使用 Terraform AWS 提供者的最佳實務

AWS 方案指引

AWS 方案指引: 使用 Terraform AWS 提供者的最佳實務

Table of Contents

使用地形 AWS 供應商的最佳實務

邁克爾開始, 高級 DevOps 顧問, Amazon Web Services (AWS)

2024 年 5 月 (文件歷史記錄)

使用 Terraform 管理基礎架構即程式碼 (IaC) 可 AWS 提供重要優勢，例如改善一致性、安全性和敏捷性。但是，隨著 Terraform 配置的大小和複雜性不斷增加，遵循最佳實踐以避免陷阱變得至關重要。

本指南提供使用來自的 Terraform 提供 AWS 者的建議最佳作法。 HashiCorp它會引導您完成適當的版本控制，安全控制，遠程後端，代碼庫結構和社區提供商，以優化 Terraform。 AWS每個部分都會深入探討如何套用這些最佳做法的詳細資訊：

- 安全性
- 後端
- 代碼庫結構和組織
- AWS 提供者版本管理
- 社群模組

目標

本指南可幫助您獲得有關 Terraform AWS Provider 的操作知識，並通過遵循有關安全性，可靠性，合規性和開發人員生產力的 IaC 最佳實踐來解決以下業務目標。

- 改善 Terraform 專案之間的基礎架構程式碼品質和一致性。
- 加速開發人員上線，並為基礎架構程式碼做出貢獻。
- 透過更快速的基礎架構變更，提高業務
- 減少與基礎架構變更相關的錯誤和停機時間
- 遵循 IaC 最佳做法，將基礎架構成本最佳化。
- 透過實作最佳實務，強化您的整體安全狀態。

目標受眾

本指南的目標受眾包括技術線索和管理人員，他們負責監督使用 Terraform 的 IaC 上的團隊。 AWS其他潛在讀者包括基礎設施工程師、 DevOps 工程師、解決方案架構師和積極使用 Terraform 管理 AWS 基礎設施的開發人員。

遵循這些最佳實踐將節省時間，並幫助解鎖 IaC 對這些角色的好處。

概觀

地形提供程序是允許地形與不同的 API 進行交互的插件。Terraform AWS 提供者是用於管理 AWS 基礎設施作為代碼（IaC）與 Terraform 的官方插件。它將 Terraform 語法轉換為 AWS API 調用以創建，讀取，更新和刪除資源。 AWS

AWS 提供者處理驗證、將 Terraform 語法轉換為 AWS API 呼叫，以及在中佈建資源。 AWS您可以使用 Terraform 程式`provider`碼區塊來設定 Terraform 用來與 API 互動的提供者外掛程式。 AWS 您可以設定多個 AWS Provider 區塊，以管理跨不同區域 AWS 帳戶 和區域的資源。

以下是一個範例 Terraform 組態，它使用多個 AWS 提供者區塊和別名來管理在不同區域和帳戶中具有複本的 Amazon 關聯式資料庫服務 (Amazon RDS) 資料庫。主要和次要提供者採用不同的 AWS Identity and Access Management (IAM) 角色：

```
# Configure the primary AWS Provider
provider "aws" {
  region = "us-west-1"
  alias  = "primary"
}

# Configure a secondary AWS Provider for the replica Region and account
provider "aws" {
  region      = "us-east-1"
  alias       = "replica"
  assume_role {
    role_arn     = "arn:aws:iam::<replica-account-id>:role/<role-name>"
    session_name = "terraform-session"
  }
}

# Primary Amazon RDS database
resource "aws_db_instance" "primary" {
  provider = aws.primary

  # ... RDS instance configuration
}

# Read replica in a different Region and account
resource "aws_db_instance" "read_replica" {
  provider = aws.replica
```

```
  # ... RDS read replica configuration
  replicate_source_db = aws_db_instance.primary.id
}
```

在此範例中：

- 第一個區provider塊會使用別名設定us-west-1區域中的主要 AWS 提供者。primary
- 第二個區provider塊會使用別名設定us-east-1區域中的次要 AWS 提供者。replica此提供者用於在不同的區域和帳戶中建立主要資料庫的僅供讀取複本。該assume_role區塊用於在複本帳戶中扮演 IAM 角色。role_arn指定要承擔的 IAM 角色的 Amazon 資源名稱 (ARN)，並且session_name是 Terraform 工作階段的唯一識別碼。
- 資aws_db_instance.primary源會使用us-west-1區域中的primary提供者建立主要的 Amazon RDS 資料庫。
- 資aws_db_instance.read_replica源會使用提供者在us-east-1區域中建立主要資料庫的僅replica供讀取複本。replicate_source_db屬性會參照primary資料庫的 ID。

安全最佳實務

正確管理身分驗證、存取控制和安全性對於 Terraform AWS 提供者的安全使用至關重要。本節概述以下方面的最佳實務：

- 最低權限存取的 IAM 角色和許可
- 保護登入資料，以協助防止未經授權存取 AWS 帳戶和資源
- 遠端狀態加密以協助保護敏感資料
- 基礎設施和原始程式碼掃描以識別組態錯誤
- 遠端狀態儲存的存取控制
- Sentinel 政策強制執行以實作控管護欄

當您使用 Terraform 管理 AWS 基礎設施時，遵循這些最佳實務有助於強化您的安全狀態。

遵循最低權限原則

最低權限是基本安全原則，是指僅授予使用者、程序或系統執行其預期功能所需的最低許可。這是存取控制的核心概念，也是防止未經授權的存取和潛在資料外洩的預防措施。

本節多次強調最低權限原則，因為它與 Terraform 如何對雲端提供者進行身分驗證和執行動作直接相關，例如 AWS。

當您使用 Terraform 佈建和管理 AWS 資源時，它會代表需要適當許可才能進行 API 呼叫的實體（使用者或角色）。未遵循最低權限會開啟主要安全風險：

- 如果 Terraform 具有超出所需範圍的過多許可，意外的錯誤設定可能會進行不必要的變更或刪除。
- 如果 Terraform 狀態檔案或登入資料遭到入侵，過度寬鬆的存取授權會增加影響範圍。
- 未遵循最低權限會違反安全最佳實務和法規合規要求，以授予最低必要存取權。

使用 IAM 角色

盡可能使用 IAM 角色而非 IAM 使用者來增強 Terraform AWS 提供者的安全性。IAM 角色提供可自動輪換的臨時安全登入資料，無需管理長期存取金鑰。角色也透過 IAM 政策提供精確的存取控制。

使用 IAM 政策授予最低權限存取

仔細建構 IAM 政策，以確保角色和使用者只有其工作負載所需的最低許可集。從空白政策開始，並反覆新增允許的服務和動作。若要達成此目的：

- 啟用 IAM Access Analyzer 以評估政策，並反白顯示可移除的未使用許可。
- 手動檢閱政策，以移除對角色的預期責任不重要的任何功能。
- 使用 IAM 政策變數和標籤來簡化許可管理。

建構良好的政策只授予足夠的存取權，以履行工作負載的責任，而且沒有其他任何功能。在操作層級定義動作，並僅允許呼叫特定資源所需的 APIs。

遵循此最佳實務可減少影響範圍，並遵循職責分離和最低權限存取的基本安全原則。視需要逐步開始嚴格且開放的存取，而不是開始開放並稍後嘗試限制存取。

擔任本機身分驗證的 IAM 角色

當您在本機執行 Terraform 時，請避免設定靜態存取金鑰。反之，請使用 IAM 角色暫時授予特殊存取權限，而不會暴露長期憑證。

首先，建立具有必要最低許可的 IAM 角色，並新增信任關係，允許您的使用者帳戶或聯合身分擔任 IAM 角色。這會授權暫時使用 角色。

信任關係政策範例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/terraform-execution"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

然後，執行 AWS CLI 命令 aws sts assume-role 來擷取角色的短期登入資料。這些登入資料通常有效一小時。

AWS CLI 命令範例：

```
aws sts assume-role --role-arn arn:aws:iam::111122223333:role/terraform-execution --
role-session-name terraform-session-example
```

命令的輸出包含存取金鑰、私密金鑰和工作階段字符，可用於驗證 AWS：

```
{
    "AssumedRoleUser": {
        "AssumedRoleId": "AROA3XFRBF535PLBIFPI4:terraform-session-example",
        "Arn": "arn:aws:sts::111122223333:assumed-role/terraform-execution/terraform-
session-example"
    },
    "Credentials": {
        "SecretAccessKey": " wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "SessionToken": " AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
+FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/
IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4OlgkBN9bkUDNCJiBeb/
AXlzBBko7b15fjrBs2+cTQtpZ3CYWFXG8C5zqx37wnOE49mRl/+OtkIKGO7fAE",
        "Expiration": "2024-03-15T00:05:07Z",
        "AccessKeyId": "ASIAIOSFODNN7EXAMPLE"
    }
}
```

AWS 提供者也可以自動處理擔任該角色。

擔任 IAM 角色的提供者組態範例：

```
provider "aws" {
  assume_role {
    role_arn     = "arn:aws:iam::111122223333:role/terraform-execution"
    session_name = "terraform-session-example"
  }
}
```

這會在 Terraform 工作階段的持續時間嚴格授予更高的權限。臨時金鑰無法洩漏，因為它們會在工作階段的最長持續時間之後自動過期。

相較於長期存取金鑰，此最佳實務的主要優點包括改善安全性、以最低權限對角色進行精細存取控制，以及透過修改角色的許可輕鬆撤銷存取權。透過使用 IAM 角色，您也可以避免將秘密直接存放在本機的指令碼或磁碟上，這可協助您在團隊中安全地共用 Terraform 組態。

使用 IAM 角色進行 Amazon EC2 身分驗證

當您從 Amazon Elastic Compute Cloud (Amazon EC2) 執行個體執行 Terraform 時，請避免在本機存放長期憑證。反之，請使用 IAM 角色和執行個體描述檔自動授予最低權限許可。

首先，建立具有最低許可的 IAM 角色，並將該角色指派給執行個體描述檔。執行個體描述檔允許 EC2 執行個體繼承角色中定義的許可。然後，透過指定執行個體描述檔來啟動執行個體。執行個體將透過連接的 角色進行身分驗證。

在您執行任何 Terraform 操作之前，請確認該角色存在於執行個體中繼資料中，以確認登入資料已成功繼承。

```
TOKEN=$(curl -s -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-
metadata-token-ttl-seconds: 21600")

curl -H "X-aws-ec2-metadata-token: $TOKEN" -s http://169.254.169.254/latest/meta-data/
iam/security-credentials/
```

此方法可避免將永久 AWS 金鑰硬式編碼為執行個體內的指令碼或 Terraform 組態。臨時登入資料可透過執行個體角色和設定檔以透明方式提供給 Terraform。

此最佳實務的主要優點包括改善長期憑證的安全性、降低憑證管理開銷，以及開發、測試和生產環境之間的一致性。IAM 角色身分驗證可簡化從 EC2 執行個體執行的 Terraform，同時強制執行最低權限存取。

使用 HCP Terraform 工作區的動態登入資料

HCP Terraform 是由 HashiCorp 提供的受管服務，可協助團隊使用 Terraform 跨多個專案和環境佈建和管理基礎設施。當您在 HCP Terraform 中執行 Terraform 時，請使用動態登入資料來簡化和保護 AWS 身分驗證。Terraform 會在每次執行時自動交換臨時登入資料，而不需要 IAM 角色假設。

優點包括更容易進行秘密輪換、跨工作區的集中式憑證管理、最低權限許可，以及消除硬式編碼金鑰。相較於長期存取金鑰，依賴雜湊暫時性金鑰可增強安全性。

在 中使用 IAM 角色 AWS CodeBuild

在 中 AWS CodeBuild，使用指派給 CodeBuild 專案的 IAM 角色來執行您的組建。這可讓每個組建自動從角色繼承臨時登入資料，而不是使用長期金鑰。

在 HCP Terraform 上遠端執行 GitHub 動作

設定 GitHub 動作工作流程以在 HCP Terraform 工作區上遠端執行 Terraform。依賴動態登入資料和遠端狀態鎖定，而不是 GitHub 秘密管理。

將 GitHub 動作與 OIDC 搭配使用，並設定 AWS 登入資料動作

使用 OpenID Connect (OIDC) 標準透過 IAM 聯合 GitHub Actions 身分。使用設定 AWS 登入資料動作，將 GitHub 字符交換為臨時 AWS 登入資料，而不需要長期存取金鑰。

搭配 OIDC 和 使用 GitLab AWS CLI

使用 OIDC 標準透過 IAM 聯合 GitLab 身分以進行暫時存取。透過依賴 OIDC，您可以避免在 GitLab 中直接管理長期 AWS 存取金鑰。登入資料會just-in-time交換，以改善安全性。使用者也會根據 IAM 角色中的許可取得最低權限存取。

搭配舊版自動化工具使用唯一的 IAM 使用者

如果您有自動化工具和指令碼，缺乏使用 IAM 角色的原生支援，您可以建立個別 IAM 使用者來授予程式設計存取權。最低權限原則仍然適用。將政策許可降至最低，並依賴每個管道或指令碼的個別角色。當您遷移到更現代化的工具或指令碼時，請開始原生支援角色，並逐漸轉換到這些角色。

> **Warning**
>
> IAM 使用者具有長期登入資料，這會造成安全風險。為了協助降低此風險，建議您只為這些使用者提供執行任務所需的許可，並在不再需要這些使用者時將其移除。

使用 Jenkins AWS 登入資料外掛程式

使用 Jenkins 中的AWS 登入資料外掛程式，以動態方式將 AWS 登入資料集中設定和插入組建。這可避免將秘密檢查為來源控制。

持續監控、驗證和最佳化最低權限

隨著時間的推移，可能會授予可能超過所需最低政策的額外許可。持續分析存取權，以識別和移除任何不必要的權利。

持續監控存取金鑰用量

如果您無法避免使用存取金鑰，請使用 IAM 登入資料報告來尋找超過 90 天的未使用存取金鑰，以及撤銷使用者帳戶和機器角色的非作用中金鑰。提醒管理員手動確認移除作用中員工和系統的金鑰。

監控金鑰用量可協助您最佳化許可，因為您可以識別和移除未使用的權利。當您遵循此最佳實務進行存取金鑰輪換時，它會限制登入資料生命週期並強制執行最低權限存取。

AWS 提供多種服務和功能，您可以用來為管理員設定提醒和通知。以下是一些選項：

- AWS Config：您可以使用 AWS Config 規則來評估 AWS 資源的組態設定，包括 IAM 存取金鑰。您可以建立自訂規則來檢查特定條件，例如早於特定天數的未使用存取金鑰。違反規則時， AWS Config 可以開始評估修補或傳送通知到 Amazon Simple Notification Service (Amazon SNS) 主題。
- AWS Security Hub：Security Hub 提供 AWS 帳戶安全狀態的完整檢視，並可協助偵測和通知您潛在的安全問題，包括未使用或非作用中的 IAM 存取金鑰。Security Hub 可以在聊天應用程式中與 Amazon EventBridge 和 Amazon SNS 或 Amazon Q Developer 整合，以傳送通知給管理員。
- AWS Lambda：Lambda 函數可以由各種事件呼叫，包括 Amazon CloudWatch Events 或 AWS Config 規則。您可以撰寫自訂 Lambda 函數來評估 IAM 存取金鑰用量、執行其他檢查，以及在聊天應用程式中使用 Amazon SNS 或 Amazon Q Developer 等服務傳送通知。

持續驗證 IAM 政策

使用 IAM Access Analyzer 評估連接到角色的政策，並識別任何未使用的服務或授予的多餘動作。實作定期存取檢閱，以手動驗證政策是否符合目前的需求。

比較現有政策與 IAM Access Analyzer 產生的政策，並移除任何不必要的許可。您也應該向使用者提供報告，並在寬限期後自動撤銷未使用的許可。這有助於確保最少的政策保持有效。

主動且頻繁地撤銷過時的存取，可將違規期間可能面臨風險的登入資料降至最低。自動化提供永續、長期的憑證衛生和許可最佳化。遵循此最佳實務，透過主動跨 AWS 身分和資源強制執行最低權限來限制影響範圍。

安全的遠端狀態儲存

遠端狀態儲存是指遠端儲存 Terraform 狀態檔案，而不是在執行 Terraform 的機器上本機儲存。狀態檔案至關重要，因為它會追蹤 Terraform 佈建的資源及其中繼資料。

無法保護遠端狀態可能會導致嚴重問題，例如遺失狀態資料、無法管理基礎設施、意外刪除資源，以及暴露狀態檔案中可能存在的敏感資訊。因此，保護遠端狀態儲存對於生產級 Terraform 使用至關重要。

啟用加密和存取控制

使用 Amazon Simple Storage Service (Amazon S3) 伺服器端加密 (SSE) 加密靜態遠端狀態。

限制對協作工作流程的直接存取

- 在 HCP Terraform 或 Git 儲存庫內 CI/CD 管道中建構協作工作流程，以限制直接狀態存取。
- 依賴提取請求、執行核准、政策檢查和通知來協調變更。

遵循這些準則有助於保護敏感資源屬性，並避免與團隊成員的變更發生衝突。加密和嚴格的存取保護有助於減少攻擊面，協作工作流程可實現生產力。

使用 AWS Secrets Manager

Terraform 中有許多資源和資料來源，可將純文字的秘密值存放在 狀態檔案中。避免將秘密存放在 狀態 – 請AWS Secrets Manager改用 。

不嘗試手動加密敏感值，而是依賴 Terraform 內建的敏感狀態管理支援。匯出敏感值至輸出時，請確定這些值標示為敏感值。

持續掃描基礎設施和原始程式碼

主動持續掃描基礎設施和原始程式碼，以找出暴露的登入資料或設定錯誤等風險，以強化您的安全狀態。透過重新設定或修補資源，快速解決問題清單。

使用 AWS 服務進行動態掃描

使用 AWS Amazon Inspector、AWS Security Hub、Amazon Detective 和 Amazon GuardDuty 等原生工具來監控跨帳戶和區域的佈建基礎設施。在 Security Hub 中排程定期掃描，以追蹤部署和組態偏離。掃描 EC2 執行個體、Lambda 函數、容器、S3 儲存貯體和其他資源。

執行靜態分析

將 Checkov 等靜態分析器直接嵌入 CI/CD 管道，以掃描 Terraform 組態碼 (HCL)，並在部署之前先識別風險。這會將安全檢查移至開發程序的較早時間點 （ 稱為向左轉移)，並防止基礎設施設定錯誤。

確保提示修復

對於所有掃描問題清單，請視需要更新 Terraform 組態、套用修補程式或手動重新設定資源，以確保及時修復。解決根本原因以降低風險等級。

同時使用基礎設施掃描和程式碼掃描，可在 Terraform 組態、佈建的資源和應用程式程式碼之間提供分層洞見。這可透過預防性、偵測性和反應性控制來最大化風險與合規的涵蓋範圍，同時更早地將安全性嵌入軟體開發生命週期 (SDLC)。

強制執行政策檢查

使用 HashiCorp Sentinel 政策等程式碼架構來提供控管防護機制和標準化範本，以使用 Terraform 進行基礎設施佈建。

Sentinel 政策可以定義 Terraform 組態的要求或限制，以符合組織標準和最佳實務。例如，您可以使用 Sentinel 政策來：

- 需要所有資源上的標籤。
- 將執行個體類型限制為核准清單。
- 強制執行強制性變數。
- 防止破壞生產資源。

將政策檢查嵌入 Terraform 組態生命週期中，可主動強制執行標準和架構準則。Sentinel 提供共用政策邏輯，有助於加速開發，同時防止未經核准的實務。

後端最佳做法

使用適當的遠端後端儲存狀態檔案對於啟用協同作業、透過鎖定確保狀態檔案完整性、提供可靠的備份和復原、與 CI/CD 工作流程整合，以及利用 HCP Terraform 等受管理服務提供的進階安全性、控管和管理功能至關重要。

地形支持各種後端類型，例如 Kubernetes， HashiCorp 領事和 HTTP。但是，本指南著重於 Amazon S3，這是大多數 AWS 使用者的最佳後端解決方案。

作為提供高耐久性和可用性的全受管物件儲存服務，Amazon S3 提供安全、可擴展且低成本的後端，用於管理 Terraform 狀態。 AWS Amazon S3 的全球足跡和彈性超過了大多數團隊透過自我管理狀態儲存所能達到的目標。此外，Amazon S3 與 AWS 存取控制、加密選項、版本控制功能和其他服務原生整合，讓 Amazon S3 成為方便的後端選擇。

本指南不提供其他解決方案 (例如 Kubernetes 或 Consul) 的後端指引，因為主要目標對象是客戶。AWS 對於完全屬於中的團隊而言 AWS 雲端，Amazon S3 通常是 Kubernetes 或 HashiCorp Consul 叢集的理想選擇。Amazon S3 狀態儲存的簡易性、彈性和緊密 AWS 整合可為遵循 AWS 最佳實務的大多數使用者提供最佳基礎。團隊可以利用 AWS 服務的耐久性、備份保護和可用性，保持遠端 Terraform 狀態的高度彈性。

遵循本節中的後端建議將導致更多協作的 Terraform 代碼庫，同時限制錯誤或未經授權修改的影響。透過實作架構良好的遠端後端，團隊可以最佳化 Terraform 工作流程。

最佳做法：

- 使用 Amazon S3 進行遠端儲存
- 促進團隊協作
- 分隔每個環境的後端
- 主動監控遠端狀態活動

使用 Amazon S3 進行遠端儲存

在 Amazon S3 中遠端存放 Terraform 狀態，並使用 Amazon DynamoDB 實作狀態鎖定和一致性檢查，可提供相較於本機檔案儲存的主要優點。遠端狀態支援團隊協同合作、變更追蹤、備份保護和遠端鎖定，以提高安全性。

將 Amazon S3 與 S3 標準儲存類別 (預設) 搭配使用，而不是暫時的本機儲存或自我管理解決方案，可提供 99.999999999% 的耐久性和 99.99% 的可用性保護，以防止意外狀態資料遺失。 AWS 受管服務

(例如 Amazon S3 和 DynamoDB) 提供的服務層級協定 (SLA) 超出大多數組織自行管理儲存時所能達到的效果。依靠這些保護來保持遠端後端的可訪問性。

啟用遠端狀態鎖定

DynamoDB 鎖定會限制狀態存取，以防止並行寫入作業。這樣可以防止多個使用者同時修改，並減少錯誤。

具有狀態鎖定的後端配置示例：

```
terraform {
  backend "s3" {
    bucket         = "myorg-terraform-states"
    key            = "myapp/production/tfstate"
    region         = "us-east-1"
    dynamodb_table = "TerraformStateLocking"
  }
}
```

啟用版本控制和自動備份

如需額外的防護功能，請 AWS Backup 在 Amazon S3 後端上使用啟用自動版本控制和備份。每當進行變更時，版本控制都會保留狀態的所有先前版本。它還可讓您在需要時還原先前的工作狀態快照，以復原不必要的變更或從事故中復原。

視需要還原先前的版本

版本化的 Amazon S3 狀態儲存貯體可透過還原先前已知的良好狀態快照，輕鬆還原變更。這有助於防止意外變更，並提供額外的備份功能。

使用醫護機構地形

HCP Terraform 提供完全受控的後端替代方案，可讓您設定自己的狀態儲存體。HCP Terraform 會自動處理安全的狀態和加密儲存，同時解鎖其他功能。

當您使用 HCP Terraform 時，默認情況下會遠程存儲狀態，從而使整個組織的狀態共享和鎖定。詳細的原則控制可協助您限制狀態存取和變更。

其他功能包括版本控制整合、原則護欄、工作流程自動化、變數管理，以及與 SAML 的單一登入整合。您也可以使用 Sentinel 原則做為程式碼來實作治理控制。

雖然 HCP Terraform 需要使用軟體即服務 (SaaS) 平台，但對於許多團隊而言，安全性、存取控制、自動化政策檢查和協作功能方面的好處，使其成為使用 Amazon S3 或 DynamoDB 自我管理狀態儲存的最佳選擇。

輕鬆地與服務（例如 GitHub 和次要 GitLab 配置）集成也吸引了完全採用雲和 SaaS 工具的用戶，以實現更好的團隊工作流程。

促進團隊協作

使用遠端後端在 Terraform 團隊的所有成員之間共用狀態資料。這有助於協同合作，因為它使整個團隊可以看到基礎結構變化。共用後端協定與狀態歷程記錄透明度相結合，可簡化內部變更 所有基礎架構變更都會透過既定的管道進行，從而提高整個企業的業務敏捷性。

透過使用改善責任 AWS CloudTrail

AWS CloudTrail 與 Amazon S3 儲存貯體整合，以擷取對狀態儲存貯體發出的 API 呼叫。過濾要跟踪`PutObject`的CloudTrail 事件以`DeleteObject`,及其他相關呼叫。

CloudTrail 記錄會顯示每個 API 呼叫狀態變更的主體 AWS 識別。使用者的身分可以與機器帳戶或與後端儲存區互動的團隊成員進行比對。

結合 CloudTrail 日誌與 Amazon S3 狀態版本控制，將基礎設施變更與套用它們的主體聯繫起來。透過分析多個修訂，您可以將任何更新歸因於機器帳戶或負責的專案團隊成員。

如果發生意外或破壞性變更，狀態版本控制會提供復原功能。 CloudTrail 追蹤使用者的變更，以便您討論預防性的改進。

我們也建議您強制執行 IAM 許可，以限制狀態值區存取權限。整體而言，S3 版本控制和 CloudTrail 監控支援跨基礎設施變更的稽核 團隊在 Terraform 狀態歷史記錄中獲得更高的責任、透明度和審核能力。

分隔每個環境的後端

針對每個應用程式環境使用不同的 Terraform 後端。獨立後端可隔離開發、測試和生產之間的狀態。

減少影響範圍

隔離狀態有助於確保較低環境中的變更不會影響生產基礎架構。在開發和測試環境中的事故或實驗的影響有限。

限制生產存取

將生產狀態後端的權限鎖定為大多數使用者的唯讀存取權。限制可以將生產基礎架構修改為 CI/CD 管線並破壞玻璃角色的人員。

簡化存取控制

在後端層級管理權限可簡化環境之間的存取控制。針對每個應用程式和環境使用不同的 S3 儲存貯體，意味著可以在整個後端儲存貯體授與廣泛的讀取或寫入許可。

避免共用工作區

雖然您可以使用 Terraform 工作區來分隔環境之間的狀態，但不同的後端提供更強的隔離能力。如果您有共用工作區，事故仍可能影響多個環境。

保持環境後端完全隔離，可將任何單一故障或漏洞的影響降到最低。獨立的後端也會將存取控制與環境的敏感度等級保持一致。例如，您可以為生產環境提供寫入保護，並為開發和測試環境提供更廣泛的存取權限。

主動監控遠端狀態活動

持續監控遠端狀態活動對於及早偵測潛在問題至關重要。尋找異常的解鎖、變更或存取嘗試。

獲取有關可疑解鎖的警報

大多數狀態變更應該透過 CI/CD 管線執行。如果狀態解鎖直接通過開發人員工作站發生，則生成警報，這可能發出未經授權或未經測試的更改

監控存取嘗試

狀態存儲桶上的身份驗證失敗可能表示偵察活動。請注意，如果多個帳戶嘗試訪問狀態，或出現異常的 IP 地址，這表明憑據洩露。

程式碼基底結構與組織的最佳做法

隨著 Terraform 在大型團隊和企業中的使用量增加，正確的代碼庫結構和組織至關重要。架構良好的程式碼基底能夠大規模協同合作，同時提升可維護性。

本節提供有關支援品質和一致性的 Terraform 模組化、命名慣例、文件和編碼標準的建議。

指導方針包括按環境和元件將組態分解為可重複使用的模組、使用前綴和尾碼建立命名慣例、記錄模組並清楚地解釋輸入和輸出，以及使用自動樣式檢查套用一致的格式化規則。

其他最佳做法涵蓋以邏輯方式組織結構階層中的模組和資源、在文件中編目公開和私有模組，以及在模組中抽取不必要的實作詳細資訊以簡化使用。

藉由實作有關模組化、文件、標準和邏輯組織的程式碼基底結構準則，您可以支援跨團隊的廣泛協同作業，同時讓 Terraform 在整個組織中的使用量分散時保持可維護。透過強制執行慣例和標準，您可以避免分散程式碼基底的複雜性。

最佳做法：

- 實作標準儲存庫結構
- 模塊化的結構
- 遵循命名慣例
- 使用附件資源
- 使用預設標籤
- 符合地形登錄要求
- 使用建議的模組來源
- 遵循編碼標準

實作標準儲存庫結構

我們建議您實作下列儲存庫配置。跨模組標準化這些一致性實務可提高可探索性、透明度、組織性和可靠性，同時在許多 Terraform 組態中重複使用。

- 根模塊或目錄：這應該是 Terraform 根模塊和可重複使用模塊的主要入口點，並且預計是唯一的。如果你有一個更複雜的架構，你可以使用嵌套模塊來創建輕量級抽象。這可協助您描述基礎結構的架構，而不是直接，就實體物件而言。

- 自述文件：根模塊和任何嵌套模塊應該具有自述文件。此檔案必須命名`README.md`。它應該包含模塊的描述以及它應該用於什麼。如果要包含將此模塊與其他資源一起使用的示例，請將其放在`examples`目錄中。請考慮加入描述模組可能建立的基礎結構資源及其關係的圖表。使用地形文檔來自動生成模塊的輸入或輸出。
- main.tf：這是主要的入口點。對於一個簡單的模塊，所有的資源可以在這個文件中創建。對於複雜的模塊，資源創建可能會分散在多個文件中，但任何嵌套模塊調用都應該在文`main.tf`件中。
- 變量 .tf 和輸出 .tf：這些文件包含變量和輸出的聲明。所有變量和輸出都應該有一句或兩句話的描述來解釋它們的目的。這些描述用於文件。如需詳細資訊，請參閱變數組態和輸出組態的 HashiCorp 文件。
  - 所有變數都必須有定義的類型。
  - 變數宣告也可以包含預設引數。如果宣告包含預設引數，則會將變數視為選用性，如果您在呼叫模組或執行 Terraform 時未設定值，則會使用預設值。預設引數需要常值，且無法參考組態中的其他物件。若要使變數成為必要項目，請省略變數宣告中的預設值，並考慮設定是否有`nullable = false`意義。
  - 對於具有與環境無關的值 (例如`disk_size`) 的變數，請提供預設值。
  - 對於具有環境特定值 (例如`project_id`) 的變數，請勿提供預設值。在這種情況下，調用模塊必須提供有意義的值。
  - 僅當變量保持空白是基礎 API 不拒絕的有效偏好設置時，才對空字符串或列表等變量使用空默認值。
  - 明智地使用變量。僅當每個例證或環境都必須有所不同時，才能將其參數化。當您決定是否公開變數時，請確定您有變更該變數的具體使用案例。如果只有一個很小的可能性可能需要一個變量，不要公開它。
    - 使用默認值添加變量是向後兼容的。
    - 移除變數是向後不相容的。
    - 在多個位置重複使用常值的情況下，您應該使用局部值而不將其公開為變數。
  - 不要直接通過輸入變量傳遞輸出，因為這樣做可以防止它們正確添加到依賴關係圖中。若要確保建立隱含相依性，請確定輸出參考資源中的屬性。而不是直接引用實例的輸入變量，而是傳遞屬性。
- locals.tf：此檔案包含將名稱指派給運算式的本機值，因此可以在模組內多次使用名稱，而不必重複運算式。局部值就像一個函數的臨時局部變量。區域值中的運算式不限於常數；它們也可以參考模組中的其他值，包括變數、資源屬性或其他區域值，以便合併它們。
- 供應商 .tf：此文件包含地形塊和提供程序塊。 `provider`塊必須僅在根模塊中由模塊的消費者聲明。

如果您使用的是 HCP 地形，還要添加一個空的雲塊。cloud區塊應完全透過環境變數和環境變數認證來設定，做為 CI/CD 管線的一部分。

- 版本 .tf：此文件包含所需的提供程序塊。所有 Terraform 模塊都必須聲明它需要哪些提供程序，以便 Terraform 可以安裝和使用這些提供程序。
- data.tf：對於簡單的配置，請將數據源放在引用它們的資源旁邊。例如，如果您要擷取要在啟動執行個體時使用的影像，請將它放置在執行個體旁邊，而不是在自己的檔案中收集資料資源。如果資料來源的數量過大，請考慮將其移至專用data.tf檔案。
- .tfvars 檔案：對於根模組，您可以使用檔案提供非敏感變數。.tfvars為了保持一致性，請命名變數檔案terraform.tfvars。將通用值放在存放庫的根目錄，並將環境特定的值放在資料夾中envs/。
- 嵌套模塊：嵌套模塊應該存在於modules/子目錄下。任何具有的嵌套模塊都README.md被外部用戶認為可用。如果README.md不存在，該模塊被視為僅供內部使用。嵌套模塊應該被用來複雜的行為分成多個小模塊，用戶可以仔細挑選和選擇。

  如果根模塊包括對嵌套模塊的調用，則這些調用應該使用相對路徑，例如，./modules/sample-module以便 Terraform 將其視為同一存儲庫或軟件包的一部分，而不是單獨下載它們。

  如果存儲庫或軟件包包含多個嵌套模塊，則理想情況下它們應該由調用者組合，而不是直接互相調用並創建一個深度嵌套的模塊樹。
- 示例：使用可重複使用模塊的示例應該存在於存儲庫根examples/目錄下的子目錄下。對於每個示例，您可以添加 README 來解釋示例的目標和用法。子模塊的實例也應放在根目examples/錄中。

  由於範例通常會複製到其他儲存庫中進行自訂，因此模組區塊應將其來源設定為外部呼叫者將使用的位址，而不是相對路徑。
- 服務命名文件：用戶通常希望在多個文件中按服務分隔 Terraform 資源。這種做法應該盡可能地不鼓勵，而資源應改為定義。main.tf但是，如果資源集合 (例如 IAM 角色和政策) 超過 150 行，則最好將其分解為自己的檔案，例如iam.tf. 否則，所有資源程式碼都應在中定義main.tf。
- 自訂指令碼：僅在必要時使用指令碼。Terraform 不會考慮或管理透過指令碼建立的資源狀態。只有當 Terraform 資源不支援所需的行為時，才使用自訂指令碼。將由 Terraform 調用的自定義腳本放在目錄中scripts/。
- 協助程式指令碼：組織目錄中未由 Terraform 呼叫的輔助程式指令碼。helpers/在文件中記錄幫助程序腳本，其README.md中包含解釋和示例調用。如果輔助腳本接受參數，請提供參數檢查和--help輸出。

- 靜態文件：Terraform 引用但不運行的靜態文件（例如，加載到 EC2 實例的啟動腳本）必須組織到一個`files/`目錄中。將冗長的文件放置在外部檔案中，與其 HCL 分開。使用文件（）函數引用它們。
- 範本：對於 Terraform 範本檔案函數讀取的檔案，請使用檔案副檔名。`.tftpl`範本必須放置在`templates/`目錄中。

根模塊結構

Terraform 總是在單個根模塊的上下文中運行。完整的 Terraform 配置由根模塊和子模塊的樹（其中包括根模塊調用的模塊，這些模塊調用的任何模塊等）組成。

地形根模塊佈局基本實例：

```
.
### data.tf
### envs
#   ### dev
#   #   ### terraform.tfvars
#   ### prod
#   #   ### terraform.tfvars
#   ### test
#       ### terraform.tfvars
### locals.tf
### main.tf
### outputs.tf
### providers.tf
### README.md
### terraform.tfvars
### variables.tf
### versions.tf
```

可重用模塊結構

可重複使用的模塊遵循與根模塊相同的概念。要定義模塊，請為其創建一個新目錄並將`.tf`文件放在其中，就像定義根模塊一樣。Terraform 可以從本地相對路徑或遠程存儲庫加載模塊。如果您希望模塊被許多配置重複使用，請將其放置在其自己的版本控制存儲庫中。重要的是要保持模塊樹相對平坦，以便更容易以不同的組合重複使用模塊。

地形可重複使用的模塊佈局基本示例：

```
.
### data.tf
### examples
#   ### multi-az-new-vpc
#   #   ### data.tf
#   #   ### locals.tf
#   #   ### main.tf
#   #   ### outputs.tf
#   #   ### providers.tf
#   #   ### README.md
#   #   ### terraform.tfvars
#   #   ### variables.tf
#   #   ### versions.tf
#   #   ### vpc.tf
#   ### single-az-existing-vpc
#   #   ### data.tf
#   #   ### locals.tf
#   #   ### main.tf
#   #   ### outputs.tf
#   #   ### providers.tf
#   #   ### README.md
#   #   ### terraform.tfvars
#   #   ### variables.tf
#   #   ### versions.tf
### iam.tf
### locals.tf
### main.tf
### outputs.tf
### README.md
### variables.tf
### versions.tf
```

模塊化的結構

原則上，您可以將任何資源和其他結構組合到一個模塊中，但是過度使用嵌套和可重複使用的模塊可能會使您的整體 Terraform 配置難以理解和維護，因此請適度使用這些模塊。

如果有意義，請將您的配置分解為可重複使用的模塊，該模塊通過描述從資源類型構建的體系結構中的新概念來提高抽象水平。

當您將基礎結構模組化為可重複使用的定義時，請以邏輯資源集合為目標，而非個別元件或過於複雜的集合。

不要包裝單一資源

您不應該在其他單個資源類型周圍創建精簡包裝器的模塊。如果您無法為模塊找到與其中主要資源類型的名稱不同的名稱，那麼您的模塊可能沒有創建新的抽象-這增加了不必要的複雜性。而是直接在調用模塊中使用資源類型。

封裝邏輯關係

群組相關資源集，例如網路基礎、資料層、安全控制和應用程式。可重複使用的模組應封裝共同運作以啟用功能的基礎結構部分。

保持繼承平坦

在子目錄中嵌套模塊時，請避免深入一個或兩個以上的層級。深度巢狀的繼承結構使組態和疑難排解複雜化。模塊應該建立在其他模塊上，而不是通過它們構建隧道。

透過將模組集中在代表架構模式的邏輯資源分組上，團隊可以快速設定可靠的基礎架構基礎架構。平衡抽象，而不會過度工程或過度簡化。

輸出中的參考資源

對於在可重複使用模塊中定義的每個資源，至少包含一個引用該資源的輸出。變數和輸出可讓您推斷模組與資源之間的相依性。如果沒有任何輸出，用戶將無法根據其 Terraform 配置正確訂購您的模塊。

結構良好的模組可提供環境一致性、目的導向的群組以及匯出的資源參考，可讓組織範圍內的 Terraform 大規模協同合作。團隊可以從可重複使用的建構區塊組合基礎

不要設定提供者

雖然共享模塊從調用模塊繼承提供程序，但模塊不應該自己配置提供程序設置。避免在模組中指定提供者組態區塊。此配置應該只在全局聲明一次。

聲明所需提供者

雖然提供者配置在模塊之間共享，但共享模塊還必須聲明自己的提供者需求。此作法可讓 Terraform 確保有單一版本的提供者與組態中的所有模組相容，並指定來源位址，以做為提供者的全域 (模組無關) 識別碼。但是，特定於模組的提供者需求不會指定任何決定提供者將存取的遠端端點的組態設定，例如. AWS 區域

通過聲明版本要求並避免硬編碼提供程序配置，模塊使用共享提供程序在 Terraform 配置之間提供可移植性和可重用性。

對於共享模塊，請在中的 required_providers 塊中定義所需的最低提供程序版本。versions.tf

要聲明模塊需要特定版本的 AWS 提供程序，請在required_providers塊內使用terraform塊：

```
terraform {
  required_version = ">= 1.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 4.0.0"
    }
  }
}
```

如果共享模塊僅支持特定版本的 AWS 提供程序，請使用悲觀約束運算符（~>），該運算符只允許最右側的版本組件增加：

```
terraform {
  required_version = ">= 1.0.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}
```

在此範例中，~> 4.0允許安裝4.57.1和 (4.67.0但不允許) 5.0.0。如需詳細資訊，請參閱文件中的版本條 HashiCorp 件約束語法。

遵循命名慣例

清晰的描述性名稱可簡化您對模組中資源之間的關係以及組態值用途的理解。與樣式指南的一致性增強了模塊用戶和維護者的可讀性。

遵循資源命名的準則

- 對所有資源名稱使用 snake_case（其中小寫字詞由下劃線分隔），以符合 Terraform 樣式標準。此做法可確保與資源類型、資料來源類型和其他預先定義值的命名慣例保持一致。此約定不適用於名稱引數。
- 若要簡化對資源類型中唯一資源的參照 (例如，整個模組的單一負載平衡器)，請為資源命名`main`或`this`為清楚起見。
- 使用有意義的名稱來描述資源的用途和內容，並有助於區分類似資源 (例如，`primary`針對主資料庫和資料庫的僅`read_replica`供讀取複本)。
- 使用單數，而不是複數名稱。
- 請勿在資源名稱中重複資源類型。

遵循變數命名的準則

- 將單位新增至輸入名稱、區域變數和代表數值的輸出，例如磁碟大小或 RAM 大小 (例如，`ram_size_gb`RAM 大小以 GB 為單位)。此做法可讓設定維護者清楚預期的輸入單元。
- 儲存區大小使用二進位單位 (例如 MiB 和 GiB)，使用十進位單位 (例如 MB 或 GB) 表示其他量度。
- 給布爾變量正數名稱，如`enable_external_access`。

使用附件資源

某些資源將偽資源作為其中的屬性內嵌。在可能的情況下，您應該避免使用這些嵌入式資源屬性，並改用唯一資源來附加該虛擬資源。這些資源關係可能會導 cause-and-effect 致每個資源都是唯一的問題。

使用嵌入的屬性（避免這種模式）：

```
resource "aws_security_group" "allow_tls" {
  ...
  ingress {
    description      = "TLS from VPC"
    from_port        = 443
    to_port          = 443
    protocol         = "tcp"
    cidr_blocks      = [aws_vpc.main.cidr_block]
    ipv6_cidr_blocks = [aws_vpc.main.ipv6_cidr_block]
  }
```

```
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }
}
```

使用附件資源（首選）：

```
resource "aws_security_group" "allow_tls" {
  ...
}

resource "aws_security_group_rule" "example" {
  type              = "ingress"
  description      = "TLS from VPC"
  from_port        = 443
  to_port          = 443
  protocol         = "tcp"
  cidr_blocks      = [aws_vpc.main.cidr_block]
  ipv6_cidr_blocks = [aws_vpc.main.ipv6_cidr_block]
  security_group_id = aws_security_group.allow_tls.id
}
```

使用預設標籤

將標籤指派給所有可接受標籤的資源。地形表單 AWS 提供者有一個 aws_default_tags 資料來源，您應該在根模組內使用它。

請考慮將必要的標籤新增至 Terraform 模組所建立的所有資源。以下是可能附加的標籤清單：

- 名稱：人類可讀的資源名稱
- AppId：使用資源之應用程式的 ID
- AppRole: 資源的技術功能; 例如，「網絡服務器」或「數據庫」
- AppPurpose：資源的業務目的；例如，「前端 UI」或「付款處理者」
- 環境：軟體環境，例如開發、測試或生產
- 專案：使用資源的專案

- CostCenter：向誰開立資源使用量帳單

符合地形登錄要求

模塊存儲庫必須滿足以下所有要求，以便可以將其發佈到 Terraform 註冊表。

即使您不打算在短期內將模組發佈到登錄，也應始終遵循這些要求。這樣，您可以稍後將模組發佈到登錄，而不必變更存放庫的組態和結構。

- 存放庫名稱：對於模組儲存庫，請使用三部分名稱`terraform-aws-<NAME>`，其中`<NAME>`反映模組管理的基礎架構類型。`<NAME>`區段可以包含其他連字號 (例如，`terraform-aws-iam-terraform-roles`)。
- 標準模塊結構：模塊必須遵守標準存儲庫結構。這可讓登錄檢查您的模組並產生文件、追蹤資源使用情況等。
  - 建立 Git 儲存庫之後，將模組檔案複製到儲存庫的根目錄。我們建議您將每個要重複使用的模組放在其本身儲存庫的根目錄中，但您也可以參考子目錄中的模組。
  - 如果您使用的是 HCP Terraform，請將要共用的模組發佈到您的組織登錄。註冊表使用 HCP Terraform API 令牌處理下載和控制訪問，因此即使消費者從命令行運行 Terraform，也不需要訪問模塊的源代碼存儲庫。
- 位置和權限：存放庫必須位於您設定的版本控制系統 (VCS) 提供者之一，且 HCP Terraform VCS 使用者帳戶必須具有存放庫的管理員存取權限。註冊表需要管理員訪問權限才能創建 Webhook 以導入新的模塊版本。
- x.y.z 發行版本標籤：您必須至少有一個發行標籤才能發佈模組。登錄會使用版本標記來識別模組版本。發行版本標籤名稱必須使用語意版本控制，您可以選擇性地使用 v (例如`v1.1.0`和`1.1.0`) 作為前綴。登錄會忽略看起來不像版本號碼的標記。如需有關發行模組的詳細資訊，請參閱 T erraform 文件。

如需詳細資訊，請參閱 Terraform 文件中的準備模組存放庫。

使用建議的模組來源

Terraform 使用模塊中的`source`參數來查找和下載子模塊的源代碼。

我們建議您針對密切相關的模組使用本機路徑，這些模組的主要目的是分解重複的程式碼元素，並針對多個組態共用的模組使用原生 Terraform 模組登錄或 VCS 提供程式。

下列範例說明共用模組最常見和建議的來源類型。登錄模組支援版本控制。您應該一律提供特定版本，如下列範例所示。

登錄檔

地形註冊表：

```
module "lambda" {
  source = "github.com/terraform-aws-modules/terraform-aws-lambda.git?
ref=e78cdf1f82944897ca6e30d6489f43cf24539374" #--> v4.18.0

  ...

}
```

通過固定提交哈希值，您可以避免從容易受到供應鏈攻擊的公共註冊表中漂移。

醫護機構地形：

```
module "eks_karpenter" {
  source = "app.terraform.io/my-org/eks/aws"
  version = "1.1.0"

  ...

  enable_karpenter = true
}
```

地形企業：

```
module "eks_karpenter" {
  source = "terraform.mydomain.com/my-org/eks/aws"
  version = "1.1.0"

  ...

  enable_karpenter = true
}
```

VCS 供應商

VCS 提供者支援用於選取特定修訂版本的`ref`引數，如下列範例所示。

GitHub (HTTPS):

```
module "eks_karpenter" {
  source = "github.com/my-org/terraform-aws-eks.git?ref=v1.1.0"

  ...

  enable_karpenter = true
}
```

通用 Git 存儲庫（HTTPS）：

```
module "eks_karpenter" {
  source = "git::https://example.com/terraform-aws-eks.git?ref=v1.1.0"

  ...

  enable_karpenter = true
}
```

通用 Git 存儲庫（SSH）：

> **Warning**
>
> 您需要配置憑據才能訪問私有存儲庫。

```
module "eks_karpenter" {
  source = "git::ssh://username@example.com/terraform-aws-eks.git?ref=v1.1.0"

  ...

  enable_karpenter = true
}
```

遵循編碼標準

在所有配置文件中應用一致的 Terraform 格式規則和樣式。在 CI/CD 管線中使用自動樣式檢查來強制執行標準。當您將編碼最佳實務嵌入到團隊工作流程中時，隨著使用情況廣泛傳播到整個組織中，配置仍然可以保持可讀性、可維護和協作。

遵循風格指南

- 使用 terraform fmt 指令來格式化所有地形.tf檔案 (檔案)，以符合樣式標準。 HashiCorp
- 使用地形驗證命令來驗證配置的語法和結構。
- 使用 T Flint 靜態分析程式碼品質。此絨毛器會檢查 Terraform 的最佳實踐，而不僅僅是格式化，並在遇到錯誤時失敗構建。

配置提交前掛鉤

在允許認可之前，先設定執行`terraform fmt`、`tflint`和`checkov`、和其他程式碼掃描和樣式檢查的用戶端提交前掛接。此做法可協助您在開發人員工作流程中早期驗證標準一致性。

使用預先提交的框架（例如預先提交）將 Terraform 線條，格式化和代碼掃描添加為本地計算機上的掛鉤。鉤子在每個 Git 提交上運行，如果檢查未通過，則提交失敗。

將樣式和品質檢查移至本機預先提交掛鉤可在變更之前向開發人員提供快速回饋。標準成為編碼工作流程的一部分。

AWS 提供者版本管理的最佳做法

仔細管理 AWS 提供者和相關 Terraform 模組的版本對於穩定性至關重要。本節概述有關版本限制和升級的最佳作法。

最佳做法：

- 新增自動版本檢查
- 監控新版本
- 貢獻給供應商

新增自動版本檢查

在 CI/CD 管道中新增 Terraform 提供者的版本檢查以驗證版本釘選，如果版本未定義，則建置失敗。

- 在 CI/CD 管線中新增 TFlint 檢查，以掃描未定義主要/次要版本限制的提供者版本。使用 Terraform 提供 AWS 者專用的 Tflint 規則集外掛程式，它提供偵測可能錯誤的規則，並檢查有關資源的最佳作法。 AWS
- 偵測未釘選的提供者版本的 CI 執行失敗，以防止隱式升級到達生產環境。

監控新版本

- 監控提供者版本說明和變更記錄摘要。獲取有關新主要/次要版本的通知。
- 評估發行說明是否有潛在的突破性變更，並評估其對現有基礎架構的影響。
- 請先升級非生產環境中的次要版本，以便在更新生產環境之前對其進行驗證。

透過自動化管道中的版本檢查並監控新版本，您可以在更新生產環境之前提早 catch 不受支援的升級，並讓團隊有時間評估新主要/次要版本的影響。

貢獻給供應商

通過報告缺陷或請求 GitHub 問題中的功能來積極為 HashiCorp AWS Provider 做出貢獻：

- 在 AWS Provider 儲存庫上開啟詳細記錄的問題，以詳細說明您遇到的任何錯誤或遺失的功能。提供可重複的步驟。

- 請求並對增強功能進行投票，以擴展 AWS 供應商管理新服務的功能。
- 當您提供提供提供者瑕疵或增強功能的提議修正時，參考已發出的提取 相關問題的連結。
- 請遵循存放庫中的編碼慣例、測試標準和文件的貢獻準則。

通過回饋您使用的提供商，您可以直接在其藍圖中提供意見，並為所有用戶提高其質量和能力。

社群模組的最佳做法

有效使用模組是管理複雜 Terraform 組態和促進重複使用的關鍵。本節提供有關社區模塊，依賴關係，源，抽象和貢獻的最佳實踐。

最佳做法：

- 探索社群模組
- 瞭解相依性
- 使用信任的來源
- 為社區模塊做出貢獻

探索社群模組

在構建新模塊之前 GitHub，請在 Terraform 註冊表和其他源中搜索可能解決您的用例的現有 AWS 模塊。尋找具有最新更新並積極維護的熱門選項。

使用變數進行自訂

當您使用社區模塊時，請通過變量傳遞輸入，而不是分叉或直接修改源代碼。在需要時覆蓋默認值，而不是更改模塊的內部。

分叉應僅限於為原始模塊提供修復程序或功能，以使更廣泛的社區受益。

瞭解相依性

在您使用模組之前，請先檢閱其原始程式碼和說明文件，以識別相依性：

- 必要提供者：請注意模組需要的 AWS、Kubernetes 或其他提供者的版本。
- 嵌套模塊：檢查內部使用的其他引入級聯依賴關係的模塊。
- 外部資料來源：記下模組所依賴的 API、自訂外掛程式或基礎結構相依性。

通過映射出直接和間接依賴關係的完整樹，可以避免在使用模塊時出現意外。

使用信任的來源

來自未經驗證或未知的發布商採購 Terraform 模塊會帶來重大風險。僅使用來自信任來源的模組。

- 喜歡由經過驗證的創建者（例如 AWS 或 HashiCorp 合作夥伴）發布的 Terraform 註冊表中的認證模塊。
- 對於自訂模組，請檢閱發行者歷史記錄、支援等級和使用信譽，即使該模組來自您自己的組織也一樣。

透過不允許來自未知或已公開來源的模組，您可以降低將弱點或維護問題注入程式碼的風險。

訂閱 通知

訂閱來自受信任的發行者之新模組版本的通知：

- 觀看 GitHub 模塊存儲庫以獲取有關模塊新版本的警報。
- 監控發布者博客和更新日誌以獲取更新。
- 從經過驗證的高度評價來源獲取新版本的主動通知，而不是隱式提取更新。

僅使用來自信任來源的模組並監控變更，可提供穩定性和安全性。經過審核的模組可提高生產力，同時將供應鏈風險降

為社區模塊做出貢獻

針對以下位置託管的社群模組提交修正和增強功能 GitHub：

- 在模塊上打開提取請求，以解決您在使用中遇到的缺陷或限制。
- 建立問題，請求將新的最佳作法組態新增至現有的 OSS 模組。

為社區模塊做出貢獻，可為所有 Terraform 從業者增強可重複使用，編纂模式。

常見問答集

問：為什麼要專注於 AWS 供應商？

答： AWS 提供者是 Terraform 中用於佈建基礎設施的最廣泛且最複雜的提供者之一。遵循這些最佳作法可協助使用者最佳化其 AWS 環境中提供者的使用方式。

問：我是地形的新手。我可以使用本指南嗎？

答：該指南適用於 Terraform 的新手以及想要提高技能的更高級的從業者。這些實踐改善了用戶在任何學習階段的工作流程。

問：涵蓋哪些關鍵的最佳實務？

答：關鍵的最佳做法包括在存取金鑰上使用 IAM 角色、固定版本、整合自動化測試、遠端狀態鎖定、憑證輪換、貢獻給提供者，以及邏輯組織程式碼庫。

問：我可以在哪裡進一步了解地形？

答：資源部分包含 HashiCorp Terraform 官方文檔和社區論壇的鏈接。使用這些連結進一步瞭解進階 Terraform 工作流程。

後續步驟

閱讀本指南後，以下是一些潛在的後續步驟：

- 如果您有現有的 Terraform 程式碼庫，請檢閱您的組態，並根據本指南中提供的建議找出可以改善的區域。例如，檢閱實作遠端後端、將程式碼分隔為模組、使用版本釘選等的最佳做法，以及在設定中驗證這些後端。
- 如果您沒有現有的 Terraform 代碼庫，請在構建新配置時使用這些最佳實踐。從一開始就遵循有關狀態管理，身份驗證，代碼結構等的建議。
- 嘗試使用本指南中引用的一些 HashiCorp 社區模塊，看看它們是否簡化了您的架構模式。這些模塊允許更高級別的抽象，因此您不必重寫通用資源。
- 啟用 linting、安全性掃描、原則檢查和自動化測試工具，強化有關安全性、合規性和程式碼品質的一些最佳作法。諸如火石，tfsec 和切科夫之類的工具可以提供幫助。
- 檢閱最新的 AWS Provider 文件，瞭解是否有任何可協助您最佳化 Terraform 使用方式的新資源或功能。掌握最新版本的 AWS 提供者。
- 如需其他指引，請參閱網站上的 Terraform 文件、最佳實務指南和樣式指南。 HashiCorp

資源

參考

以下鏈接為 Terraform 提供程 AWS 序和 IaC 使用 Terraform 提供程序提供了其他閱讀材料。 AWS

- 地形表單 AWS 提供者 (文件) HashiCorp
- 用於 AWS 服務的地形模塊（地形註冊表）
- 該 AWS 和 HashiCorp 夥伴關係（HashiCorp 博客文章）
- AWS 提供者的動態憑證 (HCP 地形文件)
- DynamoDB 狀態鎖定 (地形文件)
- 使用哨兵強制執行政策 (地形文件)

工具

下列工具可協助改善 Terraform 組態的程式碼品質和自動化 AWS，如本最佳實務指南所建議。

代碼質量：

- Checkov：在部署之前掃描地形代碼以識別錯誤的配置。
- tFlint：識別可能的錯誤、已停用的語法和未使用的宣告。這個短絨也可以強制執行 AWS 最佳實踐和命名約定。
- 地形文檔：從各種輸出格式的 Terraform 模塊生成文檔。

自動化工具：

- HCP Terraform：透過原則檢查和核准閘道，協助團隊編寫、協同合作和建立 Terraform 工作流程。
- 亞特蘭蒂斯：用於驗證代碼更改的開源 Terraform 提取請求自動化工具。
- 適用於 Terraform 的 CDK：一種架構，可讓您使用熟悉的語言，例如 Python TypeScript、Java、C# 和 Go，而不是 HashiCorp 組態語言 (HCL) 來定義、佈建和測試您的 Terraform 基礎結構做為程式碼。

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5月28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-被動遷移需要更多的工作。

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保持同步，但只有來源資料庫處理來自連接應用程式的交易，同時將資料複寫至目標資料庫。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

在資料集中永久刪除個人資訊的程序。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

您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。因此， 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)

一種工具，可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。 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，讓使用者能夠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 於部落格文章 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 遷移策略相關的詳細資訊，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)

組織之間商業文件的自動交換。如需詳細資訊，請參閱什麼是電子資料交換。

加密

一種運算程序，可將人類可讀取的純文字資料轉換為加密文字。

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界限會限制故障的影響，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，少量的提示可以有效。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，已針對廣義和未標記資料的大量資料集進行訓練。FMs 能夠執行各種一般任務，例如了解語言、產生文字和影像，以及自然語言的交談。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，並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、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)

一種 AWS 計畫，提供諮詢支援、訓練和服務，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，請參閱此詞彙表中的 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 建議使用不可變基礎設施做為最佳實務。

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

請參閱開放程序通訊 - 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 中的操作準備度審查 (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

由 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)

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。

依設計的隱私權

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。

重新定位

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

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。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。此功能會啟用聯合單一登入 (SSO)，讓使用者可以登入 AWS Management Console 或呼叫 AWS API 操作，而不必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。

單點故障 (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，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

危及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，讀取許多。

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，並防止資料遭到刪除或修改。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。