應能力。例如，在微服務型 MES 中，微服務可以將事件訊息發佈到其他微服務可訂閱的頻道。系統可以新增新的微服務，而無需變更發佈服務。

Q

查詢計劃

一系列步驟，如指示，用於存取 SQL 關聯式資料庫系統中的資料。

查詢計劃迴歸

在資料庫服務優化工具選擇的計畫比對資料庫環境進行指定的變更之前的計畫不太理想時。這可能因為對統計資料、限制條件、環境設定、查詢參數繫結的變更以及資料庫引擎的更新所導致。

R

RACI 矩陣

請參閱負責、負責、諮詢、告知 (RACI)。

RAG

請參閱擷取增強產生。

勒索軟體

一種惡意軟體，旨在阻止對計算機系統或資料的存取，直到付款為止。

RASCI 矩陣

請參閱負責、負責、諮詢、告知 (RACI)。

RCAC

請參閱資料列和資料欄存取控制。

僅供讀取複本

用於唯讀用途的資料庫複本。您可以將查詢路由至僅供讀取複本以減少主資料庫的負載。

重新架構師

請參閱 7 個 R。

復原點目標 (RPO)

自上次資料復原點以來可接受的時間上限。這會決定最後一個復原點與服務中斷之間可接受的資料遺失。

復原時間目標 (RTO)

服務中斷與服務還原之間的可接受延遲上限。

重構

請參閱 7 個 R。

區域

地理區域中的 AWS 資源集合。每個 AWS 區域 都獨立於其他 ，以提供容錯能力、穩定性和彈性。如需詳細資訊，請參閱指定 AWS 區域 您的帳戶可以使用哪些 。

迴歸

預測數值的 ML 技術。例如，為了解決「這房子會賣什麼價格？」的問題 ML 模型可以使用線性迴歸模型，根據已知的房屋事實 (例如，平方英尺) 來預測房屋的銷售價格。

重新託管

請參閱 7 個 R。

版本

在部署程序中，它是將變更提升至生產環境的動作。

重新放置

請參閱 7 個 R。

Replatform

請參閱 7 個 R。

回購

請參閱 7 個 R。

彈性

應用程式抵禦中斷或從中斷中復原的能力。在 中規劃彈性時，高可用性和災難復原是常見的考量 AWS 雲端。如需詳細資訊，請參閱AWS 雲端 彈性。

資源型政策

附接至資源的政策，例如 Amazon S3 儲存貯體、端點或加密金鑰。這種類型的政策會指定允許存取哪些主體、支援的動作以及必須滿足的任何其他條件。

負責者、當責者、事先諮詢者和事後告知者 (RACI) 矩陣

定義所有涉及遷移活動和雲端操作之各方的角色和責任的矩陣。矩陣名稱衍生自矩陣中定義的責任類型：負責人 (R)、責任 (A)、已諮詢 (C) 和知情 (I)。支援 (S) 類型為選用。如果您包含支援，則矩陣稱為 RASCI 矩陣，如果您排除它，則稱為 RACI 矩陣。

回應性控制

一種安全控制，旨在驅動不良事件或偏離安全基準的補救措施。如需詳細資訊，請參閱在 AWS上實作安全控制中的回應性控制。

保留

請參閱 7 個 R。

淘汰

請參閱 7 Rs。

檢索增強生成 (RAG)

一種生成式 AI 技術，其中 LLM 會在產生回應之前參考訓練資料來源以外的授權資料來源。例如，RAG 模型可能會對組織的知識庫或自訂資料執行語意搜尋。如需詳細資訊，請參閱什麼是 RAG。

輪換

定期更新秘密的程序，讓攻擊者更難存取登入資料。

資料列和資料欄存取控制 (RCAC)

使用已定義存取規則的基本、彈性 SQL 表達式。RCAC 包含資料列許可和資料欄遮罩。

RPO

請參閱復原點目標。

RTO

請參閱復原時間目標。

執行手冊

執行特定任務所需的一組手動或自動程序。這些通常是為了簡化重複性操作或錯誤率較高的程序而建置。

S

SAML 2.0

許多身分提供者 (IdP) 使用的開放標準。此功能可啟用聯合單一登入 (SSO)，讓使用者可以登入 AWS Management Console 或呼叫 AWS API 操作，而無需為您組織中的每個人在 IAM 中建立使用者。如需有關以 SAML 2.0 為基礎的聯合詳細資訊，請參閱 IAM 文件中的關於以 SAML 2.0 為基礎的聯合。

SCADA

請參閱監督控制和資料擷取。

SCP

請參閱服務控制政策。

秘密

您以加密形式存放的 AWS Secrets Manager機密或限制資訊，例如密碼或使用者登入資料。它由秘密值及其中繼資料組成。秘密值可以是二進位、單一字串或多個字串。如需詳細資訊，請參閱 Secrets Manager 文件中的 Secrets Manager 秘密中的什麼內容？。

依設計的安全性

透過整個開發程序將安全性納入考量的系統工程方法。

安全控制

一種技術或管理防護機制，它可預防、偵測或降低威脅行為者利用安全漏洞的能力。安全控制有四種主要類型：預防性、偵測性、回應性和主動性。

安全強化

減少受攻擊面以使其更能抵抗攻擊的過程。這可能包括一些動作，例如移除不再需要的資源、實作授予最低權限的安全最佳實務、或停用組態檔案中不必要的功能。

安全資訊與事件管理 (SIEM) 系統

結合安全資訊管理 (SIM) 和安全事件管理 (SEM) 系統的工具與服務。SIEM 系統會收集、監控和分析來自伺服器、網路、裝置和其他來源的資料，以偵測威脅和安全漏洞，並產生提醒。

安全回應自動化

預先定義和程式設計的動作，旨在自動回應或修復安全事件。這些自動化可做為偵測或回應式安全控制，協助您實作 AWS 安全最佳實務。自動化回應動作的範例包括修改 VPC 安全群組、修補 Amazon EC2 執行個體或輪換登入資料。

伺服器端加密

由 AWS 服務 接收資料的 在其目的地加密資料。

服務控制政策 (SCP)

為 AWS Organizations中的組織的所有帳戶提供集中控制許可的政策。SCP 會定義防護機制或設定管理員可委派給使用者或角色的動作限制。您可以使用 SCP 作為允許清單或拒絕清單，以指定允許或禁止哪些服務或動作。如需詳細資訊，請參閱 AWS Organizations 文件中的服務控制政策。

服務端點

的進入點 URL AWS 服務。您可以使用端點，透過程式設計方式連接至目標服務。如需詳細資訊，請參閱 AWS 一般參考 中的 AWS 服務 端點。

服務水準協議 (SLA)

一份協議，闡明 IT 團隊承諾向客戶提供的服務，例如服務正常執行時間和效能。

服務層級指標 (SLI)

服務效能方面的測量，例如其錯誤率、可用性或輸送量。

服務層級目標 (SLO)

代表服務運作狀態的目標指標，由服務層級指標測量。

共同責任模式

描述您與 共同 AWS 承擔雲端安全與合規責任的模型。 AWS 負責雲端的安全，而 負責雲端的安全。如需詳細資訊，請參閱共同責任模式。

SIEM

請參閱安全資訊和事件管理系統。

單一故障點 (SPOF)

應用程式的單一關鍵元件故障，可能會中斷系統。

SLA

請參閱服務層級協議。

SLI

請參閱服務層級指標。

SLO

請參閱服務層級目標。

先拆分後播種模型

擴展和加速現代化專案的模式。定義新功能和產品版本時，核心團隊會進行拆分以建立新的產品團隊。這有助於擴展組織的能力和服務，提高開發人員生產力，並支援快速創新。如需詳細資訊，請參閱 中的階段式應用程式現代化方法 AWS 雲端。

SPOF

請參閱單一故障點。

星狀結構描述

使用一個大型事實資料表來存放交易或測量資料的資料庫組織結構，並使用一或多個較小的維度資料表來存放資料屬性。此結構旨在用於資料倉儲或商業智慧用途。

Strangler Fig 模式

一種現代化單一系統的方法，它會逐步重寫和取代系統功能，直到舊式系統停止使用為止。此模式源自無花果藤，它長成一棵馴化樹並最終戰勝且取代了其宿主。該模式由 Martin Fowler 引入，作為重寫單一系統時管理風險的方式。如需有關如何套用此模式的範例，請參閱使用容器和 Amazon API Gateway 逐步現代化舊版 Microsoft ASP.NET (ASMX) Web 服務。

子網

您 VPC 中的 IP 地址範圍。子網必須位於單一可用區域。

監控控制和資料擷取 (SCADA)

在製造中，使用硬體和軟體來監控實體資產和生產操作的系統。

對稱加密

使用相同金鑰來加密及解密資料的加密演算法。

合成測試

以模擬使用者互動的方式測試系統，以偵測潛在問題或監控效能。您可以使用 Amazon CloudWatch Synthetics 來建立這些測試。

系統提示

一種向 LLM 提供內容、指示或指導方針以指示其行為的技術。系統提示有助於設定內容，並建立與使用者互動的規則。

T

標籤

做為中繼資料以組織 AWS 資源的鍵值對。標籤可協助您管理、識別、組織、搜尋及篩選資源。如需詳細資訊，請參閱標記您的 AWS 資源。

目標變數

您嘗試在受監督的 ML 中預測的值。這也被稱為結果變數。例如，在製造設定中，目標變數可能是產品瑕疵。

任務清單

用於透過執行手冊追蹤進度的工具。任務清單包含執行手冊的概觀以及要完成的一般任務清單。對於每個一般任務，它包括所需的預估時間量、擁有者和進度。

測試環境

請參閱 環境。

訓練

為 ML 模型提供資料以供學習。訓練資料必須包含正確答案。學習演算法會在訓練資料中尋找將輸入資料屬性映射至目標的模式 (您想要預測的答案)。它會輸出擷取這些模式的 ML 模型。可以使用 ML 模型，來預測您不知道的目標新資料。

傳輸閘道

可以用於互連 VPC 和內部部署網路的網路傳輸中樞。如需詳細資訊，請參閱 AWS Transit Gateway 文件中的什麼是傳輸閘道。

主幹型工作流程

這是一種方法，開發人員可在功能分支中本地建置和測試功能，然後將這些變更合併到主要分支中。然後，主要分支會依序建置到開發環境、生產前環境和生產環境中。

受信任的存取權

將許可授予您指定的服務，以代表您在組織中 AWS Organizations 及其帳戶中執行任務。受信任的服務會在需要該角色時，在每個帳戶中建立服務連結角色，以便為您執行管理工作。如需詳細資訊，請參閱 文件中的 AWS Organizations 搭配使用 AWS Organizations 與其他 AWS 服務。

調校

變更訓練程序的各個層面，以提高 ML 模型的準確性。例如，可以透過產生標籤集、新增標籤、然後在不同的設定下多次重複這些步驟來訓練 ML 模型，以優化模型。

雙比薩團隊

兩個比薩就能吃飽的小型 DevOps 團隊。雙披薩團隊規模可確保軟體開發中的最佳協作。

U

不確定性

這是一個概念，指的是不精確、不完整或未知的資訊，其可能會破壞預測性 ML 模型的可靠性。有兩種類型的不確定性：認知不確定性是由有限的、不完整的資料引起的，而隨機不確定性是由資料中固有的噪聲和隨機性引起的。如需詳細資訊，請參閱量化深度學習系統的不確定性指南。

未區分的任務

也稱為繁重工作，這是建立和操作應用程式的必要工作，但不為最終使用者提供直接價值或提供競爭優勢。未區分任務的範例包括採購、維護和容量規劃。

較高的環境

請參閱 環境。

V

清空

一種資料庫維護操作，涉及增量更新後的清理工作，以回收儲存並提升效能。

版本控制

追蹤變更的程序和工具，例如儲存庫中原始程式碼的變更。

VPC 對等互連

兩個 VPC 之間的連線，可讓您使用私有 IP 地址路由流量。如需詳細資訊，請參閱 Amazon VPC 文件中的什麼是 VPC 對等互連。

漏洞

危害系統安全性的軟體或硬體瑕疵。

W

暖快取

包含經常存取的目前相關資料的緩衝快取。資料庫執行個體可以從緩衝快取讀取，這比從主記憶體或磁碟讀取更快。

暖資料

不常存取的資料。查詢這類資料時，通常可接受中等緩慢的查詢。

視窗函數

SQL 函數，對與目前記錄在某種程度上相關的資料列群組執行計算。視窗函數適用於處理任務，例如根據目前資料列的相對位置計算移動平均值或存取資料列的值。

工作負載

提供商業價值的資源和程式碼集合，例如面向客戶的應用程式或後端流程。

工作串流

遷移專案中負責一組特定任務的功能群組。每個工作串流都是獨立的，但支援專案中的其他工作串流。例如，組合工作串流負責排定應用程式、波次規劃和收集遷移中繼資料的優先順序。組合工作串流將這些資產交付至遷移工作串流，然後再遷移伺服器和應用程式。

WORM

請參閱寫入一次，多次讀取。

WQF

請參閱AWS 工作負載資格架構。

寫入一次，讀取許多 (WORM)

儲存模型，可一次性寫入資料，並防止刪除或修改資料。授權使用者可以視需要多次讀取資料，但無法變更資料。此資料儲存基礎設施被視為不可變。

Z

零時差入侵

利用零時差漏洞的攻擊，通常是惡意軟體。

零時差漏洞

生產系統中未緩解的瑕疵或漏洞。威脅行為者可以使用這種類型的漏洞來攻擊系統。開發人員經常因為攻擊而意識到漏洞。

零鏡頭提示

提供 LLM 執行任務的指示，但沒有可協助引導任務的範例 (快照)。LLM 必須使用其預先訓練的知識來處理任務。零鏡頭提示的有效性取決於任務的複雜性和提示的品質。另請參閱少量擷取提示。

殭屍應用程式

CPU 和記憶體平均使用率低於 5% 的應用程式。在遷移專案中，通常會淘汰這些應用程式。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。