使用 Terra AWS form 提供程序的最佳实践

AWS 规范性指导

AWS 规范性指导: 使用 Terra AWS form 提供程序的最佳实践

Table of Contents

使用 Terraform AWS 提供商的最佳实践

Michael Begin，Amazon Web Services 高级 DevOps 顾问 (AWS)

2024 年 5 月（文件历史记录）

启用 Terraform 管理基础设施即代码 (IaC) AWS 可带来重要好处，例如提高一致性、安全性和敏捷性。但是，随着您的 Terraform 配置规模和复杂性的增长，遵循最佳实践以避免陷阱变得至关重要。

本指南提供了使用来自 Terraform AWS 提供程序的推荐最佳实践。 HashiCorp它会引导你了解正确的版本控制、安全控制、远程后端、代码库结构和社区提供商，以优化 Terraform。 AWS每个部分都详细介绍了应用这些最佳实践的细节：

- 安全性
- 后端
- 代码库结构和组织
- AWS 提供商版本管理
- 社区模块

目标

本指南可帮助您获得有关 Terraform Prov AWS ider 的操作知识，并通过遵循围绕安全性、可靠性、合规性和开发人员生产力的 IaC 最佳实践，您可以实现以下业务目标。

- 提高 Terraform 项目的基础设施代码质量和一致性。
- 加快开发人员入职和为基础架构代码做出贡献的能力。
- 通过更快地更改基础架构来提高业务灵活性。
- 减少与基础架构变更相关的错误和停机时间。
- 遵循 IaC 最佳实践，优化基础设施成本。
- 通过实施最佳实践，加强您的整体安全状况。

目标受众

本指南的目标受众包括技术主管和经理，他们负责监督使用 Terraform for IaC 的团队。 AWS其他潜在读者包括基础设施工程师、 DevOps 工程师、解决方案架构师和积极使用 Terraform 管理 AWS 基础设施的开发人员。

遵循这些最佳实践将节省时间，并有助于为这些角色发挥 IaC 的好处。

概述

Terraform 提供者是允许 Terraform 与不同 API 进行交互的插件。Terraform P AWS rovider 是用于使用 Terraform 管理 AWS 基础设施即代码 (IaC) 的官方插件。它将 Terraform 语法转换为 AWS API 调用，用于创建、读取、更新和删除资源。 AWS

AWS 提供程序负责处理身份验证、将 Terraform 语法转换为 AWS API 调用以及在中配置资源。 AWS 你可以使用 Terraform `provider` 代码块来配置 Terraform 用来与 API 交互的提供者插件。 AWS 您可以配置多个 P AWS rovider 区块来管理不同 AWS 账户 和地区的资源。

以下是 Terraform 配置示例，该配置使用多个带有别名的 AWS 提供程序块来管理在不同区域和账户中具有副本的亚马逊关系数据库服务 (Amazon RDS) 数据库。主要提供商和次要提供商扮演不同的 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
}
```

在本示例中：

- 第一个provider区块使用别primary名配置us-west-1区域中的主 AWS 提供商。
- 第二个provider模块使用别replica名配置us-east-1区域中的辅助 AWS 提供商。此提供程序用于在不同的区域和账户中创建主数据库的只读副本。该assume_role区块用于在副本账户中扮演 IAM 角色。role_arn指定要代入的 IAM 角色的亚马逊资源名称 (ARN)，并且session_name是 Terraform 会话的唯一标识符。
- 该aws_db_instance.primary资源使用该us-west-1地区的primary提供商创建主 Amazon RDS 数据库。
- 该aws_db_instance.read_replica资源使用replica提供程序在us-east-1区域中创建主数据库的只读副本。该replicate_source_db属性引用primary数据库的 ID。

安全最佳实操

正确管理身份验证、访问控制和安全性对于安全使用 Terraform AWS Provider 至关重要。本节概述了以下方面的最佳实践：

- IAM 角色和最低权限访问权限
- 保护凭证以帮助防止未经授权访问 AWS 账户和资源
- 远程状态加密可帮助保护敏感数据
- 扫描基础设施和源代码以识别错误配置
- 远程状态存储的访问控制
- 执行哨兵政策以实施治理护栏

在使用 Terraform 管理 AWS 基础设施时，遵循这些最佳实践有助于增强您的安全状况。

遵循最低权限原则

最低权限是一项基本的安全原则，指的是仅授予用户、进程或系统执行其预期功能所需的最低权限。它是访问控制的核心概念，也是防止未经授权的访问和潜在数据泄露的预防措施。

本节多次强调了最低权限原则，因为它直接关系到Terraform如何对云提供商进行身份验证和运行操作，例如. AWS

当您使用 Terraform 配置和管理 AWS 资源时，它代表需要适当权限才能进行 API 调用的实体（用户或角色）行事。不遵循最低权限会带来重大安全风险：

- 如果 Terraform 拥有超出所需权限的过多权限，则意想不到的配置错误可能会导致意想不到的更改或删除。
- 如果 Terraform 状态文件或凭证遭到泄露，过于宽松的访问权限会增加影响范围。
- 不遵循最低权限违背了授予最低所需访问权限的安全最佳实践和监管合规性要求。

使用 IAM 角色

尽可能使用 IAM 角色代替 IAM 用户，以增强使用 Terraform AWS 提供商的安全性。IAM 角色提供可自动轮换的临时安全证书，无需管理长期访问密钥。角色还通过 IAM 策略提供精确的访问控制。

使用 IAM 策略授予最低权限访问权限

谨慎构建 IAM 策略，确保角色和用户仅拥有其工作负载所需的最低权限集。从空策略开始，然后反复添加允许的服务和操作。要实现这一点，请执行以下操作：

- 启用 IAM Access Analyzer 来评估策略并突出显示可以删除的未使用权限。
- 手动查看政策，删除对角色的预期职责不重要的任何权能。
- 使用 IAM 策略变量和标签来简化权限管理。

精心构造的策略授予的访问权限刚好足以完成工作负载的职责，仅此而已。在操作级别定义操作，并且仅允许在特定资源上调用必需的 API。

遵循这种最佳做法可以缩小影响范围，并遵循职责分离和最小权限访问的基本安全原则。根据需要逐步开始严格和开放访问，而不是开始开放并稍后再尝试限制访问。

假设 IAM 角色进行本地身份验证

在本地运行 Terraform 时，请避免配置静态访问密钥。相反，可以使用 IAM 角色临时授予特权访问权限，而不会暴露长期证书。

首先，创建具有必要最低权限的 IAM 角色并添加信任关系，允许您的用户账户或联合身份担任 IAM 角色。这允许临时使用该角色。

信任关系策略示例：

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

然后，运行 AWS CLI 命令 aws sts assume-role 来检索该角色的短期证书。这些证书的有效期通常为一小时。

AWS CLI 命令示例：

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

该命令的输出包含访问密钥、密钥和会话令牌，您可以使用这些令牌对以下内容进行身份验证 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 提供者还可以自动处理担任该角色的问题。

担任 IAM 角色的提供商配置示例：

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

这会严格在 Terraform 会话期间授予更高的权限。临时密钥不能泄露，因为它们会在会话的最长持续时间后自动过期。

这种最佳做法的主要好处包括与长期访问密钥相比提高了安全性，对角色进行了精细的访问控制以获得最少权限，以及能够通过修改角色的权限轻松撤消访问权限。通过使用 IAM 角色，您还可以避免将机密直接存储在本地脚本或磁盘上，这有助于您在团队中安全地共享 Terraform 配置。

使用 IAM 角色进行 Amazon EC2 身份验证

当您从亚马逊弹性计算云 (Amazon EC2) 实例运行 Terraform 时，请避免在本地存储长期证书。而是使用 IAM 角色和实例配置文件自动授予最低权限权限。

首先，创建具有最低权限的 IAM 角色并将该角色分配给实例配置文件。实例配置文件允许 EC2 实例继承角色中定义的权限。然后，通过指定该实例配置文件来启动实例。该实例将通过附加的角色进行身份验证。

在运行任何 Terraform 操作之前，请验证该角色是否存在于实例元数据中，以确认证书已成功继承。

```
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/
```

这种方法可以避免将永久 AWS 密钥硬编码到实例中的脚本或 Terraform 配置中。临时证书可通过实例角色和配置文件透明地提供给 Terraform。

这种最佳实践的主要好处包括提高长期证书的安全性、减少凭证管理开销以及开发、测试和生产环境之间的一致性。IAM 角色身份验证简化了 Terraform 从 EC2 实例运行，同时强制执行最低权限访问权限。

为 HCP Terraform 工作空间使用动态凭据

HCP Terraform 是一项由提供的托管服务 HashiCorp ，可帮助团队使用 Terraform 在多个项目和环境中配置和管理基础架构。在 HCP Terraform 中运行 Terraform 时，请使用动态凭据来简化和保护身份验证。 AWS Terraform 在每次运行时都会自动交换临时证书，无需担任 IAM 角色。

好处包括更轻松地轮换密钥、跨工作空间的集中式凭证管理、最低权限以及取消硬编码密钥。与长期访问密钥相比，依赖经过哈希处理的临时密钥可以提高安全性。

在中使用 IAM 角色 AWS CodeBuild

在中 AWS CodeBuild，使用分配给 CodeBuild 项目的 IAM 角色运行您的构建。这允许每个版本自动继承角色的临时证书，而不是使用长期密钥。

在 HCP Terraform 上远程运行 GitHub 操作

将 GitHub 操作工作流配置为在 HCP Terraform 工作空间上远程运行 Terraform。依赖动态凭证和远程状态锁定，而不是 GitHub 密钥管理。

在 OIDC 中使用 GitHub 操作并配置 “ AWS 凭证” 操作

使用 OpenID Connect (OIDC) 标准通过 IAM 联合操作身份。 GitHub 使用 “配置 AWS 证书” 操作将 GitHub 令牌交换为临时 AWS 证书，而无需长期访问密钥。

GitLab 与 OIDC 和 AWS CLI

使用 OIDC 标准通过 IAM 联合 GitLab 身份以进行临时访问。通过依赖 OIDC，您无需直接管理其中的长期 AWS 访问密钥。 GitLab交换凭证 just-in-time，这提高了安全性。根据 IAM 角色中的权限，用户还可以获得最低权限访问权限。

将独一无二的 IAM 用户与传统自动化工具配合使用

如果您的自动化工具和脚本缺乏对使用 IAM 角色的原生支持，则可以创建单个 IAM 用户来授予编程访问权限。最低权限原则仍然适用。最大限度地减少策略权限，并依赖每个管道或脚本使用不同的角色。当您迁移到更现代的工具或工具时，请开始以原生方式支持角色，然后逐渐过渡到这些角色或工具。

> **Warning**
>
> IAM 用户拥有长期证书，这会带来安全风险。为帮助减轻这种风险，我们建议仅向这些用户提供执行任务所需的权限，并在不再需要这些用户时将其移除。

使用 Jenkins AWS 凭证插件

使用 Jenkins 中的AWS 凭据插件集中配置 AWS 凭据并将其动态注入构建中。这样可以避免在源代码管理中检查机密。

持续监控、验证和优化最低权限

随着时间的推移，授予的额外权限可能会超过所需的最低策略。持续分析访问权限以识别和删除任何不必要的权利。

持续监控访问密钥的使用情况

如果您无法避免使用访问密钥，请使用 IAM 凭证报告来查找超过 90 天的未使用访问密钥，并撤消用户账户和计算机角色中的非活动密钥。提醒管理员手动确认移除在职员工和系统的密钥。

监控密钥使用情况可以帮助您优化权限，因为您可以识别和删除未使用的授权。当您遵循访问密钥轮换的最佳实践时，它会限制凭据的有效期并强制执行最低权限访问。

AWS 提供了多种服务和功能，可用于为管理员设置警报和通知。以下是一些选项：

- AWS Config：您可以使用 AWS Config 规则来评估 AWS 资源的配置设置，包括 IAM 访问密钥。您可以创建自定义规则来检查特定条件，例如超过特定天数的未使用访问密钥。当违反规则时， AWS Config 可以开始评估补救措施或向亚马逊简单通知服务 (Amazon SNS) Simple SNS Service 主题发送通知。
- AWS Security Hub: Security Hub 可全面了解您的 AWS 账户的安全状况，可以帮助检测和通知您潜在的安全问题，包括未使用或不活跃的 IAM 访问密钥。Security Hub 可以与亚马逊 EventBridge 和亚马逊 SNS 集成 AWS Chatbot ，也可以向管理员发送通知。
- AWS Lambda：Lambda 函数可以由各种事件调用，包括亚马逊 CloudWatch 事件或 AWS Config 规则。您可以使用诸如 Amazon SNS 或之类的服务编写自定义 Lambda 函数来评估 IAM 访问密钥的使用情况、执行其他检查和发送通知。 AWS Chatbot

持续验证 IAM 政策

使用 IAM Access Analyzer 评估附加到角色的策略，并识别任何未使用的服务或已授予的多余操作。实施定期访问审查，以手动验证策略是否符合当前要求。

将现有策略与 IAM Access Analyzer 生成的策略进行比较，并删除所有不必要的权限。您还应该向用户提供报告，并在宽限期过后自动撤销未使用的权限。这有助于确保最低限度的策略仍然有效。

主动且频繁地撤销过时的访问权限可以最大限度地减少在泄露期间可能面临风险的凭证。自动化提供了可持续、长期的凭证卫生和权限优化。遵循这一最佳实践，通过主动对 AWS 身份和资源强制执行最低权限来限制影响范围。

安全的远程状态存储

远程状态存储是指远程存储 Terraform 状态文件，而不是在运行 Terraform 的计算机上本地存储。状态文件至关重要，因为它可以跟踪 Terraform 提供的资源及其元数据。

未能保护远程状态可能会导致严重的问题，例如状态数据丢失、无法管理基础架构、无意中删除资源以及状态文件中可能存在的敏感信息泄露。因此，保护远程状态存储对于生产级 Terraform 的使用至关重要。

启用加密和访问控制

使用亚马逊简单存储服务 (Amazon S3) Simple Service 服务器端加密 (SSE) 对远程静态状态进行加密。

限制直接访问协作工作流程

- 在 HCP Terraform 或 Git 存储库中的 CI/CD 管道中构建协作工作流程，以限制直接访问状态。
- 依靠拉取请求、运行批准、策略检查和通知来协调变更。

遵循这些准则有助于保护敏感的资源属性，并避免与团队成员的更改发生冲突。加密和严格的访问保护有助于减少攻击面，而协作工作流程可提高工作效率。

使用 AWS Secrets Manager

Terraform 中有许多资源和数据源将秘密值以纯文本形式存储在状态文件中。避免将机密存储在状态下——改用AWS Secrets Manager。

与其尝试手动加密敏感值，不如依靠 Terraform 对敏感状态管理的内置支持。将敏感值导出到输出时，请确保将这些值标记为敏感。

持续扫描基础设施和源代码

持续主动扫描基础架构和源代码，以发现诸如泄露凭据或配置错误之类的风险，以加强您的安全状况。通过重新配置或修补资源来迅速解决发现的问题。

使用 AWS 服务进行动态扫描

使用 Amazon Inspector AWS Security Hub、A mazon Detectiv e 和 Amazon GuardDuty 等 AWS 原生工具跨账户和地区监控预配置的基础设施。在 Security Hub 中安排定期扫描，以跟踪部署和配置偏差。扫描 EC2 实例、Lambda 函数、容器、S3 存储桶和其他资源。

执行静态分析

将诸如 Checkov 之类的静态分析器直接嵌入到 CI/CD 管道中，以扫描 Terraform 配置代码 (HCL)，并在部署之前先发制人地识别风险。这样可以将安全检查移到开发过程的较早阶段（称为向左移动），并防止基础设施配置错误。

确保及时补救

对于所有扫描结果，请根据需要更新 Terraform 配置、应用补丁或手动重新配置资源，确保及时进行修复。通过解决根本原因来降低风险水平。

同时使用基础架构扫描和代码扫描可提供对 Terraform 配置、已配置资源和应用程序代码的分层见解。这通过预防、侦查和被动控制最大限度地提高了风险和合规性的覆盖范围，同时将安全性嵌入到软件开发生命周期 (SDLC) 的早期阶段。

强制执行政策检查

使用诸如 HashiCorp Sentinel 策略之类的代码框架为使用 Terraform 进行基础设施配置提供治理护栏和标准化模板。

Sentinel 策略可以定义对 Terraform 配置的要求或限制，以符合组织标准和最佳实践。例如，您可以使用 Sentinel 策略来：

- 所有资源都需要标签。
- 将实例类型限制为允许列表。
- 强制执行强制变量。
- 防止破坏生产资源。

将策略检查嵌入到 Terraform 配置生命周期中，可以主动执行标准和架构指南。Sentinel 提供共享的策略逻辑，有助于加快开发速度，同时防止未经批准的做法。

后端最佳实践

使用适当的远程后端存储状态文件对于实现协作、通过锁定确保状态文件的完整性、提供可靠的备份和恢复、与 CI/CD 工作流程集成以及利用 HCP Terraform 等托管服务提供的高级安全、治理和管理功能至关重要。

Terraform 支持各种后端类型，例如 Kubernetes、 HashiCorp Consul 和 HTTP。但是，本指南重点介绍了 Amazon S3，它是大多数 AWS 用户的最佳后端解决方案。

作为一项具有高持久性和可用性的完全托管的对象存储服务，Amazon S3 为管理 Terraform 状态提供了安全、可扩展且低成本的后端。 AWS Amazon S3 的全球足迹和弹性超过了大多数团队通过自我管理状态存储所能达到的水平。此外，Amazon S3 与 AWS 访问控制、加密选项、版本控制功能和其他服务原生集成使得 Amazon S3 成为便捷的后端选择。

本指南不为其他解决方案（例如 Kubernetes 或 Consul）提供后端指导，因为主要目标受众是客户。AWS 对于完全加入的团队来说，Amazon S3 通常是理想的选择 AWS Cloud，而不是 Kubernetes 或 HashiCorp Consul 集群。Amazon S3 状态存储的简单性、弹性和紧密 AWS 集成性为大多数遵循最佳实践的用户提供了 AWS 最佳基础。团队可以利用 AWS 服务的耐久性、备份保护和可用性来保持远程 Terraform 状态的高弹性。

遵循本节中的后端建议将提高Terraform代码库的协作性，同时限制错误或未经授权的修改的影响。通过实施架构良好的远程后端，团队可以优化 Terraform 工作流程。

最佳实践：

- 使用 Amazon S3 进行远程存储
- 促进团队协作
- 将每个环境的后端分开
- 主动监控远程状态活动

使用 Amazon S3 进行远程存储

与本地文件存储相比，在 Amazon S3 中远程存储 Terraform 状态以及使用 Amazon DynamoDB 实现状态锁定和一致性检查具有重大优势。远程状态支持团队协作、变更跟踪、备份保护和远程锁定，从而提高安全性。

使用具有 S3 标准存储类别（默认）的 Amazon S3，而不是临时的本地存储或自我管理解决方案，可提供 99.999999999% 的持久性和 99.99% 的可用性保护，以防止状态数据意外丢失。 AWS 诸如

Amazon S3 和 DynamoDB 之类的托管服务提供的服务级别协议 (SLA) 超过了大多数组织在自行管理存储时所能达到的水平。依靠这些保护来保持远程后端的可访问性。

启用远程状态锁定

DynamoDB 锁定会限制状态访问以防止并发写入操作。这样可以防止多个用户同时进行修改并减少错误。

带有状态锁定的后端配置示例：

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

启用版本控制和自动备份

为了获得更多保护，请在 Amazon S3 后端 AWS Backup 上使用启用自动版本控制和备份。每当进行更改时，版本控制都会保留状态的所有先前版本。它还允许您在需要时恢复以前的工作状态快照，以回滚不需要的更改或从意外中恢复。

如果需要，还原以前的版本

受版本控制的 Amazon S3 状态存储桶可通过恢复先前已知的良好状态快照来轻松恢复更改。这有助于防止意外更改，并提供额外的备份功能。

使用 HCP Terraform

HCP Terraform 为配置自己的状态存储提供了一种完全托管的后端替代方案。HCP Terraform 可自动处理状态和加密的安全存储，同时解锁其他功能。

当您使用 HCP Terraform 时，默认情况下状态是远程存储的，这样就可以在整个组织中共享和锁定状态。详细的策略控制可帮助您限制州访问权限和更改。

其他功能包括版本控制集成、策略护栏、工作流程自动化、变量管理以及与 SAML 的单点登录集成。您也可以使用 Sentinel 策略作为代码来实施治理控制。

尽管HCP Terraform需要使用软件即服务 (SaaS) 平台，但对于许多团队来说，安全、访问控制、自动策略检查和协作功能方面的优势使其成为使用Amazon S3或DynamoDB进行自我管理状态存储的最佳选择。

与诸如 GitHub 次要配置之类 GitLab 的服务轻松集成也吸引了那些完全采用云和SaaS工具以改善团队工作流程的用户。

促进团队协作

使用远程后端在 Terraform 团队的所有成员之间共享状态数据。这便于协作，因为它使整个团队都能了解基础架构的变化。共享后端协议与状态历史透明度相结合，简化了内部变更管理。所有基础架构变更都要经过既定管道，从而提高了整个企业的业务灵活性。

通过使用来改善问责制 AWS CloudTrail

与 AWS CloudTrail Amazon S3 存储桶集成，以捕获对状态存储桶进行的 API 调用。筛选要跟踪CloudTrail 的事件`PutObjectDeleteObject`,以及其他相关呼叫。

CloudTrail 日志显示了每次 API 调用以更改状态的委托人的 AWS 身份。用户的身份可以与计算机帐户或与后端存储进行交互的团队成员进行匹配。

将 CloudTrail 日志与 Amazon S3 状态版本控制相结合，将基础设施变更与应用这些变更的委托人联系起来。通过分析多个版本，您可以将任何更新归因于计算机帐户或负责的团队成员。

如果发生意外或破坏性更改，状态版本控制将提供回滚功能。 CloudTrail 将更改追踪到用户，以便您可以讨论预防性改进。

我们还建议您强制执行 IAM 权限以限制状态存储桶的访问权限。总体而言，S3 版本控制和 CloudTrail 监控支持对基础设施变更进行审计。团队在 Terraform 状态历史中提高了问责制、透明度和审计能力。

将每个环境的后端分开

为每个应用程序环境使用不同的 Terraform 后端。单独的后端将开发、测试和生产之间的状态隔离开来。

缩小影响范围

隔离状态有助于确保较低环境中的变化不会影响生产基础架构。开发和测试环境中的事故或实验影响有限。

限制生产访问权限

将生产状态后端的权限锁定为大多数用户的只读访问权限。将谁可以修改生产基础架构限制为 CI/CD 管道和破碎玻璃角色。

简化访问控制

在后端级别管理权限可简化环境之间的访问控制。为每个应用程序和环境使用不同的 S3 存储桶意味着可以对整个后端存储桶授予广泛的读取或写入权限。

避免共享工作空间

尽管你可以使用 Terraform 工作区来区分环境之间的状态，但不同的后端可以提供更强的隔离。如果您共享工作空间，则事故仍可能影响多个环境。

保持环境后端完全隔离可以最大限度地减少任何单一故障或漏洞的影响。单独的后端还可以使访问控制与环境的敏感度级别保持一致。例如，您可以为生产环境提供写入保护，为开发和测试环境提供更广泛的访问权限。

主动监控远程状态活动

持续监控远程状态活动对于及早发现潜在问题至关重要。查找异常解锁、更改或访问尝试。

获取有关可疑解锁的警报

大多数状态更改都应通过 CI/CD 管道进行。如果状态解锁直接通过开发者工作站发生，则生成警报，这可能表示未经授权或未经测试的更改。

监控访问尝试

状态存储桶上的身份验证失败可能表示有侦测活动。请注意，如果有多个账户正在尝试访问状态，或者出现异常的 IP 地址，这表明凭据已被泄露。

代码库结构和组织的最佳实践

随着大型团队和企业使用Terraform的增长，适当的代码库结构和组织至关重要。架构良好的代码库可实现大规模协作，同时增强可维护性。

本节提供有关支持质量和一致性的 Terraform 模块化、命名约定、文档和编码标准的建议。

指导包括按环境和组件将配置分解为可重复使用的模块，通过使用前缀和后缀建立命名约定，记录模块并清楚地解释输入和输出，以及使用自动样式检查来应用一致的格式化规则。

其他最佳实践包括在结构化层次结构中以逻辑方式组织模块和资源，在文档中对公共和私有模块进行分类，以及在模块中抽象不必要的实现细节以简化使用。

通过围绕模块化、文档、标准和逻辑组织实施代码库结构指南，您可以支持跨团队的广泛协作，同时随着使用情况在整个组织中的普及，Terraform 保持可维护性。通过强制执行惯例和标准，可以避免零碎代码库的复杂性。

最佳实践：

- 实现标准存储库结构
- 模块化结构
- 遵循命名惯例
- 使用附件资源
- 使用默认标签
- 满足 Terraform 注册表要求
- 使用推荐的模块源
- 遵循编码标准

实现标准存储库结构

我们建议您实现以下存储库布局。跨模块对这些一致性实践进行标准化可以提高可发现性、透明度、组织性和可靠性，同时支持在许多 Terraform 配置中重复使用。

- 根模块或目录：这应该是 Terraform 根模块和可重用模块的主要入口点，并且应该是唯一的。如果您的架构更复杂，则可以使用嵌套模块来创建轻量级抽象。这可以帮助您用架构来描述基础架构，而不是直接用物理对象来描述基础架构。

- 自述文件：根模块和任何嵌套模块都应有自述文件。此文件必须命名README.md。它应包含对模块的描述以及该模块的用途。如果要包括将此模块与其他资源一起使用的示例，请将其放在examples目录中。可以考虑添加一个图表，描述该模块可能创建的基础架构资源及其关系。使用 terraform-docs 自动生成模块的输入或输出。
- main.tf：这是主要的入口点。对于一个简单的模块，所有资源都可能在此文件中创建。对于复杂模块，资源创建可能分散在多个文件中，但任何嵌套的模块调用都应在main.tf文件中。
- v@@ ariables.tf 和 outputs.tf：这些文件包含变量和输出的声明。所有变量和输出都应有一句话或两句话来说明其用途。这些描述用于文档。有关更多信息，请参阅变量配置和输出配置 HashiCorp 文档。
  - 所有变量都必须具有已定义的类型。
  - 变量声明也可以包含默认参数。如果声明包含默认参数，则该变量被视为可选变量，如果您在调用模块或运行 Terraform 时未设置值，则使用默认值。默认参数需要文字值，并且不能引用配置中的其他对象。要使变量成为必填变量，请在变量声明中省略默认值，并考虑设置是否有nullable = false意义。
  - 对于具有与环境无关的值的变量（例如disk_size），请提供默认值。
  - 对于具有环境特定值的变量（例如project_id），请不要提供默认值。在这种情况下，调用模块必须提供有意义的值。
  - 仅当将变量留空是底层 API 不会拒绝的有效首选项时，才对空字符串或列表等变量使用空默认值。
  - 谨慎使用变量。仅当每个实例或环境的值必须有所不同时，才对其进行参数化。在决定是否公开变量时，请确保您有更改该变量的具体用例。如果需要变量的可能性很小，请不要公开它。
    - 添加具有默认值的变量是向后兼容的。
    - 移除变量不向后兼容。
    - 如果在多个地方重复使用文字，则应使用局部值而不将其作为变量公开。
  - 不要直接通过输入变量传递输出，因为这样做会阻止它们正确添加到依赖关系图中。为确保创建隐式依赖关系，请确保输出引用资源中的属性。与其直接引用实例的输入变量，不如传递该属性。
- locals.tf：此文件包含为表达式指定名称的本地值，因此可以在模块中多次使用一个名称，而不必重复该表达式。局部值就像函数的临时局部变量。局部值中的表达式不限于字面常量；它们还可以引用模块中的其他值，包括变量、资源属性或其他局部值，以便将它们组合起来。
- p@@ roviders.tf：此文件包含 terraform 块和提供程序块。provider模块的使用者只能在根模块中声明。

  如果你使用的是 HCP Terraform，还要添加一个空的云块。作为 CI/CD 管道的一部分，该cloud区块应完全通过环境变量和环境变量凭据进行配置。

- versions.tf：此文件包含 req uired_providers 块。所有 Terraform 模块都必须声明它需要哪些提供程序，这样 Terraform 才能安装和使用这些提供程序。
- data.tf：要进行简单配置，请将数据源放在引用它们的资源旁边。例如，如果您要获取用于启动实例的图像，请将其放在实例旁边，而不是在它们自己的文件中收集数据资源。如果数据源数量过大，可以考虑将其移至专用`data.tf`文件。
- .tfvars 文件：对于根模块，您可以使用文件提供非敏感变量。`.tfvars`为了保持一致性，请命名变量文件`terraform.tfvars`。将常用值放在存储库的根目录下，将特定于环境的值放在文件夹中。`envs/`
- 嵌套模块：`modules/`子目录下应存在嵌套模块。任何具有的嵌套模块`README.md`都被外部用户视为可用。如果 a `README.md` 不存在，则该模块仅供内部使用。应使用嵌套模块将复杂的行为拆分为多个小模块，用户可以仔细选择这些模块。

  如果根模块包括对嵌套模块的调用，则这些调用应使用相对路径，例如，`./modules/sample-module`这样 Terraform 就会将它们视为同一个存储库或包的一部分，而不是单独下载它们。

  如果存储库或包包含多个嵌套模块，则理想情况下，它们应可由调用者组合，而不是直接相互调用并创建深度嵌套的模块树。
- 示例：使用可重用模块的示例应存在于存储库根`examples/`目录下的子目录下。对于每个示例，您可以添加自述文件来解释示例的目标和用法。子模块的示例也应放在根`examples/`目录中。

  由于示例通常被复制到其他存储库中进行自定义，因此模块块的源应设置为外部调用者使用的地址，而不是相对路径。
- 服务命名文件：用户通常希望在多个文件中按服务分隔 Terraform 资源。应尽量不鼓励这种做法，`main.tf`而应在其中界定资源。但是，如果资源集合（例如，IAM 角色和策略）超过 150 行，则可以合理地将其分解为自己的文件，例如`iam.tf`。否则，所有资源代码都应在中定义`main.tf`。
- 自定义脚本：仅在必要时使用脚本。Terraform 不考虑或管理通过脚本创建的资源的状态。仅当 Terraform 资源不支持所需行为时才使用自定义脚本。将 Terraform 调用的自定义脚本放在目录中。`scripts/`
- 帮助脚本：将 Terraform 未调用的帮助脚本整理到目录中。`helpers/`在文件中记录助手脚本，`README.md`并附有说明和示例调用。如果帮助脚本接受参数，则提供参数检查和`--help`输出。
- 静态文件：Terraform 引用但未运行的静态文件（例如加载到 EC2 实例上的启动脚本）必须组织到一个`files/`目录中。将冗长的文档放在外部文件中，与其 HCL 分开。使用 file () 函数引用它们。
- 模板：对于 Terraform 模板文件函数读入的文件，请使用文件扩展名。`.tftpl`模板必须放在`templates/`目录中。

根模块结构

Terraform 总是在单个根模块的上下文中运行。完整的 Terraform 配置由根模块和子模块树（包括根模块调用的模块、这些模块调用的任何模块等）组成。

Terraform 根模块布局基本示例：

```
.
### 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
```

可重复使用的模块结构

可重复使用的模块遵循与根模块相同的概念。要定义模块，请为其创建一个新目录并将`.tf`文件放入其中，就像定义根模块一样。Terraform 可以从本地相对路径或远程存储库加载模块。如果您希望某个模块可以被许多配置重复使用，请将其放在自己的版本控制存储库中。保持模块树相对平坦很重要，这样可以更轻松地以不同的组合重复使用模块。

Terraform 可重复使用的模块布局基本示例：

```
.
### 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
```

模块化结构

原则上，您可以将任何资源和其他结构组合到一个模块中，但是过度使用嵌套和可重用的模块会使您的整体 Terraform 配置更难理解和维护，因此请谨慎使用这些模块。

如果有意义，可以将您的配置分解为可重复使用的模块，这些模块通过描述架构中由资源类型构造的新概念来提高抽象级别。

当您将基础设施模块化为可重复使用的定义时，应瞄准逻辑资源集，而不是单个组件或过于复杂的集合。

不要封装单个资源

你不应该创建围绕其他单一资源类型的薄包装模块。如果你在为模块找到与其中的主资源类型名称不同的名称时遇到困难，那么你的模块可能没有创建新的抽象，而是增加了不必要的复杂性。相反，直接在调用模块中使用资源类型。

封装逻辑关系

对相关资源进行分组，例如网络基础、数据层、安全控制和应用程序。可重复使用的模块应封装协同工作的基础架构部分，以实现一项功能。

保持继承不变

将模块嵌套在子目录中时，请避免深度超过一两个级别。深度嵌套的继承结构使配置和故障排除变得复杂。模块应该建立在其他模块之上，而不是通过它们构建隧道。

通过将模块重点放在代表架构模式的逻辑资源分组上，团队可以快速配置可靠的基础架构。在不进行过度设计或过度简化的情况下平衡抽象。

产出中的参考资源

对于在可重用模块中定义的每个资源，至少包括一个引用该资源的输出。变量和输出允许您推断模块和资源之间的依赖关系。如果没有任何输出，用户就无法根据他们的 Terraform 配置正确订购您的模块。

结构良好的模块可提供环境一致性、以目的为导向的分组和导出的资源引用，可实现组织范围内的 Terraform 大规模协作。团队可以利用可重复使用的构造块组装基础架构。

不要配置提供商

尽管共享模块继承调用模块的提供程序，但模块不应自行配置提供程序设置。避免在模块中指定提供程序配置块。此配置只能在全局声明一次。

申报所需的提供商

尽管提供程序配置在模块之间共享，但共享模块还必须声明自己的提供者要求。这种做法使 Terraform 能够确保提供程序的单一版本与配置中的所有模块兼容，并指定用作提供程序全局（与模块无关）标识符的源地址。但是，模块特定的提供程序要求并未指定任何用于确定提供程序将访问哪些远程端点的配置设置，例如。 AWS 区域

通过声明版本要求并避免使用硬编码的提供程序配置，模块使用共享提供程序提供跨Terraform配置的可移植性和可重用性。

对于共享模块，请在中的 requi red_providers 块中定义所需的最低提供者版本。`versions.tf`

要声明模块需要特定版本的 AWS 提供程序，请在`required_providers`块内使用一个`terraform`块：

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

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

如果共享模块仅支持 AWS 提供程序的特定版本，请使用悲观约束运算符 (~>)，它只允许最右边的版本组件递增：

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

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

在本示例中，`~> 4.0`允许安装`4.57.1`和，`4.67.0`但不允许`5.0.0`。有关更多信息，请参阅 HashiCorp 文档中的版本约束语法。

遵循命名惯例

清晰的描述性名称可简化您对模块中资源之间关系和配置值用途的理解。与风格指南的一致性提高了模块用户和维护者的可读性。

遵循资源命名准则

- 对所有资源名称使用 s nake_cas e（其中小写术语用下划线分隔），以匹配 Terraform 样式标准。这种做法可确保与资源类型、数据源类型和其他预定义值的命名约定保持一致。此约定不适用于名称参数。
- 为了简化对资源类型的唯一资源（例如，整个模块的单个负载均衡器）的引用，`this`为清楚起见，请将该资源命名为 o `main` r。

- 使用有意义的名称来描述资源的用途和上下文，并有助于区分相似的资源（例如，primary用于主数据库和read_replica数据库的只读副本）。
- 使用单数名称，而不是复数名称。
- 不要在资源名称中重复资源类型。

请遵循变量命名指南

- 在输入、局部变量和输出名称中添加表示磁盘大小或 RAM 大小等数值的单位（例如，以千兆字节ram_size_gb为单位的 RAM 大小）。这种做法使配置维护者可以清楚地了解预期的输入单元。
- 使用二进制单位（例如 MiB 和 GiB）作为存储大小，使用十进制单位（例如 MB 或 GB）来表示其他指标。
- 给布尔变量起正名，例如enable_external_access。

使用附件资源

有些资源中嵌入了伪资源作为属性。在可能的情况下，应避免使用这些嵌入的资源属性，而是使用唯一的资源来附加该伪资源。这些资源关系可能导致每种资源都存在独特 cause-and-effect 的问题。

使用嵌入式属性（避免这种模式）：

```
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"]
  }
```

```
}
```

使用附件资源（首选）：

```
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
}
```

使用默认标签

为所有可以接受标签的资源分配标签。Terraform P AWS rovider 有一个 aws_default_tags 数据源，你应该在根模块中使用它。

考虑向 Terraform 模块创建的所有资源添加必要的标签。以下是可能要附加的标签列表：

- 名称：人类可读的资源名称
- AppId：使用该资源的应用程序的 ID
- AppRole: 资源的技术功能；例如 “网络服务器” 或 “数据库”
- AppPurpose: 资源的业务目的；例如，“前端用户界面” 或 “支付处理器”
- 环境：软件环境，例如开发、测试或生产
- 项目：使用资源的项目
- CostCenter: 向谁收取资源使用费账单

满足 Terraform 注册表要求

模块存储库必须满足以下所有要求才能发布到 Terraform 注册表。

即使您不打算在短期内将模块发布到注册表，也应始终遵循这些要求。这样，您就可以稍后将模块发布到注册表，而不必更改存储库的配置和结构。

- 存储库名称：对于模块存储库，请使用由三部分组成的名称`terraform-aws-<NAME>`，其中`<NAME>`反映了模块管理的基础架构的类型。该`<NAME>`段可以包含其他连字符（例如，`terraform-aws-iam-terraform-roles`）。
- 标准模块结构：模块必须符合标准存储库结构。这允许注册表检查您的模块并生成文档、跟踪资源使用情况等。
  - 创建 Git 仓库后，将模块文件复制到仓库的根目录。我们建议您将每个打算重复使用的模块放在各自存储库的根目录中，但也可以从子目录中引用模块。
  - 如果您使用的是 HCP Terraform，请发布要共享到您的组织注册表的模块。注册表使用HCP Terraform API令牌处理下载并控制访问权限，因此，即使消费者从命令行运行Terraform，也无需访问模块的源存储库。
- 位置和权限：存储库必须位于您配置的版本控制系统 (VCS) 提供程序中，并且 HCP Terraform VCS 用户帐户必须具有存储库的管理员访问权限。注册表需要管理员访问权限才能创建 webhook 以导入新的模块版本。
- 发行版的 x.y.z 标签：要发布模块，必须至少有一个发布标签。注册表使用发布标签来标识模块版本。发布标签名称必须使用语义版本控制，您可以选择在语义版本控制前面加上`v`（例如`v1.1.0`和`1.1.0`）。注册表会忽略看起来不像版本号的标签。有关发布模块的更多信息，请参阅 Terraform 文档。

有关更多信息，请参阅 Terraform 文档中的准备模块存储库。

使用推荐的模块源

Terraform 使用模块块中的`source`参数来查找和下载子模块的源代码。

我们建议您对密切相关的模块使用本地路径，这些模块的主要目的是分解重复的代码元素，对于打算由多个配置共享的模块，使用原生 Terraform 模块注册表或 VCS 提供程序。

以下示例说明了共享模块的最常见和最推荐的源类型。注册表模块支持版本控制。您应始终提供特定的版本，如以下示例所示。

注册表

Terraform 注册表：

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

  ...

}
```

通过固定提交哈希，您可以避免偏离容易受到供应链攻击的公共注册表。

HCP Terraform：

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

  ...

  enable_karpenter = true
}
```

Terraform Enterpr

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

  ...

  enable_karpenter = true
}
```

VCS 提供商

版本控制系统（VCS）提供程序支持选择特定修订版的`ref`论点，如以下示例所示。

GitHub (HTTPS)：

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

```
  ...

  enable_karpenter = true
}
```

通用 Git 存储库 (HTTPS)：

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

  ...

  enable_karpenter = true
}
```

通用 Git 存储库 (SSH)：

> **Warning**
>
> 您需要配置凭据才能访问私有仓库。

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

  ...

  enable_karpenter = true
}
```

遵循编码标准

在所有配置文件中应用一致的 Terraform 格式规则和样式。通过在 CI/CD 管道中使用自动样式检查来强制执行标准。当您将编码最佳实践嵌入到团队工作流程中时，随着使用情况在整个组织中的广泛分布，配置仍保持可读性、可维护性和协作性。

遵循风格指南

- 使用 terraform fm t 命令格式化所有 Terraform .tf 文件（文件）以匹配样式标准。 HashiCorp

- 使用 terraform validat e 命令来验证配置的语法和结构。
- 使用 t fLint 静态分析代码质量。这个 linter 会检查 Terraform 的最佳实践，而不仅仅是格式化，并且在遇到错误时会失败构建。

配置预提交挂钩

在允许提交之前，配置运行`terraform fmt`、`tflint`和`checkov`、和其他代码扫描和样式检查的客户端预提交挂钩。这种做法可以帮助您在开发人员工作流程中尽早验证标准一致性。

使用预提交框架（例如预提交）在本地计算机上将 Terraform linting、格式化和代码扫描作为挂钩添加。Hooks 会在每个 Git 提交上运行，如果检查未通过，则提交失败。

将样式和质量检查移至本地预提交挂钩可在引入更改之前向开发人员提供快速反馈。标准成为编码工作流程的一部分。

AWS 提供商版本管理的最佳实践

仔细管理 AWS 提供程序和关联的 Terraform 模块的版本对于稳定性至关重要。本节概述了有关版本限制和升级的最佳实践。

最佳实践：

- 添加自动版本检查
- 监控新版本
- 向提供者捐款

添加自动版本检查

在 CI/CD 管道中为 Terraform 提供者添加版本检查以验证版本固定，如果版本未定义，则生成失败。

- 在 CI/CD 管道中添加 tfLint 检查，以扫描未定义固定主版本/次要版本限制的提供程序版本。使用适用于 Terraform Prov AWS ider 的 tfLint 规则集插件，该插件提供了检测可能的错误的规则并检查有关资源的最佳实践。 AWS
- Fail CI 运行可检测未固定的提供程序版本，以防止隐式升级进入生产环境。

监控新版本

- 监控提供商的发行说明和变更日志提要。获取有关新的主要/次要版本的通知。
- 评估发行说明中是否存在潜在的重大更改，并评估其对现有基础架构的影响。
- 在更新生产环境之前，请先升级非生产环境中的次要版本以对其进行验证。

通过在管道中自动检查版本并监控新版本，您可以尽早发现不支持的升级，让您的团队有时间在更新生产环境之前评估新的主要/次要版本的影响。

向提供者捐款

通过报告缺陷或请求 GitHub 问题中的功能来积极为 HashiCorp AWS 提供商做出贡献：

- 在 Prov AWS ider 存储库中打开有据可查的问题，详细说明您遇到的任何错误或缺少的功能。提供可重现的步骤。

- 请求并就增强功能进行投票，以扩展 AWS 提供商管理新服务的能力。
- 当您为提供者缺陷或增强功能提供建议的修复程序时，请引用已发出的拉取请求。相关问题链接。
- 请遵循存储库中的贡献指南，了解编码惯例、测试标准和文档。

通过回馈您使用的提供商，您可以直接为他们的路线图提供意见，并帮助他们提高所有用户的质量和功能。

社区模块的最佳实践

有效使用模块是管理复杂的 Terraform 配置和促进重复使用的关键。本节提供了有关社区模块、依赖关系、来源、抽象和贡献的最佳实践。

最佳实践：

- 探索社区模块
- 了解依赖关系
- 使用可信来源
- 为社区模块做出贡献

探索社区模块

在构建新模块之前 GitHub，请在 Terraform Re gistry 和其他来源中搜索可能解决您的用例的现有 AWS 模块。寻找具有最新更新且正在积极维护的热门选项。

使用变量进行自定义

使用社区模块时，请通过变量传递输入，而不是分叉或直接修改源代码。在需要时覆盖默认值，而不是更改模块的内部结构。

Forking 应仅限于为原始模块提供修复或功能，以使更广泛的社区受益。

了解依赖关系

在使用该模块之前，请查看其源代码和文档以确定依赖关系：

- 必需的提供程序：请注意模块所需的版本 AWS、Kubernetes 或其他提供程序。
- 嵌套模块：检查内部使用的其他引入级联依赖关系的模块。
- 外部数据源：记下模块所依赖的 API、自定义插件或基础架构依赖关系。

通过绘制直接和间接依赖关系的完整树，可以在使用该模块时避免意外。

使用可信来源

从未经验证或未知的发行商那里采购 Terraform 模块会带来重大风险。仅使用来自可信来源的模块。

- 优先选择来自 Terraform Re gistry 的认证模块，这些模块由经过验证的创作者（例如 AWS 或 HashiCorp 合作伙伴）发布。
- 对于自定义模块，请查看发布者历史记录、支持级别和使用声誉，即使该模块来自您自己的组织。

通过不允许使用来自未知或未经审查的来源的模块，可以降低在代码中注入漏洞或维护问题的风险。

订阅 通知

订阅来自值得信赖的发行商发布的新模块的通知：

- 查看 GitHub 模块存储库以获取有关该模块新版本的警报。
- 监控发布商博客和变更日志以获取更新。
- 从经过验证的、高度评价的来源获取有关新版本的主动通知，而不是隐含地获取更新。

仅使用来自可信来源的模块并监控更改可提供稳定性和安全性。经过审查的模块可提高生产力，同时最大限度地降低供应链风险。

为社区模块做出贡献

提交托管在以下位置的社区模块的修复和增强功能 GitHub：

- 在模块上打开拉取请求，以解决您在使用中遇到的缺陷或限制。
- 通过创建问题，请求将新的最佳实践配置添加到现有 OSS 模块中。

为社区模块做出贡献可以增强所有 Terraform 从业者的可重复使用的编纂模式。

常见问题解答

问：为什么要关注 AWS 提供商？

答： AWS 提供者是Terraform中用于配置基础设施的最广泛和最复杂的提供商之一。遵循这些最佳实践可以帮助用户优化他们对 AWS 环境提供商的使用。

问：我是 Terraform 的新手。我可以使用这份指南吗？

答：该指南适用于刚接触Terraform的人以及想要提高技能的更高级的从业者。这些实践可以改善用户在任何学习阶段的工作流程。

问：涵盖了哪些关键的最佳实践？

答：关键的最佳实践包括在访问密钥上使用 IAM 角色、固定版本、整合自动测试、远程状态锁定、凭证轮换、向提供商贡献以及逻辑组织代码库。

问：在哪里可以了解有关 Terraform 的更多信息？

答：“资源” 部分包含指向 HashiCorp Terraform 官方文档和社区论坛的链接。使用链接了解有关高级 Terraform 工作流程的更多信息。

后续步骤

阅读本指南后，以下是一些可能的后续步骤：

- 如果您已有 Terraform 代码库，请检查您的配置，并根据本指南中提供的建议确定可以改进的方面。例如，查看实现远程后端、将代码分成模块、使用版本固定等的最佳实践，然后在配置中对其进行验证。
- 如果您没有现有 Terraform 代码库，请在构建新配置时使用这些最佳实践。从一开始就遵循有关状态管理、身份验证、代码结构等的建议。
- 尝试使用本指南中提及的一些 HashiCorp 社区模块，看看它们是否简化了您的架构模式。这些模块允许更高级别的抽象，因此您不必重写公共资源。
- 启用 linting、安全扫描、策略检查和自动测试工具，以强化安全性、合规性和代码质量方面的一些最佳实践。诸如 tFlint、tfsec 和 Checkov 之类的工具可以提供帮助。
- 查看最新的 P AWS rovider 文档，看看是否有任何新的资源或功能可以帮助优化 Terraform 的使用。随时了解 AWS 提供商的新版本。
- 有关其他指导，请参阅网站上的 Terraform 文档、最佳实践指南和风格指南。 HashiCorp

资源

参考信息

以下链接为 Terraform 提供 AWS 商提供了其他阅读材料，并在 IaC 上使用 Terraform。 AWS

- Terraform AWS 提供者（文档）HashiCorp
- 用于 AWS 服务的 Terraform 模块（Terraform Registry）
- AWS 和 Partn HashiCorp er shi HashiCorp p（博客文章）
- AWS 提供商的动态凭证（HCP Terraform 文档）
- DynamoDB 状态锁定（Terraform 文档）
- 使用 Sentinel 强制执行政策（Terraform 文档）

工具

按照本最佳实践指南中的建议，以下工具有助于提高代码质量和Terraform配置的自动化程度。 AWS

代码质量：

- Checkov：在部署之前扫描 Terraform 代码以识别错误配置。
- tfLint：识别可能的错误、不推荐使用的语法和未使用的声明。这个 linter 还可以强制执行 AWS 最佳实践和命名约定。
- terraform-docs：从 Terraform 模块生成各种输出格式的文档。

自动化工具：

- HCP Terraform：通过政策检查和批准门帮助团队进行版本控制、协作和构建 Terraform 工作流程。
- 亚特兰蒂斯：一款用于验证代码更改的开源 Terraform 拉取请求自动化工具。
- 适用于 Terraform 的 CDK：一个框架，允许你使用熟悉的语言（例如 TypeScript Python、Java、C# 和 Go）来代替 HashiCorp 配置语言 (HCL) 来定义、配置和测试你的 Terraform 基础架构，即代码。

文档历史记录

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

变更	说明	日期
初次发布	—	2024 年 5 月 28 日

AWS 规范性指导词汇表

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

数字

7 R

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

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

A

ABAC

请参阅基于属性的访问控制。

抽象服务

参见托管服务。

酸

参见原子性、一致性、隔离性、耐久性。

主动-主动迁移

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

主动-被动迁移

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

聚合函数

一个 SQL 函数，它对一组行进行操作并计算该组的单个返回值。聚合函数的示例包括SUM和MAX。

AI

参见人工智能。

AIOps

参见人工智能操作。

匿名化

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

反模式

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

应用程序控制

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

应用程序组合

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

人工智能（AI）

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

人工智能运营（AIOps）

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

非对称加密

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

原子性、一致性、隔离性、持久性（ACID）

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

基于属性的访问权限控制（ABAC）

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

权威数据源

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

可用区

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

AWS 云采用框架 (AWS CAF)

该框架包含指导方针和最佳实践 AWS ，可帮助组织制定高效且有效的计划，以成功迁移到云端。AWS CAF将指导分为六个重点领域，称为视角：业务、人员、治理、平台、安全和运营。业务、人员和治理角度侧重于业务技能和流程；平台、安全和运营角度侧重于技术技能和流程。例如，人员角度针对的是负责人力资源（HR）、人员配置职能和人员管理的利益相关者。从这个角度来看，AWS CAF 为人员发展、培训和沟通提供了指导，以帮助组织为成功采用云做好准备。有关更多信息，请参阅 AWS CAF 网站和 AWS CAF 白皮书。

AWS 工作负载资格框架 (AWS WQF)

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

B

坏机器人

旨在破坏个人或组织或对其造成伤害的机器人。

BCP

参见业务连续性计划。

行为图

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

大端序系统

一个先存储最高有效字节的系统。另请参见字节顺序。

二进制分类

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

bloom 筛选条件

一种概率性、内存高效的数据结构，用于测试元素是否为集合的成员。

蓝/绿部署

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

自动程序

一种通过互联网运行自动任务并模拟人类活动或互动的软件应用程序。有些机器人是有用或有益的，例如在互联网上索引信息的网络爬虫。其他一些被称为恶意机器人的机器人旨在破坏个人或组织或对其造成伤害。

僵尸网络

被恶意软件感染并受单方（称为机器人牧民或机器人操作员）控制的机器人网络。僵尸网络是最著名的扩展机器人及其影响力的机制。

分支

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

破碎的玻璃通道

在特殊情况下，通过批准的流程，用户 AWS 账户 可以快速访问他们通常没有访问权限的内容。有关更多信息，请参阅 Well -Architected 指南中的 “实施破碎玻璃程序” 指示 AWS 器。

棕地策略

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

缓冲区缓存

存储最常访问的数据的内存区域。

业务能力

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

业务连续性计划（BCP）

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

C

CAF

参见AWS 云采用框架。

金丝雀部署

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

CCoE

参见云卓越中心。

CDC

参见变更数据捕获。

更改数据捕获（CDC）

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

混沌工程

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

CI/CD

查看持续集成和持续交付。

分类

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

客户端加密

在目标 AWS 服务 收到数据之前，对数据进行本地加密。

云卓越中心（CCoE）

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

云计算

通常用于远程数据存储和 IoT 设备管理的云技术。云计算通常与边缘计算技术相关。

云运营模型

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

云采用阶段

组织迁移到以下阶段时通常会经历四个阶段 AWS Cloud：

- 项目 - 出于概念验证和学习目的，开展一些与云相关的项目
- 基础 - 进行基础投资以扩大云采用率（例如，创建登录区、定义 CCoE、建立运营模型）

- 迁移 - 迁移单个应用程序
- 重塑 - 优化产品和服务，在云中创新

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

CMDB

参见配置管理数据库。

代码存储库

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

冷缓存

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

冷数据

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

计算机视觉 (CV)

人工智能领域，使用机器学习来分析和提取数字图像和视频等视觉格式中的信息。例如， AWS Panorama 提供将 CV 添加到本地摄像机网络的设备，而 Amazon 则为 CV SageMaker 提供图像处理算法。

配置偏差

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

配置管理数据库（CMDB）

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

合规性包

一系列 AWS Config 规则和补救措施，您可以汇编这些规则和补救措施，以自定义合规性和安全性检查。您可以使用 YAML 模板将一致性包作为单个实体部署在 AWS 账户 和区域或整个组织中。有关更多信息，请参阅 AWS Config 文档中的一致性包。

持续集成和持续交付（CI/CD）

自动执行软件发布过程的源代码、构建、测试、暂存和生产阶段的过程。CI/CD 通常被描述为管道。CI/CD 可以帮助您实现流程自动化、提高工作效率、改善代码质量并加快交付速度。有关更多信息，请参阅持续交付的优势。CD 也可以表示持续部署。有关更多信息，请参阅持续交付与持续部署。

CV

参见计算机视觉。

D

静态数据

网络中静止的数据，例如存储中的数据。

数据分类

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

数据漂移

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

传输中数据

在网络中主动移动的数据，例如在网络资源之间移动的数据。

数据网格

一种架构框架，可提供分布式、去中心化的数据所有权以及集中式管理和治理。

数据最少化

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

数据边界

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

数据预处理

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

数据溯源

在数据的整个生命周期跟踪其来源和历史的过程，例如数据如何生成、传输和存储。

数据主体

正在收集和处理其数据的个人。

数据仓库

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

数据库定义语言（DDL）

在数据库中创建或修改表和对象结构的语句或命令。

数据库操作语言（DML）

在数据库中修改（插入、更新和删除）信息的语句或命令。

DDL

参见数据库定义语言。

深度融合

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

深度学习

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

defense-in-depth

一种信息安全方法，经过深思熟虑，在整个计算机网络中分层实施一系列安全机制和控制措施，以保护网络及其中数据的机密性、完整性和可用性。当你采用这种策略时 AWS，你会在 AWS Organizations 结构的不同层面添加多个控件来帮助保护资源。例如，一种 defense-in-depth 方法可以结合多因素身份验证、网络分段和加密。

委托管理员

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

部署

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

开发环境

参见环境。

侦测性控制

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

开发价值流映射 (DVSM)

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

数字孪生

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

维度表

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

灾难

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

灾难恢复 (DR)

您用来最大限度地减少灾难造成的停机时间和数据丢失的策略和流程。有关更多信息，请参阅 Well-Architected Fr ame AWS work 中的 “工作负载灾难恢复：云端 AWS 恢复”。

DML

参见数据库操作语言。

领域驱动设计

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

DR

参见灾难恢复。

漂移检测

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

DVSM

参见开发价值流映射。

E

EDA

参见探索性数据分析。

边缘计算

该技术可提高位于 IoT 网络边缘的智能设备的计算能力。与云计算相比，边缘计算可以减少通信延迟并缩短响应时间。

加密

一种将人类可读的纯文本数据转换为密文的计算过程。

加密密钥

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

字节顺序

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

端点

参见服务端点。

端点服务

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

企业资源规划 (ERP)

一种自动化和管理企业关键业务流程（例如会计、MES 和项目管理）的系统。

信封加密

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

environment

正在运行的应用程序的实例。以下是云计算中常见的环境类型：

- 开发环境 — 正在运行的应用程序的实例，只有负责维护应用程序的核心团队才能使用。开发环境用于测试更改，然后再将其提升到上层环境。这类环境有时称为测试环境。
- 下层环境 — 应用程序的所有开发环境，比如用于初始构建和测试的环境。
- 生产环境 — 最终用户可以访问的正在运行的应用程序的实例。在 CI/CD 管道中，生产环境是最后一个部署环境。
- 上层环境 — 除核心开发团队以外的用户可以访问的所有环境。这可能包括生产环境、预生产环境和用户验收测试环境。

epic

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

ERP

参见企业资源规划。

探索性数据分析（EDA）

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

F

事实表

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

失败得很快

一种使用频繁和增量测试来缩短开发生命周期的理念。这是敏捷方法的关键部分。

故障隔离边界

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

功能分支

参见分支。

特征

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

特征重要性

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

功能转换

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

FGAC

请参阅精细的访问控制。

精细访问控制 (FGAC)

使用多个条件允许或拒绝访问请求。

快闪迁移

一种数据库迁移方法，它使用连续的数据复制，通过更改数据捕获在尽可能短的时间内迁移数据，而不是使用分阶段的方法。目标是将停机时间降至最低。

G

地理封锁

请参阅地理限制。

地理限制（地理阻止）

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

GitFlow 工作流程

一种方法，在这种方法中，下层和上层环境在源代码存储库中使用不同的分支。Gitflow 工作流程被认为是传统的，而基于主干的工作流程是现代的首选方法。

全新策略

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

防护机制

一种高级规则，用于跨组织单位（OU）管理资源、策略和合规性。预防性防护机制会执行策略以确保符合合规性标准。它们是使用服务控制策略和 IAM 权限边界实现的。侦测性防护机制会检测策略违规和合规性问题，并生成警报以进行修复。它们通过使用 AWS Config、、Amazon、 AWS Security Hub GuardDuty AWS Trusted Advisor、Amazon Inspector 和自定义 AWS Lambda 支票来实现。

H

HA

参见高可用性。

异构数据库迁移

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

高可用性 (HA)

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

历史数据库现代化

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

同构数据库迁移

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

热数据

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

修补程序

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

hypercare 周期

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

I

IaC

参见基础架构即代码。

基于身份的策略

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

空闲应用程序

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

IIoT

参见工业物联网。

不可变的基础架构

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

入站（入口）VPC

在 AWS 多账户架构中，一种接受、检查和路由来自应用程序外部的网络连接的 VPC。AWS 安全参考架构建议使用入站、出站和检查 VPC 设置网络账户，保护应用程序与广泛的互联网之间的双向接口。

增量迁移

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

工业 4.0

该术语由克劳斯·施瓦布（Klaus Schwab）于2016年推出，指的是通过连接、实时数据、自动化、分析和人工智能/机器学习的进步实现制造流程的现代化。

基础设施

应用程序环境中包含的所有资源和资产。

基础设施即代码（IaC）

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

工业物联网（IIoT）

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

检查 VPC

在 AWS 多账户架构中，一种集中式 VPC，用于管理 VPC（相同或不同 AWS 区域）、互联网和本地网络之间的网络流量检查。AWS 安全参考架构建议使用入站、出站和检查 VPC 设置网络账户，保护应用程序与广泛的互联网之间的双向接口。

物联网（IoT）

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

可解释性

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

IoT

参见物联网。

IT 信息库（ITIL）

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

IT 服务管理（ITSM）

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

ITIL

请参阅 IT 信息库。

ITSM

请参阅 IT 服务管理。

L

基于标签的访问控制 (LBAC)

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

登录区

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

大规模迁移

迁移 300 台或更多服务器。

LBAC

参见基于标签的访问控制。

最低权限

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

直接迁移

见 7 R。

小端序系统

一个先存储最低有效字节的系统。另请参见字节顺序。

下层环境

参见环境。

M

机器学习（ML）

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

主分支

参见分支。

恶意软件

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

托管服务

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

制造执行系统 (MES)

一种软件系统，用于跟踪、监控、记录和控制在车间将原材料转化为成品的生产过程。

MAP

参见迁移加速计划。

机制

一个完整的过程，在此过程中，您可以创建工具，推动工具的采用，然后检查结果以进行调整。机制是一种在运行过程中自我增强和改进的循环。有关更多信息，请参阅在 Well-Architect AWS ed 框架中构建机制。

成员账户

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

MES

参见制造执行系统。

消息队列遥测传输 (MQTT)

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

微服务

一种小型独立服务，通过明确定义的 API 进行通信，通常由小型独立团队拥有。例如，保险系统可能包括映射到业务能力（如销售或营销）或子域（如购买、理赔或分析）的微服务。微服务的好处包括敏捷、灵活扩展、易于部署、可重复使用的代码和恢复能力。有关更多信息，请参阅使用 AWS 无服务器服务集成微服务。

微服务架构

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

迁移加速计划（MAP）

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

大规模迁移

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

迁移工厂

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

迁移元数据

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

迁移模式

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

迁移组合评测（MPA）

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

迁移准备情况评测（MRA）

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

迁移策略

用于将工作负载迁移到的方法 AWS Cloud。有关更多信息，请参阅此词汇表中的 7 R 条目和动员组织以加快大规模迁移。

ML

参见机器学习。

现代化

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

现代化准备情况评估

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

单体应用程序（单体式）

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

MPA

参见迁移组合评估。

MQTT

请参阅消息队列遥测传输。

多分类器

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

可变基础架构

一种用于更新和修改现有生产工作负载基础架构的模型。为了提高一致性、可靠性和可预测性，Well-Architect AWS ed Framework 建议使用不可变基础设施作为最佳实践。

O

OAC

请参阅源站访问控制。

OAI

参见源访问身份。

OCM

参见组织变更管理。

离线迁移

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

OI

参见运营集成。

OLA

参见运营层协议。

在线迁移

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

OPC-UA

参见开放流程通信-统一架构。

开放流程通信-统一架构 (OPC-UA)

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

运营级别协议（OLA）

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

运营准备情况审查 (ORR)

一份问题清单和相关的最佳实践，可帮助您理解、评估、预防或缩小事件和可能的故障的范围。有关更多信息，请参阅 Well-Architecte AWS d Frame work 中的运营准备情况评估 (ORR)。

操作技术 (OT)

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

运营整合（OI）

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

组织跟踪

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

组织变革管理（OCM）

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

来源访问控制（OAC）

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

来源访问身份（OAI）

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

或者

参见运营准备情况审查。

OT

参见运营技术。

出站（出口）VPC

在 AWS 多账户架构中，一种处理从应用程序内部启动的网络连接的 VPC。AWS 安全参考架构建议使用入站、出站和检查 VPC 设置网络账户，保护应用程序与广泛的互联网之间的双向接口。

P

权限边界

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

个人身份信息 (PII)

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

PII

查看个人身份信息。

playbook

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

PLC

参见可编程逻辑控制器。

PLM

参见产品生命周期管理。

策略

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

多语言持久性

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

组合评测

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

谓词

返回`true`或的查询条件`false`，通常位于子`WHERE`句中。

谓词下推

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

预防性控制

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

主体

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

隐私设计

一种贯穿整个工程化过程考虑隐私的系统工程方法。

私有托管区

私有托管区就是一个容器，其中包含的信息说明您希望 Amazon Route 53 如何响应一个或多个 VPC 中的某个域及其子域的 DNS 查询。有关更多信息，请参阅 Route 53 文档中的私有托管区的使用。

主动控制

一种安全控制措施，旨在防止部署不合规的资源。这些控件会在资源置备之前对其进行扫描。如果资源与控件不兼容，则不会对其进行配置。有关更多信息，请参阅 AWS Control Tower 文档中的控制参考指南，并参见在上实施安全控制中的主动控制 AWS。

产品生命周期管理 (PLM)

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

生产环境

参见环境。

可编程逻辑控制器 (PLC)

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

假名化

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

发布/订阅（发布/订阅）

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

Q

查询计划

一系列步骤，例如指令，用于访问 SQL 关系数据库系统中的数据。

查询计划回归

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

R

RACI 矩阵

参见 “负责任、负责、咨询、知情”（RACI）。

勒索软件

一种恶意软件，旨在阻止对计算机系统或数据的访问，直到付款为止。

RASCI 矩阵

参见 “负责任、负责、咨询、知情”（RACI）。

RCAC

请参阅行和列访问控制。

只读副本

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

重新架构师

见 7 R。

恢复点目标 (RPO)

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

恢复时间目标 (RTO)

服务中断和服务恢复之间可接受的最大延迟。

重构

见 7 R。

区域

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

回归

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

重新托管

见 7 R。

版本

在部署过程中，推动生产环境变更的行为。

搬迁

见 7 R。

更换平台

见 7 R。

回购

见 7 R。

故障恢复能力

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

基于资源的策略

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

责任、问责、咨询和知情（RACI）矩阵

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

响应性控制

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

保留

见 7 R。

退休

见 7 R。

旋转

定期更新密钥以使攻击者更难访问凭据的过程。

行列访问控制 (RCAC)

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

RPO

参见恢复点目标。

RTO

参见恢复时间目标。

运行手册

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

S

SAML 2.0

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

SCADA

参见监督控制和数据采集。

SCP

参见服务控制政策。

secret

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

安全控制

一种技术或管理防护机制，可防止、检测或降低威胁行为体利用安全漏洞的能力。安全控制主要有四种类型：预防性、侦测、响应式和主动式。

安全加固

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

安全信息和事件管理（SIEM）系统

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

安全响应自动化

一种预定义和编程的操作，旨在自动响应或修复安全事件。这些自动化可作为侦探或响应式安全控制措施，帮助您实施 AWS 安全最佳实践。自动响应操作的示例包括修改 VPC 安全组、修补 Amazon EC2 实例或轮换证书。

服务器端加密

在目的地对数据进行加密，由接收数据 AWS 服务 的人加密。

服务控制策略（SCP）

一种策略，用于集中控制 AWS Organizations的组织中所有账户的权限。SCP 为管理员可以委托给用户或角色的操作定义了防护机制或设定了限制。您可以将 SCP 用作允许列表或拒绝列表，指定允许或禁止哪些服务或操作。有关更多信息，请参阅 AWS Organizations 文档中的服务控制策略。

服务端点

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

服务水平协议（SLA）

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

服务级别指示器 (SLI)

对服务性能方面的衡量，例如其错误率、可用性或吞吐量。

服务级别目标 (SLO)

代表服务运行状况的目标指标，由服务级别指标衡量。

责任共担模式

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

暹粒

参见安全信息和事件管理系统。

单点故障 (SPOF)

应用程序的单个关键组件出现故障，可能会中断系统。

SLA

参见服务级别协议。

SLI

参见服务级别指标。

SLO

参见服务级别目标。

split-and-seed 模型

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

恶作剧

参见单点故障。

星型架构

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

strangler fig 模式

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

子网

您的 VPC 内的一个 IP 地址范围。子网必须位于单个可用区中。

监控和数据采集 (SCADA)

在制造业中，一种使用硬件和软件来监控有形资产和生产操作的系统。

对称加密

一种加密算法，它使用相同的密钥来加密和解密数据。

综合测试

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

T

标签

键值对，充当用于组织资源的元数据。 AWS 标签可帮助您管理、识别、组织、搜索和筛选资源。有关更多信息，请参阅标记您的 AWS 资源。

目标变量

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

任务列表

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

测试环境

参见环境。

训练

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

中转网关

中转网关是网络中转中心，您可用它来互连 VPC 和本地网络。有关更多信息，请参阅 AWS Transit Gateway 文档中的什么是公交网关。

基于中继的工作流程

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

可信访问权限

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

优化

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

双披萨团队

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

U

不确定性

这一概念指的是不精确、不完整或未知的信息，这些信息可能会破坏预测式 ML 模型的可靠性。不确定性有两种类型：认知不确定性是由有限的、不完整的数据造成的，而偶然不确定性是由数据中固有的噪声和随机性导致的。有关更多信息，请参阅量化深度学习系统中的不确定性指南。

无差别任务

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

上层环境

参见环境。

V

vacuum 操作

一种数据库维护操作，包括在增量更新后进行清理，以回收存储空间并提高性能。

版本控制

跟踪更改的过程和工具，例如存储库中源代码的更改。

VPC 对等连接

两个 VPC 之间的连接，允许您使用私有 IP 地址路由流量。有关更多信息，请参阅 Amazon VPC 文档中的什么是 VPC 对等连接。

漏洞

损害系统安全的软件缺陷或硬件缺陷。

W

热缓存

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

暖数据

不常访问的数据。查询此类数据时，通常可以接受中速查询。

窗口函数

一个 SQL 函数，用于对一组以某种方式与当前记录相关的行进行计算。窗口函数对于处理任务很有用，例如计算移动平均线或根据当前行的相对位置访问行的值。

工作负载

一系列资源和代码，它们可以提供商业价值，如面向客户的应用程序或后端过程。

工作流

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

蠕虫

参见一次写入，多读。

WQF

请参阅 AWS 工作负载资格框架。

一次写入，多次读取 (WORM)

一种存储模型，它可以一次写入数据并防止数据被删除或修改。授权用户可以根据需要多次读取数据，但他们无法对其进行更改。这种数据存储基础架构被认为是不可变的。

Z

零日漏洞利用

一种利用未修补漏洞的攻击，通常是恶意软件。

零日漏洞

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

僵尸应用程序

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