Terraform AWS プロバイダーを使用するためのベストプラクティス

AWS 規範ガイダンス

AWS 規範ガイダンス: Terraform AWS プロバイダーを使用するためのベストプラクティス

Table of Contents

Terraform AWS プロバイダーを使用するためのベストプラクティス

Michael Begin、Amazon Web Services シニア DevOps コンサルタント (AWS)

2025 年 8 月 (ドキュメント履歴)

で Terraform を使用してコードとしてのインフラストラクチャ (IaC) を管理すると、一貫性、セキュリティ、俊敏性の向上などの重要な利点 AWS が得られます。ただし、Terraform 設定のサイズと複雑さが大きくなるにつれて、落とし穴を避けるためにベストプラクティスに従うことが重要です。

このガイドでは、HashiCorp の Terraform AWS プロバイダーを使用する際に推奨されるベストプラクティスについて説明します。Terraform を最適化するための適切なバージョニング、セキュリティコントロール、リモートバックエンド、コードベース構造、コミュニティプロバイダーについて説明します AWS。各セクションでは、以下のベストプラクティスの適用の詳細について説明します。

- セキュリティ
- バックエンド
- コードのベース構造と組織
- AWS プロバイダーのバージョン管理
- コミュニティモジュール

目的

このガイドは、Terraform AWS プロバイダーに関する運用上の知識を深め、セキュリティ、信頼性、コンプライアンス、開発者の生産性に関する IaC のベストプラクティスに従うことで達成できる以下のビジネス目標に対処するのに役立ちます。

- Terraform プロジェクト全体でインフラストラクチャコードの品質と一貫性を向上させます。
- 開発者のオンボーディングを加速し、インフラストラクチャコードに貢献できるようにします。
- インフラストラクチャの迅速な変更により、ビジネスの俊敏性を向上させます。
- インフラストラクチャの変更に関連するエラーとダウンタイムを削減します。
- IaC のベストプラクティスに従ってインフラストラクチャコストを最適化します。
- ベストプラクティスの実装を通じて全体的なセキュリティ体制を強化します。

ターゲットオーディエンス

このガイドの対象者には、Terraform for IaC を使用するチームを監督する技術リーダーとマネージャーが含まれます AWS。その他の潜在的な読者には、インフラストラクチャエンジニア、DevOps エンジニア、ソリューションアーキテクト、Terraform を積極的に使用して AWS インフラストラクチャを管理する開発者が含まれます。

これらのベストプラクティスに従うことで、時間を節約し、これらのロールに対する IaC の利点を引き出すことができます。

概要

Terraform プロバイダーは、Terraform がさまざまな APIs。Terraform AWS プロバイダーは、Terraform で AWS infrastructure as code (IaC) を管理するための公式プラグインです。Terraform 構文を AWS API コールに変換して、 AWS リソースを作成、読み取り、更新、削除します。

AWS プロバイダーは、認証、Terraform 構文の AWS API コールへの変換、および でのリソースのプロビジョニングを処理します AWS。Terraform `provider`コードブロックを使用して、Terraform が AWS API とのやり取りに使用するプロバイダープラグインを設定します。複数の AWS プロバイダーブロックを設定して、異なる AWS アカウント およびリージョンのリソースを管理できます。

以下は、エイリアスで複数の AWS プロバイダーブロックを使用して、別のリージョンとアカウントにレプリカを持つ Amazon Relational Database Service (Amazon RDS) データベースを管理する Terraform 設定の例です。プライマリプロバイダーとセカンダリプロバイダーは、異なる AWS Identity and Access Management (IAM) ロールを引き受けます。

```
# 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
}
```

この例では、以下のようになっています。

- 最初のproviderブロックは、 us-west-1 リージョンのプライマリ AWS プロバイダーをエイリアス で設定しますprimary。
- 2 番目のproviderブロックは、 us-east-1 リージョンのセカンダリ AWS プロバイダーをエイリアス で設定しますreplica。このプロバイダーは、別のリージョンとアカウントにプライマリデータベースのリードレプリカを作成するために使用されます。assume_role ブロックは、レプリカアカウントの IAM ロールを引き受けるために使用されます。は、引き受ける IAM ロールの Amazon リソースネーム (ARN) role_arnを指定し、Terraform セッションの一意の識別子session_nameです。
- aws_db_instance.primary リソースは、リージョンのprimaryプロバイダーを使用してプライマリ Amazon RDS データベースを作成しますus-west-1。
- aws_db_instance.read_replica リソースは、 replicaプロバイダーを使用して、 us-east-1リージョンにプライマリデータベースのリードレプリカを作成します。replicate_source_db 属性はprimaryデータベースの ID を参照します。

セキュリティのベストプラクティス

Terraform AWS プロバイダーを安全に使用するには、認証、アクセスコントロール、セキュリティを適切に管理することが重要です。このセクションでは、以下に関するベストプラクティスの概要を説明します。

- 最小特権アクセスの IAM ロールとアクセス許可
- AWS アカウントとリソースへの不正アクセスを防ぐための認証情報の保護
- 機密データの保護に役立つリモート状態の暗号化
- インフラストラクチャとソースコードのスキャンによる設定ミスの特定
- リモートステートストレージのアクセスコントロール
- ガバナンスガードレールを実装するためのセンチネルポリシーの適用

これらのベストプラクティスに従うことで、Terraform を使用して AWS インフラストラクチャを管理する際のセキュリティ体制を強化できます。

最小特権の原則に従う

最小特権は、ユーザー、プロセス、またはシステムが意図した機能を実行するために必要な最小限のアクセス許可のみを付与することを指す基本的なセキュリティ原則です。これは、アクセスコントロールの中核的な概念であり、不正アクセスや潜在的なデータ侵害に対する予防手段です。

最小特権の原則は、Terraform が などのクラウドプロバイダーに対して認証およびアクションを実行する方法に直接関係するため、このセクションでは複数回強調されます AWS。

Terraform を使用して AWS リソースをプロビジョニングおよび管理する場合、API コールを行うための適切なアクセス許可を必要とするエンティティ (ユーザーまたはロール) に代わって動作します。最低限の特権に従うことで、重大なセキュリティリスクが生じます。

- Terraform に必要なアクセス許可を超えると、意図しない設定ミスによって望ましくない変更や削除が行われる可能性があります。
- アクセス許可が過度に許可されると、Terraform 状態ファイルまたは認証情報が侵害された場合の影響範囲が拡大します。
- 最小限の権限に従うことは、最小限のアクセスを許可するためのセキュリティのベストプラクティスと規制コンプライアンス要件に反します。

IAM ロールの使用

Terraform AWS プロバイダーでセキュリティを強化するために、可能な限り IAM ユーザーの代わりに IAM ロールを使用します。IAM ロールは、自動的にローテーションする一時的なセキュリティ認証情報を提供するため、長期的なアクセスキーを管理する必要がなくなります。ロールは、IAM ポリシーを通じて正確なアクセスコントロールも提供します。

IAM ポリシーを使用して最小特権アクセスを付与する

IAM ポリシーを慎重に構築して、ロールとユーザーがワークロードに必要な最小限のアクセス許可のセットのみを持つようにします。空のポリシーから開始し、許可されたサービスとアクションを繰り返し追加します。これを行うには、以下の手順を使用します。

- IAM Access Analyzer を有効にしてポリシーを評価し、削除できる未使用のアクセス許可を強調表示します。
- ポリシーを手動で確認して、ロールの目的の責任に不可欠ではない機能をすべて削除します。
- IAM ポリシー変数とタグを使用して、アクセス許可の管理を簡素化します。

適切に構築されたポリシーは、ワークロードの責任を達成するために十分なアクセス権を付与します。オペレーションレベルでアクションを定義し、特定のリソースで必要な APIs への呼び出しのみを許可します。

このベストプラクティスに従うことで、影響範囲が軽減され、職務の分離と最小特権アクセスという基本的なセキュリティ原則が適用されます。必要に応じて、オープンを開始して後でアクセスを制限するのではなく、厳格かつオープンなアクセスを徐々に開始します。

ローカル認証用の IAM ロールを引き受ける

Terraform をローカルで実行するときは、静的アクセスキーを設定しないでください。代わりに、IAM ロールを使用して、長期的な認証情報を公開せずに特権アクセスを一時的に付与します。

まず、必要な最小限のアクセス許可を持つ IAM ロールを作成し、ユーザーアカウントまたはフェデレーティッド ID が IAM ロールを引き受けることを許可する信頼関係を追加します。これにより、ロールの一時的な使用が許可されます。

信頼関係ポリシーの例:

```
{
  "Version": "2012-10-17",
```

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

次に、aws sts assume-role AWS CLI コマンドを実行して、ロールの存続期間の短い認証情報を取得します。これらの認証情報は通常 1 時間有効です。

AWS CLI コマンドの例:

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

コマンドの出力には、認証に使用できるアクセスキー、シークレットキー、およびセッショントークンが含まれています 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 プロバイダーは、ロールの引き受けを自動的に処理することもできます。

IAM ロールを引き受けるプロバイダー設定の例:

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

これにより、Terraform セッションの期間に対してのみ昇格された権限が付与されます。一時キーは、セッションの最大期間後に自動的に期限切れになるため、リークできません。

このベストプラクティスの主な利点には、存続期間の長いアクセスキーと比較したセキュリティの向上、最小特権のロールに対するきめ細かなアクセスコントロール、ロールのアクセス許可を変更してアクセスを簡単に取り消す機能などがあります。IAM ロールを使用すると、シークレットをスクリプトまたはディスクに直接保存する必要もなくなります。これにより、チーム間で Terraform 設定を安全に共有できます。

Amazon EC2 認証に IAM ロールを使用する

Amazon Elastic Compute Cloud (Amazon EC2) インスタンスから Terraform を実行する場合は、長期的な認証情報をローカルに保存しないでください。代わりに、IAM ロールとインスタンスプロファイルを使用して、最小特権のアクセス許可を自動的に付与します。

まず、最小限のアクセス許可を持つ IAM ロールを作成し、そのロールをインスタンスプロファイルに割り当てます。インスタンスプロファイルにより、EC2 インスタンスはロールで定義されたアクセス許可を継承できます。次に、そのインスタンスプロファイルを指定してインスタンスを起動します。インスタンスは、アタッチされたロールを通じて認証されます。

Terraform オペレーションを実行する前に、ロールがインスタンスメタデータに存在することを確認し、認証情報が正常に継承されたことを確認します。

```
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/
```

このアプローチにより、永続的 AWS なキーをスクリプトやインスタンス内の Terraform 設定にハードコーディングすることを回避できます。一時的な認証情報は、インスタンスのロールとプロファイルを通じて Terraform で透過的に利用できるようになります。

このベストプラクティスの主な利点には、長期的な認証情報に対するセキュリティの向上、認証情報管理のオーバーヘッドの削減、開発、テスト、本番環境間の一貫性などがあります。IAM ロール認証は、最小特権アクセスを適用しながら、EC2 インスタンスからの Terraform 実行を簡素化します。

HCP Terraform ワークスペースに動的認証情報を使用する

HCP Terraform は HashiCorp が提供するマネージドサービスで、チームが Terraform を使用して複数のプロジェクトや環境でインフラストラクチャをプロビジョニングおよび管理できるようにします。HCP Terraform で Terraform を実行する場合は、動的認証情報を使用して AWS 認証を簡素化し、保護します。Terraform は、IAM ロールを引き受けることなく、実行ごとに一時的な認証情報を自動的に交換します。

利点には、シークレットのローテーションの簡素化、ワークスペース間の認証情報の一元管理、最小特権のアクセス許可、ハードコードされたキーの排除などがあります。ハッシュされたエフェメラルキーに依存すると、存続期間の長いアクセスキーと比較してセキュリティが向上します。

で IAM ロールを使用する AWS CodeBuild

で AWS CodeBuild、CodeBuild プロジェクトに割り当てられた IAM ロールを使用してビルドを実行します。これにより、各ビルドは長期キーを使用する代わりに、ロールから一時的な認証情報を自動的に継承できます。

HCP Terraform で GitHub Actions をリモートで実行する

HCP Terraform ワークスペースで Terraform をリモートで実行するように GitHub Actions ワークフローを設定します。GitHub シークレット管理の代わりに、動的認証情報とリモート状態ロックに依存します。

OIDC で GitHub Actions を使用し、 AWS 認証情報アクションを設定する

OpenID Connect (OIDC) 標準を使用して、IAM を介して GitHub Actions ID をフェデレーションします。AWS 認証情報の設定 アクションを使用して、長期アクセスキーを必要とせずに GitHub トークンを一時的な AWS 認証情報と交換します。

OIDC と で GitLab を使用する AWS CLI

OIDC 標準を使用して、一時的なアクセスのために IAM を介して GitLab ID をフェデレーションします。OIDC に依存することで、GitLab 内で長期的な AWS アクセスキーを直接管理する必要がなく

なります。認証情報はjust-in-timeで交換されるため、セキュリティが向上します。ユーザーは、IAM ロールのアクセス許可に従って最小特権アクセスも取得します。

レガシーオートメーションツールで一意の IAM ユーザーを使用する

IAM ロールの使用をネイティブにサポートしていない自動化ツールやスクリプトがある場合は、個々の IAM ユーザーを作成してプログラムによるアクセスを許可できます。最小特権の原則は引き続き適用されます。ポリシーのアクセス許可を最小限に抑え、パイプラインまたはスクリプトごとに個別のロールに依存します。より最新のツールやスクリプトに移行するときは、ロールをネイティブにサポートし始め、徐々にロールに移行します。

> **Warning**
>
> IAM ユーザーには長期的な認証情報があり、セキュリティ上のリスクがあります。このリスクを軽減するために、これらのユーザーにはタスクの実行に必要な権限のみを付与し、不要になったユーザーは削除することをお勧めします。

Jenkins AWS 認証情報プラグインを使用する

AWS Jenkins の認証情報プラグインを使用して、 AWS 認証情報を一元的に設定し、ビルドに動的に挿入できます。これにより、シークレットをソースコントロールにチェックする必要がなくなります。

最小特権を継続的にモニタリング、検証、最適化する

時間の経過とともに、必要な最小ポリシーを超える可能性のある追加のアクセス許可が付与される可能性があります。アクセスを継続的に分析して、不要な使用権限を特定して削除します。

アクセスキーの使用状況を継続的にモニタリングする

アクセスキーの使用を回避できない場合は、IAM 認証情報レポートを使用して、90 日以上経過した未使用のアクセスキーを検索し、ユーザーアカウントとマシンロールの両方で非アクティブなキーを取り消します。アクティブな従業員とシステムのキーの削除を手動で確認するように管理者に警告します。

キーの使用状況をモニタリングすると、未使用の使用権限を特定して削除できるため、アクセス許可を最適化できます。アクセスキーローテーションでこのベストプラクティスに従うと、認証情報の有効期間が制限され、最小特権アクセスが適用されます。

AWS には、管理者のアラートと通知をセットアップするために使用できるいくつかのサービスと機能が用意されています。いくつかのオプションは次のとおりです。

- AWS Config: AWS Config ルールを使用して、IAM アクセスキーを含む AWS リソースの設定を評価できます。カスタムルールを作成して、特定の日数より古い未使用のアクセスキーなど、特定の条件を確認できます。ルールに違反すると、 は修復の評価を開始したり、Amazon Simple Notification Service (Amazon SNS) トピックに通知を送信 AWS Config したりできます。
- AWS Security Hub CSPM: Security Hub CSPM は、 AWS アカウントのセキュリティ体制を包括的に把握し、未使用または非アクティブな IAM アクセスキーなど、潜在的なセキュリティ問題を検出して通知するのに役立ちます。Security Hub CSPM は、チャットアプリケーションで Amazon EventBridge および Amazon SNS または Amazon Q Developer と統合して、管理者に通知を送信できます。
- AWS Lambda: Lambda 関数は、Amazon CloudWatch Events や AWS Config ルールなど、さまざまなイベントで呼び出すことができます。カスタム Lambda 関数を記述して、チャットアプリケーションで Amazon SNS や Amazon Q Developer などのサービスを使用して、IAM アクセスキーの使用状況を評価し、追加のチェックを実行し、通知を送信できます。

IAM ポリシーを継続的に検証する

IAM Access Analyzer を使用して、ロールにアタッチされているポリシーを評価し、付与された未使用のサービスや余分なアクションを特定します。定期的なアクセスレビューを実装して、ポリシーが現在の要件を満たしていることを手動で確認します。

既存のポリシーを IAM Access Analyzer によって生成されたポリシーと比較し、不要なアクセス許可を削除します。また、ユーザーにレポートを提供し、猶予期間後に未使用のアクセス許可を自動的に取り消す必要があります。これにより、最小限のポリシーが引き続き有効になります。

古いアクセスをプロアクティブかつ頻繁に取り消すと、侵害中にリスクにさらされる可能性のある認証情報が最小限に抑えられます。自動化は、持続可能で長期的な認証情報の衛生とアクセス許可の最適化を提供します。このベストプラクティスに従うことで、 AWS アイデンティティとリソースに最小特権を積極的に適用することで、影響の範囲を制限できます。

安全なリモート状態ストレージ

リモート状態ストレージとは、Terraform が実行されているマシンにローカルではなく、Terraform 状態ファイルをリモートに保存することです。状態ファイルは、Terraform によってプロビジョニングされたリソースとそのメタデータを追跡するため、重要です。

リモート状態を保護しないと、状態データの損失、インフラストラクチャの管理不能、不注意によるリソースの削除、状態ファイルに存在する可能性のある機密情報の漏洩などの重大な問題が発生する可能性があります。このため、本番稼働用グレードの Terraform の使用には、リモートステートストレージの保護が不可欠です。

暗号化とアクセスコントロールを有効にする

Amazon Simple Storage Service (Amazon S3) サーバー側の暗号化 (SSE) を使用して、保管中のリモート状態を暗号化します。

コラボレーションワークフローへの直接アクセスを制限する

- HCP Terraform または Git リポジトリ内の CI/CD パイプラインでコラボレーションワークフローを構築し、直接的な状態アクセスを制限します。
- プルリクエスト、実行承認、ポリシーチェック、通知に依存して変更を調整します。

これらのガイドラインに従うことで、機密性の高いリソース属性を保護し、チームメンバーの変更との競合を回避できます。暗号化と厳格なアクセス保護は攻撃対象領域の削減に役立ち、コラボレーションワークフローにより生産性が向上します。

を使用する AWS Secrets Manager

Terraform には、状態ファイルにシークレット値をプレーンテキストで保存するリソースとデータソースが多数あります。シークレットを 状態に保存することは避けてください。AWS Secrets Manager代わりに を使用してください。

機密値を手動で暗号化するのではなく、Terraform の機密状態管理の組み込みサポートを利用してください。機密値を出力にエクスポートする場合は、その値が機密としてマークされていることを確認してください。

インフラストラクチャとソースコードを継続的にスキャンする

インフラストラクチャとソースコードの両方をプロアクティブにスキャンして、公開された認証情報や設定ミスなどのリスクがないか確認し、セキュリティ体制を強化します。リソースを再設定またはパッチ適用して、検出結果に迅速に対処します。

動的スキャンに AWS サービスを使用する

Amazon Inspector、、Amazon DetectiveAWS Security Hub CSPM、Amazon GuardDuty などの AWS ネイティブツールを使用して、アカウントとリージョン全体でプロビジョニングされたインフラストラクチャをモニタリングします。Security Hub CSPM で定期的なスキャンをスケジュールして、デプロイと設定のドリフトを追跡します。EC2 インスタンス、Lambda 関数、コンテナ、S3 バケット、およびその他のリソースをスキャンします。

静的分析を実行する

Checkov などの静的アナライザーを CI/CD パイプラインに直接埋め込み、Terraform 設定コード (HCL) をスキャンして、デプロイ前にリスクを事前に特定します。これにより、セキュリティチェックが開発プロセスの以前の時点 (左へのシフトと呼ばれます) に移動され、インフラストラクチャの設定ミスが防止されます。

プロンプトの修正を確認する

すべてのスキャン結果について、Terraform 設定の更新、パッチの適用、または必要に応じてリソースを手動で再設定することで、迅速な修復を確保します。根本原因に対処することで、リスクレベルを下げます。

インフラストラクチャスキャンとコードスキャンの両方を使用すると、Terraform 設定、プロビジョニングされたリソース、およびアプリケーションコード間のレイヤードインサイトが得られます。これにより、ソフトウェア開発ライフサイクル (SDLC) の早い段階でセキュリティを埋め込むと同時に、予防的、検出的、事後対応的なコントロールを通じてリスクとコンプライアンスのカバレッジを最大化できます。

ポリシーチェックを強制する

HashiCorp Sentinel ポリシーなどのコードフレームワークを使用して、Terraform によるインフラストラクチャプロビジョニングのためのガバナンスガードレールと標準化されたテンプレートを提供します。

Sentinel ポリシーでは、組織の標準とベストプラクティスに合わせて Terraform 設定の要件または制限を定義できます。たとえば、Sentinel ポリシーを使用して次のことができます。

- すべてのリソースにタグが必要です。
- インスタンスタイプを承認済みリストに制限します。
- 必須変数を適用します。
- 本番稼働用リソースの破壊を防止します。

ポリシーチェックを Terraform 設定ライフサイクルに埋め込むと、標準とアーキテクチャガイドラインを積極的に適用できます。Sentinel は、未承認のプラクティスを防止しながら開発を加速するのに役立つ共有ポリシーロジックを提供します。

バックエンドのベストプラクティス

適切なリモートバックエンドを使用してステートファイルを保存することは、コラボレーションの有効化、ロックによるステートファイルの整合性の確保、信頼性の高いバックアップとリカバリの提供、CI/CD ワークフローとの統合、HCP Terraform などのマネージドサービスが提供する高度なセキュリティ、ガバナンス、管理機能の利用に不可欠です。

Terraform は、Kubernetes、HashiCorp Consul、HTTP などのさまざまなバックエンドタイプをサポートしています。ただし、このガイドでは、ほとんどの AWS ユーザーに最適なバックエンドソリューションである Amazon S3 に焦点を当てています。

高い耐久性と可用性を提供するフルマネージドオブジェクトストレージサービスである Amazon S3 は、Terraform 状態を管理するための安全でスケーラブルで低コストのバックエンドを提供します AWS。Amazon S3 のグローバルフットプリントとレジリエンスは、ほとんどのチームがステートストレージを自己管理することで達成できる範囲を超えています。さらに、 AWS アクセスコントロール、暗号化オプション、バージョニング機能、およびその他のサービスとネイティブに統合されているため、Amazon S3 は便利なバックエンドの選択肢になります。

このガイドでは、Kubernetes や Consul などの他のソリューションのバックエンドガイダンスは提供していません。主な対象者は AWS 顧客であるためです。に完全に属しているチームの場合 AWS クラウド、Amazon S3 は通常、Kubernetes または HashiCorp Consul クラスターよりも理想的な選択肢です。Amazon S3 ステートストレージのシンプルさ、耐障害性、緊密な AWS 統合は、 AWS ベストプラクティスに従うほとんどのユーザーに最適な基盤を提供します。チームは、 AWS サービスの耐久性、バックアップ保護、可用性を活用して、リモート Terraform の状態の耐障害性を高めることができます。

このセクションのバックエンドの推奨事項に従うことで、エラーや不正な変更の影響を制限しながら、より協調的な Terraform コードベースにつながります。適切に設計されたリモートバックエンドを実装することで、チームは Terraform ワークフローを最適化できます。

ベストプラクティス：

- リモートストレージに Amazon S3 を使用する
- チームコラボレーションの促進
- 各環境のバックエンドを分離する
- リモート状態アクティビティを積極的にモニタリングする

リモートストレージに Amazon S3 を使用する

Amazon DynamoDB を使用して Terraform 状態を Amazon S3 にリモートで保存し、状態ロックと整合性チェックを実装すると、ローカルファイルストレージよりも大きな利点があります。リモート状態により、チームのコラボレーション、変更の追跡、バックアップ保護、リモートロックが可能になり、安全性が向上します。

エフェメラルローカルストレージまたはセルフマネージドソリューションの代わりに Amazon S3 を S3 Standard ストレージクラス (デフォルト) で使用すると、99.999999999% の耐久性と 99.99% の可用性保護を実現し、偶発的な状態データ損失を防止できます。Amazon S3 や DynamoDB などの AWS マネージドサービスは、ほとんどの組織がストレージを自己管理する際に達成できる範囲を超えるサービスレベルアグリーメント (SLAs) を提供します。リモートバックエンドにアクセスできるようにするには、これらの保護に依存します。

リモート状態ロックを有効にする

ステートロックは、同時書き込み操作を防ぐためにアクセスを制限し、複数のユーザーによる同時変更によるエラーを減らします。Terraform は、Amazon S3 バックエンドの 2 つのロックメカニズムをサポートしています。

- Amazon S3 ネイティブ状態ロック (推奨): Terraform 1.10.0 以降利用可能。Amazon S3 のネイティブロック機能を使用
- DynamoDB 状態ロック (廃止): 今後の Terraform バージョンで削除されるレガシーアプローチ

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

移行中の下位互換性のために、Amazon S3 と DynamoDB の両方のロックを同時に設定できます。ただし、DynamoDB ベースのロックは廃止されているため、Amazon S3 ネイティブロックに移行することをお勧めします。

バージョニングと自動バックアップを有効にする

追加の保護のために、Amazon S3 バックエンド AWS Backup で を使用して自動バージョニングとバックアップを有効にします。バージョニングは、変更が行われるたびに以前のすべてのバージョンの 状態を保持します。また、不要な変更をロールバックしたり、事故から回復したりするために、必要に応じて以前の作業状態のスナップショットを復元することもできます。

必要に応じて以前のバージョンを復元する

バージョニングされた Amazon S3 ステートバケットを使用すると、以前の既知の正常な状態のスナップショットを復元することで、変更を簡単に元に戻すことができます。これにより、偶発的な変更から保護し、追加のバックアップ機能が提供されます。

HCP Terraform を使用する

HCP Terraform は、独自のステートストレージを設定する代わりに、フルマネージド型のバックエンドを提供します。HCP Terraform は、追加の機能のロックを解除しながら、状態と暗号化の安全なストレージを自動的に処理します。

HCP Terraform を使用すると、状態はデフォルトでリモートに保存されるため、組織全体で状態の共有とロックが可能になります。詳細なポリシーコントロールは、状態アクセスと変更を制限するのに役立ちます。

その他の機能には、バージョン管理統合、ポリシーガードレール、ワークフロー自動化、変数管理、SAML とのシングルサインオン統合などがあります。Sentinel ポリシーをコードとして使用してガバナンスコントロールを実装することもできます。

HCP Terraform では Software as a Service (SaaS) プラットフォームを使用する必要がありますが、多くのチームにとって、セキュリティ、アクセスコントロール、自動ポリシーチェック、コラボレーション機能の利点により、Amazon S3 または DynamoDB による自己管理状態ストレージよりも最適な選択肢となります。

GitHub や GitLab などのサービスとの簡単な統合とマイナー設定は、チームワークフローを向上させるためにクラウドツールや SaaS ツールを完全に採用しているユーザーにとっても魅力的です。

チームコラボレーションの促進

リモートバックエンドを使用して、Terraform チームのすべてのメンバー間で状態データを共有します。これにより、チーム全体がインフラストラクチャの変更を可視化できるため、コラボレーションが容易になります。共有バックエンドプロトコルと状態履歴の透明性を組み合わせることで、内部変

更管理が簡素化されます。インフラストラクチャの変更はすべて確立されたパイプラインを経由するため、企業全体のビジネスの俊敏性が向上します。

を使用して説明責任を向上させる AWS CloudTrail

Amazon S3 バケット AWS CloudTrail と統合して、ステートバケットに対して行われた API コールをキャプチャします。CloudTrail イベントをフィルタリングして`PutObject`、 `DeleteObject`,やその他の関連する呼び出しを追跡します。

CloudTrail ログには、状態変更の各 API コールを行ったプリンシパルの AWS ID が表示されます。ユーザーの ID は、マシンアカウントまたはバックエンドストレージを操作するチームのメンバーと照合できます。

CloudTrail ログと Amazon S3 ステートバージョニングを組み合わせて、インフラストラクチャの変更を適用したプリンシパルに結び付けます。複数のリビジョンを分析することで、更新をマシンアカウントまたは担当チームメンバーに関連付けることができます。

意図しない変更や破壊的な変更が発生した場合、ステートバージョニングはロールバック機能を提供します。CloudTrail は変更をユーザーにトレースするため、予防的改善について議論できます。

また、IAM アクセス許可を適用してステートバケットへのアクセスを制限することもお勧めします。全体として、S3 バージョニングと CloudTrail モニタリングは、インフラストラクチャの変更全体の監査をサポートします。チームは、Terraform の状態履歴に対する説明責任、透明性、監査機能を強化できます。

各環境のバックエンドを分離する

アプリケーション環境ごとに個別の Terraform バックエンドを使用します。バックエンドは、開発、テスト、本番環境間で分離状態を分離します。

影響範囲の縮小

分離状態は、より低い環境の変化が本稼働インフラストラクチャに影響を与えないようにするのに役立ちます。開発環境やテスト環境における事故や実験の影響は限られています。

本番稼働用アクセスの制限

本番稼働状態のバックエンドのアクセス許可を、ほとんどのユーザーの読み取り専用アクセスにロックダウンします。本番稼働用インフラストラクチャを変更できるユーザーを CI/CD パイプラインに制限し、Break Glass ロールを制限します。

アクセスコントロールの簡素化

バックエンドレベルでのアクセス許可の管理は、環境間のアクセスコントロールを簡素化します。アプリケーションと環境ごとに異なる S3 バケットを使用すると、バックエンドバケット全体に広範な読み取りまたは書き込みアクセス許可を付与できます。

共有ワークスペースを避ける

Terraform ワークスペースを使用して環境間で状態を分離することはできますが、個別のバックエンドはより強力な分離を提供します。共有ワークスペースがある場合でも、事故は複数の環境に影響を与える可能性があります。

環境バックエンドを完全に分離することで、単一の障害や違反の影響を最小限に抑えることができます。個別のバックエンドも、アクセスコントロールを環境の機密性レベルに合わせます。たとえば、本番環境の書き込み保護と、開発環境とテスト環境の広範なアクセスを提供できます。

リモート状態アクティビティを積極的にモニタリングする

リモート状態のアクティビティを継続的にモニタリングすることは、潜在的な問題を早期に検出するために重要です。異常なロック解除、変更、またはアクセス試行を探します。

不審なロック解除に関するアラートを取得する

ほとんどの状態変更は、CI/CD パイプラインを介して実行する必要があります。状態のロック解除がデベロッパーワークステーションを介して直接発生した場合はアラートを生成します。デベロッパーワークステーションは、許可されていない変更やテストされていない変更を通知する可能性があります。

アクセス試行のモニタリング

ステートバケットでの認証の失敗は、偵察アクティビティを示している可能性があります。複数のアカウントが 状態にアクセスしようとしているか、異常な IP アドレスが表示されて、認証情報が侵害されたことを示します。

コードベースの構造と組織のベストプラクティス

大規模なチームや企業で Terraform の使用が増加するにつれて、適切なコードベース構造と組織ことが重要です。優れた設計のコードベースにより、大規模なコラボレーションが可能になり、保守性が向上します。

このセクションでは、品質と一貫性をサポートする Terraform のモジュール性、命名規則、ドキュメント、コーディング標準に関する推奨事項を提供します。

ガイダンスには、環境とコンポーネント別の再利用可能なモジュールへの設定の分割、プレフィックスとサフィックスを使用した命名規則の確立、モジュールの文書化と入出力の明確な説明、自動スタイルチェックを使用した一貫したフォーマットルールの適用が含まれます。

その他のベストプラクティスでは、モジュールとリソースを構造化階層に論理的に整理し、ドキュメントでパブリックモジュールとプライベートモジュールをカタログ化し、モジュールで不要な実装の詳細を抽象化して使用を簡素化します。

モジュール性、ドキュメント、標準、論理的な組織に関するコードベース構造のガイドラインを実装することで、組織全体に使用量が分散するにつれて Terraform を維持したまま、チーム間の広範なコラボレーションをサポートできます。規則と標準を適用することで、フラグメント化されたコードベースの複雑さを回避できます。

ベストプラクティス:

- 標準リポジトリ構造を実装する
- モジュール性の構造
- 命名規則に従う
- アタッチメントリソースを使用する
- デフォルトのタグを使用する
- Terraform レジストリ要件を満たす
- 推奨されるモジュールソースを使用する
- コーディング標準に従う

標準リポジトリ構造を実装する

次のリポジトリレイアウトを実装することをお勧めします。これらの整合性プラクティスをモジュール間で標準化することで、検出可能性、透明性、整理、信頼性が向上し、多くの Terraform 設定で再利用できます。

- ルートモジュールまたはディレクトリ: これは Terraform ルートモジュールと再利用可能なモジュールの両方のプライマリエントリポイントであり、一意であることが期待されます。より複雑なアーキテクチャがある場合は、ネストされたモジュールを使用して軽量な抽象化を作成できます。これにより、物理オブジェクトの観点からではなく、アーキテクチャの観点からインフラストラクチャを記述できます。
- README: ルートモジュールとネストされたモジュールには README ファイルが必要です。このファイルの名前は である必要がありますREADME.md。これには、モジュールの説明と使用目的が含まれている必要があります。このモジュールを他のリソースで使用する例を含める場合は、examples ディレクトリに配置します。モジュールが作成するインフラストラクチャリソースとその関係を示す図を含めることを検討してください。terraform-docs を使用して、モジュールの入力または出力を自動的に生成します。
- main.tf: これはプライマリエントリポイントです。シンプルなモジュールの場合、すべてのリソースがこのファイルに作成される場合があります。複雑なモジュールの場合、リソースの作成は複数のファイルに分散される場合がありますが、ネストされたモジュールの呼び出しは main.tf ファイル内に存在する必要があります。
- variables.tf および outputs.tf: これらのファイルには、変数と出力の宣言が含まれています。すべての変数と出力には、その目的を説明する 1 文または 2 文の説明が必要です。これらの説明はドキュメントに使用されます。詳細については、変数設定と出力設定の HashiCorp ドキュメントを参照してください。
  - すべての変数には定義された型が必要です。
  - 変数宣言には、デフォルトの引数を含めることもできます。宣言にデフォルトの引数が含まれている場合、変数はオプションと見なされ、モジュールを呼び出すか Terraform を実行するときに値を設定しない場合、デフォルト値が使用されます。デフォルトの引数にはリテラル値が必要であり、設定内の他のオブジェクトを参照することはできません。変数を必須にするには、変数宣言でデフォルトを省略し、 設定nullable = falseが理にかなっているかどうかを考慮します。
  - 環境に依存しない値 (などdisk_size) を持つ変数には、デフォルト値を指定します。
  - 環境固有の値 (などproject_id) を持つ変数の場合は、デフォルト値を指定しないでください。この場合、呼び出し元のモジュールは意味のある値を指定する必要があります。

  - 空の文字列やリストなどの変数には、変数を空のままにすることが、基になる APIs が拒否しない有効な設定である場合にのみ、空のデフォルトを使用します。
  - 変数の使用には慎重を期してください。値は、インスタンスまたは環境ごとに変更する必要がある場合にのみパラメータ化します。変数を公開するかどうかを決定するときは、その変数を変更するための具体的なユースケースがあることを確認してください。変数が必要になる可能性がわずかしかない場合は、公開しないでください。
    - デフォルト値を持つ変数を追加すると、下位互換性があります。
    - 変数の削除には下位互換性がありません。
    - リテラルが複数の場所で再利用される場合は、変数として公開せずにローカル値を使用する必要があります。
  - 入力変数は依存関係グラフに適切に追加されないため、出力を直接渡さないでください。暗黙的な依存関係が作成されるように、 がリソースから属性を参照するようにします。インスタンスの入力変数を直接参照する代わりに、 属性を渡します。
- locals.tf: このファイルには、式に名前を割り当てるローカル値が含まれているため、式を繰り返す代わりに、モジュール内で名前を複数回使用できます。ローカル値は、関数の一時的なローカル変数のようなものです。ローカル値の式はリテラル定数に制限されません。変数、リソース属性、その他のローカル値など、モジュール内の他の値を参照して組み合わせることもできます。
- providers.tf: このファイルには、Terraform ブロックとプロバイダーブロックが含まれています。`provider`ブロックは、モジュールのコンシューマーがルートモジュールでのみ宣言する必要があります。

  HCP Terraform を使用している場合は、空のクラウドブロックも追加します。`cloud` ブロックは、CI/CD パイプラインの一部として、環境変数と環境変数認証情報を使用して完全に設定する必要があります。
- versions.tf: このファイルには required_providers ブロックが含まれています。すべての Terraform モジュールは、Terraform がこれらのプロバイダーをインストールして使用できるように、必要なプロバイダーを宣言する必要があります。
- data.tf: シンプルな設定の場合は、データソースを参照するリソースの横にデータソースを配置します。たとえば、インスタンスの起動に使用するイメージを取得する場合は、独自のファイルにデータリソースを収集するのではなく、インスタンスと一緒に配置します。データソースの数が多すぎる場合は、専用`data.tf`ファイルに移動することを検討してください。
- .tfvars ファイル: ルートモジュールの場合、 `.tfvars` ファイルを使用して、機密性のない変数を指定できます。整合性を保つには、変数ファイルに という名前を付けます`terraform.tfvars`。リポジトリのルートに共通の値を配置し、 `envs/`フォルダ内に環境固有の値を配置します。

- ネストされたモジュール: ネストされたモジュールは modules/サブディレクトリに存在する必要があります。を持つネストされたモジュールはREADME.md、外部ユーザーが使用できると見なされます。README.md が存在しない場合、モジュールは内部使用のみと見なされます。ネストされたモジュールを使用して、複雑な動作をユーザーが慎重に選択して選択できる複数の小さなモジュールに分割する必要があります。

  ルートモジュールにネストされたモジュールへの呼び出しが含まれている場合、これらの呼び出しは、Terraform がそれらを個別にダウンロードするのではなく、同じリポジトリまたはパッケージの一部と見な./modules/sample-moduleすように、 などの相対パスを使用する必要があります。

  リポジトリまたはパッケージに複数のネストされたモジュールが含まれている場合は、相互に直接呼び出してモジュールの深いネストされたツリーを作成するのではなく、呼び出し元が構成できるのが理想的です。

- 例: 再利用可能なモジュールを使用する例は、リポジトリのルートの examples/サブディレクトリに存在する必要があります。各例について、README を追加して、例の目標と使用方法を説明することができます。サブモジュールの例は、ルートexamples/ディレクトリにも配置する必要があります。

  多くの場合、例はカスタマイズのために他のリポジトリにコピーされるため、モジュールブロックのソースは、相対パスではなく、外部発信者が使用するアドレスに設定する必要があります。

- サービス名付きファイル: ユーザーは多くの場合、複数のファイルでサービスごとに Terraform リソースを分離します。この方法はできるだけ推奨せず、main.tf代わりに でリソースを定義する必要があります。ただし、リソースのコレクション (IAM ロールやポリシーなど) が 150 行を超える場合は、 などの独自のファイルに分割することをお勧めしますiam.tf。それ以外の場合は、すべてのリソースコードを で定義する必要がありますmain.tf。

- カスタムスクリプト: 必要な場合にのみスクリプトを使用します。Terraform は、スクリプトによって作成されたリソースの状態を考慮または管理しません。Terraform リソースが目的の動作をサポートしていない場合にのみ、カスタムスクリプトを使用します。Terraform によって呼び出されたカスタムスクリプトを scripts/ ディレクトリに配置します。

- ヘルパースクリプト: ディレクトリで Terraform によって呼び出されないヘルパースクリプトを整理しますhelpers/。説明と呼び出し例を使用して、 README.md ファイルにヘルパースクリプトをドキュメント化します。ヘルパースクリプトが引数を受け入れる場合は、引数チェックと--help出力を指定します。

- 静的ファイル: Terraform が参照するが実行しない静的ファイル (EC2 インスタンスにロードされたスタートアップスクリプトなど) は、 `files/` ディレクトリに整理する必要があります。HCL とは別に、外部ファイルに長いドキュメントを配置します。file() 関数で参照します。
- テンプレート: Terraform templatefile 関数が読み取るファイルには、ファイル拡張子 を使用します `.tftpl`。テンプレートは `templates/` ディレクトリに配置する必要があります。

ルートモジュール構造

Terraform は常に単一のルートモジュールのコンテキストで実行されます。完全な Terraform 設定は、ルートモジュールと子モジュールのツリー (ルートモジュールによって呼び出されるモジュール、それらのモジュールによって呼び出されるモジュールなどを含む) で構成されます。

Terraform ルートモジュールレイアウトの基本的な例:

```
.
### 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
```

再利用可能なモジュール構造

再利用可能なモジュールは、ルートモジュールと同じ概念に従います。モジュールを定義するには、ルートモジュールを定義するのと同様に、新しいディレクトリを作成し、その中に `.tf`ファイルを配置します。Terraform は、ローカル相対パスまたはリモートリポジトリからモジュールをロードできます。モジュールが多くの設定で再利用されることが予想される場合は、独自のバージョン管理リポジトリに配置します。さまざまな組み合わせでモジュールを簡単に再利用できるように、モジュールツリーを比較的平らに保つことが重要です。

Terraform 再利用可能なモジュールレイアウトの基本例:

```
.
### 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
```

モジュール性の構造

原則として、任意のリソースやその他のコンストラクトをモジュールに結合できますが、ネストされた再利用可能なモジュールを過剰に使用すると、Terraform 設定全体の理解と維持が困難になる可能性があるため、これらのモジュールをモデレーションで使用します。

理にかなっている場合は、リソースタイプから構築されたアーキテクチャの新しい概念を記述することで、抽象化のレベルを高める再利用可能なモジュールに構成を分割します。

インフラストラクチャを再利用可能な定義にモジュール化する場合は、個々のコンポーネントや過度に複雑なコレクションではなく、リソースの論理セットを目指します。

単一のリソースをラップしない

他の単一のリソースタイプを囲む薄いラッパーのモジュールを作成しないでください。モジュール内のメインリソースタイプの名前とは異なるモジュールの名前を見つけられない場合、モジュールが新しい抽象化を作成していない可能性があります。不要な複雑さが発生しています。代わりに、呼び出し元のモジュールで リソースタイプを直接使用します。

論理関係をカプセル化する

ネットワーク基盤、データ層、セキュリティコントロール、アプリケーションなどの関連リソースのグループセット。再利用可能なモジュールは、機能を有効にするために連携するインフラストラクチャ部分をカプセル化する必要があります。

継承を平らに保つ

モジュールをサブディレクトリにネストする場合は、1 つまたは 2 つのレベルを深くしないようにします。深くネストされた継承構造は、設定とトラブルシューティングを複雑にします。モジュールは他のモジュール上に構築する必要があります。それらを介してトンネルを構築しないでください。

アーキテクチャパターンを表す論理リソースグループに焦点を当てることで、チームは信頼性の高いインフラストラクチャ基盤をすばやく設定できます。過剰エンジニアリングや過剰簡素化なしで抽象化のバランスを取ります。

出力のリソースを参照する

再利用可能なモジュールで定義されているリソースごとに、リソースを参照する出力を少なくとも 1 つ含めます。変数と出力を使用すると、モジュールとリソース間の依存関係を推測できます。出力がないと、ユーザーは Terraform 設定に関連してモジュールを適切に注文できません。

環境の一貫性、目的主導のグループ化、エクスポートされたリソースリファレンスを提供する構造化されたモジュールにより、組織全体の Terraform コラボレーションを大規模に実現できます。チームは、再利用可能な構成要素からインフラストラクチャを組み立てることができます。

プロバイダーを設定しない

共有モジュールはモジュールの呼び出しからプロバイダーを継承しますが、モジュールはプロバイダー設定自体を設定しないでください。モジュールでプロバイダー設定ブロックを指定しないでください。この設定はグローバルに 1 回だけ宣言する必要があります。

必要なプロバイダーを宣言する

プロバイダー設定はモジュール間で共有されますが、共有モジュールは独自のプロバイダー要件も宣言する必要があります。この方法により、Terraform は、設定内のすべてのモジュールと互換性のあるプロバイダーの単一バージョンがあることを確認し、プロバイダーのグローバル (モジュールに依存しない) 識別子として機能するソースアドレスを指定できます。ただし、モジュール固有のプロバイダー要件では、 など、プロバイダーがどのリモートエンドポイントにアクセスするかを決定する設定は指定されません AWS リージョン。

バージョン要件を宣言し、ハードコードされたプロバイダー設定を回避することで、モジュールは共有プロバイダーを使用して Terraform 設定間で移植性と再利用性を提供します。

共有モジュールの場合、 の required_providers ブロックで最低限必要なプロバイダーバージョンを定義しますversions.tf。

モジュールが特定のバージョンの AWS プロバイダーを必要とすることを宣言するには、required_providersブロック内で terraformブロックを使用します。

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

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

共有モジュールが特定のバージョンの AWS プロバイダーのみをサポートしている場合は、悲観的制約演算子 (~>) を使用します。これにより、右端のバージョンコンポーネントのみが増分されます。

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

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

この例では、 は 4.57.1と のインストール~> 4.0を許可します4.67.0が、 のインストールは許可しません5.0.0。詳細については、HashiCorp ドキュメントの「バージョン制約構文」を参照してください。

命名規則に従う

わかりやすいわかりやすい名前を使用すると、モジュール内のリソースと設定値の目的との関係を簡単に理解できます。スタイルガイドラインとの整合性により、モジュールユーザーとメンテナーの両方の読みやすさが向上します。

リソースの命名に関するガイドラインに従う

- Terraform スタイルの標準に一致するように、すべてのリソース名に snake_case (小文字はアンダースコアで区切られます) を使用します。この方法により、リソースタイプ、データソースタイプ、およびその他の事前定義された値の命名規則との整合性が確保されます。この規則は name 引数には適用されません。
- そのタイプの唯一のリソース (モジュール全体の 1 つのロードバランサーなど) への参照を簡素化するには、リソースに mainまたは という名前をthis付けます。
- リソースの目的とコンテキストを記述し、同様のリソース (たとえば、メインデータベースprimaryの場合は 、データベースのリードレプリカread_replicaの場合は) を区別するのに役立つ意味のある名前を使用します。
- 複数名ではなく単数名を使用します。
- リソース名でリソースタイプを繰り返しないでください。

変数の命名に関するガイドラインに従う

- ディスクサイズや RAM サイズ (ギガバイト単位の ram_size_gb RAM サイズなど) などの数値を表す入力、ローカル変数、出力の名前に単位を追加します。この方法により、設定メンテナーに対して期待される入力単位が明確になります。

- ストレージサイズには MiB や GiB などのバイナリ単位を使用し、他のメトリクスには MB や GB などの小数単位を使用します。
- などのブール変数に正の名前を付けますenable_external_access。

アタッチメントリソースを使用する

一部のリソースには、擬似リソースが属性として埋め込まれています。可能であれば、これらの埋め込みリソース属性の使用を避け、代わりに一意のリソースを使用してその擬似リソースをアタッチする必要があります。これらのリソース関係は、リソースごとに固有のcause-and-effectの問題を引き起こす可能性があります。

埋め込み属性の使用 (このパターンは避けてください) 。

```
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"]
  }
}
```

アタッチメントリソースの使用 (推奨):

```
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
}
```

デフォルトのタグを使用する

タグを受け入れることができるすべてのリソースにタグを割り当てます。Terraform AWS プロバイダーには、ルートモジュール内で使用する必要がある aws_default_tags データソースがあります。

Terraform モジュールによって作成されたすべてのリソースに必要なタグを追加することを検討してください。アタッチできるタグのリストを次に示します。

- 名前: 人間が読めるリソース名
- AppId: リソースを使用するアプリケーションの ID
- AppRole: リソースの技術機能。たとえば、「webserver」や「database」など。
- AppPurpose: リソースのビジネス目的。例 : 「フロントエンド ui」または「支払いプロセッサ」
- 環境: 開発、テスト、製品などのソフトウェア環境
- プロジェクト: リソースを使用するプロジェクト
- CostCenter: リソース使用量の請求対象

Terraform レジストリ要件を満たす

モジュールリポジトリは、Terraform レジストリに発行できるように、以下のすべての要件を満たしている必要があります。

短期的にモジュールをレジストリに公開する予定がない場合でも、常にこれらの要件に従う必要があります。これにより、リポジトリの設定と構造を変更することなく、後でモジュールをレジストリに発行できます。

- リポジトリ名: モジュールリポジトリには、3 つの部分からなる名前 を使用します。はterraform-aws-<NAME>、モジュールが管理するインフラストラクチャのタイプ<NAME>を反映します。<NAME> セグメントには、追加のハイフン (などterraform-aws-iam-terraform-roles) を含めることができます。

- 標準モジュール構造: モジュールは標準リポジトリ構造に従う必要があります。これにより、レジストリはモジュールを検査し、ドキュメントを生成したり、リソースの使用状況を追跡したりできます。
  - Git リポジトリを作成したら、モジュールファイルをリポジトリのルートにコピーします。再利用可能な各モジュールを独自のリポジトリのルートに配置することをお勧めしますが、サブディレクトリからモジュールを参照することもできます。
  - HCP Terraform を使用している場合は、組織レジストリと共有することを意図したモジュールを発行します。レジストリは、HCP Terraform API トークンを使用してダウンロードを処理してアクセスを制御するため、コンシューマーはコマンドラインから Terraform を実行してもモジュールのソースリポジトリにアクセスする必要はありません。
- 場所とアクセス許可: リポジトリは、設定されたバージョン管理システム (VCS) プロバイダーのいずれかに存在し、HCP Terraform VCS ユーザーアカウントにはリポジトリへの管理者アクセス権が必要です。レジストリには、新しいモジュールバージョンをインポートするためのウェブフックを作成するための管理者アクセスが必要です。
- リリースの x.y.z タグ: モジュールを発行するには、少なくとも 1 つのリリースタグが必要です。レジストリは、リリースタグを使用してモジュールバージョンを識別します。リリースタグ名にはセマンティックバージョニングを使用する必要があります。セマンティックバージョニングには、オプションで のプレフィックスを付けることができます v (例: `v1.1.0`および `1.1.0`)。レジストリは、バージョン番号に似ていないタグを無視します。モジュールの公開の詳細については、Terraform ドキュメントを参照してください。

詳細については、Terraform ドキュメントの「モジュールリポジトリの準備」を参照してください。

推奨されるモジュールソースを使用する

Terraform は、モジュールブロックの `source`引数を使用して、子モジュールのソースコードを検索およびダウンロードします。

繰り返しコード要素を除外することを主な目的とする密接に関連するモジュールにはローカルパスを使用し、複数の設定で共有することが意図されているモジュールにはネイティブの Terraform モジュールレジストリまたは VCS プロバイダーを使用することをお勧めします。

次の例は、モジュールを共有するための最も一般的なソースタイプと推奨されるソースタイプを示しています。レジストリモジュールはバージョニングをサポートしています。次の例に示すように、常に特定のバージョンを指定する必要があります。

レジストリ

Terraform レジストリ:

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

  ...

}
```

コミットハッシュを固定することで、サプライチェーン攻撃に対して脆弱なパブリックレジストリからのドリフトを回避できます。

HCP Terraform:

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

  ...

  enable_karpenter = true
}
```

Terraform Enterprise:

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

  ...

  enable_karpenter = true
}
```

VCS プロバイダー

VCS プロバイダーは、次の例に示すように、特定のリビジョンを選択するための `ref`引数をサポートしています。

GitHub (HTTPS):

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

  ...

  enable_karpenter = true
}
```

汎用 Git リポジトリ (HTTPS):

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

  ...

  enable_karpenter = true
}
```

汎用 Git リポジトリ (SSH):

> ⚠ Warning
>
> プライベートリポジトリにアクセスするには、認証情報を設定する必要があります。

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

  ...

  enable_karpenter = true
}
```

コーディング標準に従う

すべての設定ファイルに一貫した Terraform フォーマットルールとスタイルを適用します。CI/CD パイプラインで自動スタイルチェックを使用して標準を適用します。コーディングのベストプラクティ

スをチームワークフローに埋め込むと、組織全体で使用が広く普及するにつれて、設定は読みやすく、保守可能で、共同作業性が維持されます。

スタイルガイドラインに従う

- HashiCorp スタイルの標準に一致するように、すべての Terraform ファイル (`.tf` ファイル) を terraform fmt コマンドでフォーマットします。
- terraform validate コマンドを使用して、設定の構文と構造を検証します。
- TFLint を使用してコード品質を静的に分析します。この linter は、フォーマットだけでなく、Terraform のベストプラクティスをチェックし、エラーが発生したときにビルドに失敗します。

事前コミットフックを設定する

コミットを許可する前に`terraform fmt`、、`tflint`、`checkov`、、およびその他のコードスキャンとスタイルチェックを実行するクライアント側の事前コミットフックを設定します。このプラクティスは、開発者ワークフローの早い段階で標準への準拠を検証するのに役立ちます。

事前コミットなどの事前コミットフレームワークを使用して、Terraform リンティング、フォーマット、コードスキャンをローカルマシンのフックとして追加します。フックは各 Git コミットで実行され、チェックに合格しなかった場合はコミットに失敗します。

スタイルチェックと品質チェックをローカルのコミット前フックに移動すると、変更が導入される前に開発者に迅速なフィードバックが提供されます。標準はコーディングワークフローの一部になります。

AWS プロバイダーのバージョン管理のベストプラクティス

AWS プロバイダーのバージョンと関連する Terraform モジュールを慎重に管理することは、安定性にとって重要です。このセクションでは、バージョン制約とアップグレードに関するベストプラクティスの概要を説明します。

ベストプラクティス：

- 自動バージョンチェックを追加する
- 新しいリリースのモニタリング
- プロバイダーへの貢献

自動バージョンチェックを追加する

CI/CD パイプラインに Terraform プロバイダーのバージョンチェックを追加してバージョンピンニングを検証し、バージョンが未定義の場合はビルドを失敗させます。

- CI/CD パイプラインに TFLint チェックを追加して、固定されたメジャー/マイナーバージョンの制約が定義されていないプロバイダーバージョンをスキャンします。Terraform AWS Provider の TFLint ルールセットプラグインを使用します。これにより、考えられるエラーを検出するためのルールと、 リソースに関する AWS ベストプラクティスを確認できます。
- ピン留めされていないプロバイダーバージョンを検出する失敗 CI の実行により、暗黙的なアップグレードが本番環境に到達しないようにします。

新しいリリースのモニタリング

- プロバイダーのリリースノートと変更ログフィードをモニタリングします。新しいメジャー/マイナーリリースに関する通知を受け取ります。
- リリースノートを評価して、重大な変更の可能性がないか確認し、既存のインフラストラクチャへの影響を評価します。
- 非本番環境のマイナーバージョンをアップグレードして、本番環境を更新する前に検証します。

パイプラインのバージョンチェックを自動化し、新しいリリースをモニタリングすることで、サポートされていないアップグレードを早期に検出し、本番環境を更新する前に新しいメジャー/マイナーリリースの影響を評価する時間を与えることができます。

プロバイダーへの貢献

欠陥を報告したり、 GitHub 問題の機能をリクエストしたりして HashiCorp AWS 、プロバイダーに積極的に貢献します。

- AWS プロバイダーリポジトリで適切に文書化された問題を開き、発生したバグや欠落している機能を詳しく説明します。再現可能なステップを提供します。
- 拡張機能をリクエストして投票し、新しい のサービスを管理するプロバイダーの機能を拡張します AWS 。
- プロバイダーの欠陥または機能強化の提案された修正を提供するときに発行されたプルリクエストを参照します。関連する問題へのリンク。
- コーディング規則、テスト標準、およびドキュメントについては、リポジトリの投稿ガイドラインに従ってください。

使用するプロバイダーにフィードバックすることで、ロードマップに直接入力し、すべてのユーザーの品質と機能を向上させることができます。

コミュニティモジュールのベストプラクティス

モジュールを効果的に使用することは、複雑な Terraform 設定を管理し、再利用を促進する上で重要です。このセクションでは、コミュニティモジュール、依存関係、ソース、抽象化、貢献に関するベストプラクティスについて説明します。

ベストプラクティス：

- コミュニティモジュールの検出
- 依存関係を理解する
- 信頼できるソースを使用する
- コミュニティモジュールへの貢献

コミュニティモジュールの検出

Terraform Registry 、、およびその他のソースでGitHub、新しい AWS モジュールを構築する前にユースケースを解決する可能性のある既存のモジュールを検索します。最近の更新があり、アクティブにメンテナンスされている一般的なオプションを探します。

カスタマイズに変数を使用する

コミュニティモジュールを使用する場合は、ソースコードを強制または直接変更するのではなく、変数を通じて入力を渡します。モジュールの内部を変更する代わりに、必要に応じてデフォルトを上書きします。

フォークは、より広範なコミュニティに役立つように、元のモジュールに修正や機能を提供することに限定する必要があります。

依存関係を理解する

モジュールを使用する前に、ソースコードとドキュメントを確認して依存関係を特定します。

- 必要なプロバイダー： モジュールが必要とする AWS、Kubernetes、またはその他のプロバイダーのバージョンを書き留めます。
- ネストされたモジュール：カスケード依存関係を導入する内部で使用されている他のモジュールを確認します。

- 外部データソース: モジュールが依存する APIsカスタムプラグイン、またはインフラストラクチャの依存関係を書き留めます。

直接依存関係と間接依存関係の完全なツリーをマッピングすることで、 モジュールを使用する際の予期しない事態を回避できます。

信頼できるソースを使用する

未検証または未知のパブリッシャーから Terraform モジュールを調達すると、重大なリスクが生じます。信頼できるソースからのモジュールのみを使用してください。

- AWS や HashiCorp パートナーなどの検証済み作成者によって公開される Terraform Registry の認定モジュールを優先します。
- カスタムモジュールの場合は、モジュールが自分の組織からのものである場合でも、パブリッシャーの履歴、サポートレベル、使用状況の評価を確認します。

不明なソースや未検証のソースからモジュールを許可しないことで、コードに脆弱性やメンテナンスの問題が挿入されるリスクを軽減できます。

の通知のサブスクライブ

信頼できるパブリッシャーからの新しいモジュールリリースの通知をサブスクライブします。

- GitHub モジュールリポジトリを監視して、モジュールの新しいバージョンに関するアラートを取得します。
- パブリッシャーブログと変更ログの更新をモニタリングします。
- 更新を暗黙的にプルするのではなく、検証済みで評価の高いソースから新しいバージョンに関する事前通知を取得します。

信頼できるソースからのみモジュールを消費し、変更をモニタリングすることで、安定性とセキュリティが確保されます。ベッティングモジュールは、サプライチェーンのリスクを最小限に抑えながら生産性を向上させます。

コミュニティモジュールへの貢献

でホストされているコミュニティモジュールの修正と機能強化を送信します GitHub。

- モジュールでプルリクエストを開き、使用時に発生した欠陥や制限に対処します。
- 問題を作成して、新しいベストプラクティス設定を既存の OSS モジュールに追加するようリクエストします。

コミュニティモジュールへの貢献により、すべての Terraform 実務者にとって再利用可能な体系的なパターンが強化されます。

よくある質問

Q: AWS プロバイダーに注目する理由は何ですか？

A. AWS プロバイダーは、Terraform でインフラストラクチャをプロビジョニングするために最も広く使用され、複雑なプロバイダーの 1 つです。これらのベストプラクティスに従うことで、ユーザーは AWS 環境のプロバイダーの使用を最適化できます。

Q: Terraform は初めてです。このガイドを使用できますか？

A. このガイドは、Terraform を初めて使用する場合や、スキルをレベルアップしたいと考えている上級者を対象としています。このプラクティスにより、学習のあらゆる段階でユーザーのワークフローが向上します。

Q: 主なベストプラクティスにはどのようなものがありますか？

A. 主なベストプラクティスには、アクセスキー での IAM ロールの使用、バージョン の固定、自動テスト の組み込み、リモート状態ロック 、認証情報ローテーション 、プロバイダー への貢献、コードベース の論理的な整理などがあります。

Q: Terraform の詳細については、どこで確認できますか？

A. リソースセクションには、 HashiCorp Terraform の公式ドキュメントとコミュニティフォーラムへのリンクが含まれています。リンクを使用して、高度な Terraform ワークフローの詳細を確認してください。

次のステップ

このガイドを読んだ後の潜在的な次のステップは次のとおりです。

- 既存の Terraform コードベースがある場合は、設定を確認し、このガイドに記載されている推奨事項に基づいて改善可能な領域を特定します。例えば、リモートバックエンドの実装、モジュールへのコードの分離、バージョンピンニングの使用などのベストプラクティスを確認し、設定で検証します。
- 既存の Terraform コードベースがない場合は、新しい設定を構築するときに以下のベストプラクティスを使用してください。状態管理、認証、コード構造などについては、最初からアドバイスに従ってください。
- このガイドで参照されている HashiCorp コミュニティモジュールの一部を使用して、アーキテクチャパターンが単純化されているかどうかを確認します。モジュールはより高いレベルの抽象化を可能にするため、一般的なリソースを書き直す必要はありません。
- リンティング、セキュリティスキャン、ポリシーチェック、自動テストツールを有効にして、セキュリティ、コンプライアンス、コード品質に関するベストプラクティスを強化します。TFLint 、tfsec、Checkov などのツールが役立ちます。
- 最新の AWS プロバイダードキュメントを確認して、Terraform の使用を最適化するのに役立つ新しいリソースや機能があるかどうかを確認します。 AWS プロバイダーの新しいバージョンを最新の状態に保つ。
- その他のガイダンスについては、 HashiCorp ウェブサイトの「Terraform ドキュメント」、「ベストプラクティスガイド」、および「スタイルガイド」を参照してください。

リソース

リファレンス

以下のリンクは、Terraform AWS プロバイダー用の追加の資料と、 での Terraform for IaC の使用を示しています AWS。

- Terraform AWS プロバイダー (HashiCorp ドキュメント)
- AWS サービスの Terraform モジュール (Terraform Registry)
- AWS と HashiCorp パートナーシップ (HashiCorp ブログ記事)
- AWS 「プロバイダーによる動的認証情報」(HCP Terraform ドキュメント)
- DynamoDB ステートロック (Terraform ドキュメント)
- 「Enforce Policy with Sentinel」(Terraform ドキュメント)

ツール

以下のツールは、このベストプラクティスガイドで推奨されているように AWS、 での Terraform 設定のコード品質と自動化を向上させるのに役立ちます。

コード品質：

- Checkov : デプロイ前に Terraform コードをスキャンして設定ミスを特定します。
- TFLint: 考えられるエラー、廃止された構文、未使用の宣言を識別します。この linter では、 AWS ベストプラクティスと命名規則を適用することもできます。
- terraform-docs : さまざまな出力形式で Terraform モジュールからドキュメントを生成します。

自動化ツール：

- HCP Terraform : ポリシーチェックと承認ゲートを使用して、チームが Terraform ワークフローをバージョン化、コラボレーション、構築するのに役立ちます。
- Atlantis : コード変更を検証するためのオープンソースの Terraform プルリクエスト自動化ツール。

- CDK for Terraform : HashiCorp 設定言語 (HCL) の代わりに TypeScript、Python、Java、C#、Go などの使い慣れた言語を使用して、Terraform インフラストラクチャをコードとして定義、プロビジョニング、テストできるフレームワーク。

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、RSS フィード をサブスクライブできます。

変更	説明	日付
リモート状態ロックの更新	Amazon S3 ネイティブ状態ロックを反映するようにバックエンドのベストプラクティスセクションを更新しました。これは現在推奨されるアプローチです。	2025 年 8 月 26 日
初版発行	—	2024 年 5 月 28 日

AWS 規範ガイダンスの用語集

以下は、 AWS 規範ガイダンスによって提供される戦略、ガイド、パターンで一般的に使用される用語です。エントリを提案するには、用語集の最後のフィードバックの提供リンクを使用します。

数字

7 Rs

アプリケーションをクラウドに移行するための 7 つの一般的な移行戦略。これらの戦略は、ガートナーが 2011 年に特定した 5 Rs に基づいて構築され、以下で構成されています。

- リファクタリング/アーキテクチャの再設計 — クラウドネイティブ特徴を最大限に活用して、俊敏性、パフォーマンス、スケーラビリティを向上させ、アプリケーションを移動させ、アーキテクチャを変更します。これには、通常、オペレーティングシステムとデータベースの移植が含まれます。例: オンプレミスの Oracle データベースを Amazon Aurora PostgreSQL 互換エディションに移行する。
- リプラットフォーム (リフトアンドリシェイプ) — アプリケーションをクラウドに移行し、クラウド機能を活用するための最適化レベルを導入します。例: お客様のオンプレミスの Oracle データベースを AWS クラウドの Oracle 用の Amazon Relational Database Service (Amazon RDS) に移行する。
- 再購入 (ドロップアンドショップ) — 通常、従来のライセンスから SaaS モデルに移行して、別の製品に切り替えます。例: 顧客関係管理 (CRM) システムを Salesforce.com に移行する。
- リホスト (リフトアンドシフト) — クラウド機能を活用するための変更を加えずに、アプリケーションをクラウドに移行します。例: お客様のオンプレミスの Oracle データベースを AWS クラウドの EC2 インスタンス上の Oracle に移行する。
- 再配置 (ハイパーバイザーレベルのリフトアンドシフト) – 新しいハードウェアを購入したり、アプリケーションを書き換えたり、既存の運用を変更したりすることなく、インフラストラクチャをクラウドに移行できます。オンプレミスプラットフォームから同じプラットフォームのクラウドサービスにサーバーを移行します。例： Microsoft Hyper-Vアプリケーションを に移行します AWS。
- 保持 (再アクセス) — アプリケーションをお客様のソース環境で保持します。これには、主要なリファクタリングを必要とするアプリケーションや、お客様がその作業を後日まで延期したいアプリケーション、およびそれらを移行するためのビジネス上の正当性がないため、お客様が保持するレガシーアプリケーションなどがあります。
- 廃止 — お客様のソース環境で不要になったアプリケーションを停止または削除します。

A

ABAC

「属性ベースのアクセス制御」をご覧ください。

抽象化されたサービス

「マネージドユーザー」をご覧ください。

ACID

「原子性、一貫性、分離性、耐久性 (ACID)」をご覧ください。

アクティブ/アクティブ移行

(双方向レプリケーションツールまたは二重書き込み操作を使用して) ソースデータベースとターゲットデータベースを同期させ、移行中に両方のデータベースが接続アプリケーションからのトランザクションを処理するデータベース移行方法。この方法では、1 回限りのカットオーバーの必要がなく、管理された小規模なバッチで移行できます。アクティブ/パッシブ移行よりも柔軟な方法ですが、さらに多くの作業が必要となります。

アクティブ/パッシブ移行

ソースデータベースとターゲットデータベースを同期させながら、データがターゲットデータベースにレプリケートされている間、接続しているアプリケーションからのトランザクションをソースデータベースのみで処理するデータベース移行方法。移行中、ターゲットデータベースはトランザクションを受け付けません。

集計関数

複数行に処理を行い、グループ全体を対象に単一の戻り値を計算する SQL 関数。集計関数の例としては、SUM や MAX などがあります。

AI

「人工知能」をご覧ください。

AIOps

「AI オペレーション」をご覧ください。

匿名化

データセット内の個人情報を完全に削除するプロセス。匿名化は個人のプライバシー保護に役立ちます。匿名化されたデータは、もはや個人データとは見なされません。

アンチパターン

繰り返し起こる問題に対して頻繁に用いられる解決策で、その解決策が逆効果であったり、効果がなかったり、代替案よりも効果が低かったりするもの。

アプリケーション制御

マルウェアからシステムを保護するために、承認されたアプリケーションのみを使用できるようにするセキュリティアプローチ。

アプリケーションポートフォリオ

アプリケーションの構築と維持にかかるコスト、およびそのビジネス価値を含む、組織が使用する各アプリケーションに関する詳細情報の集まり。この情報は、ポートフォリオの検出と分析プロセス の重要な要素であり、移行、モダナイズ、最適化するアプリケーションを特定し、優先順位を付けるのに役立ちます。

人工知能 (AI)

コンピューティングテクノロジーを使用し、学習、問題の解決、パターンの認識など、通常は人間に関連づけられる認知機能の実行に特化したコンピュータサイエンスの分野。詳細については、「人工知能 (AI) とは何ですか?」をご覧ください。

AI オペレーション (AIOps)

機械学習技術を使用して運用上の問題を解決し、運用上のインシデントと人の介入を減らし、サービス品質を向上させるプロセス。 AWS 移行戦略での AIOps の使用方法については、オペレーション統合ガイド を参照してください。

非対称暗号化

暗号化用のパブリックキーと復号用のプライベートキーから成る 1 組のキーを使用した、暗号化のアルゴリズム。パブリックキーは復号には使用されないため共有しても問題ありませんが、プライベートキーの利用は厳しく制限する必要があります。

原子性、一貫性、分離性、耐久性 (ACID)

エラー、停電、その他の問題が発生した場合でも、データベースのデータ有効性と運用上の信頼性を保証する一連のソフトウェアプロパティ。

属性ベースのアクセス制御 (ABAC)

部署、役職、チーム名など、ユーザーの属性に基づいてアクセス許可をきめ細かく設定する方法。詳細については、 AWS Identity and Access Management (IAM) ドキュメントの「 の ABAC AWS」を参照してください。

信頼できるデータソース

最も信頼性のある情報源とされるデータのプライマリーバージョンを保存する場所。匿名化、編集、仮名化など、データを処理または変更する目的で、信頼できるデータソースから他の場所にデータをコピーすることができます。

アベイラビリティーゾーン (AZ)

他のアベイラビリティーゾーンの障害から AWS リージョン 隔離され、同じリージョン内の他のアベイラビリティーゾーンへの低コストで低レイテンシーのネットワーク接続を提供する 内の別の場所。

AWS クラウド導入フレームワーク (AWS CAF)

組織がクラウドへの移行を成功させるための効率的で効果的な計画を立て AWS るための、 のガイドラインとベストプラクティスのフレームワークです。 AWS CAF は、ビジネス、人材、ガバナンス、プラットフォーム、セキュリティ、運用という 6 つの重点分野にガイダンスを整理しています。ビジネス、人材、ガバナンスの観点では、ビジネススキルとプロセスに重点を置き、プラットフォーム、セキュリティ、オペレーションの視点は技術的なスキルとプロセスに焦点を当てています。例えば、人材の観点では、人事 (HR)、人材派遣機能、および人材管理を扱うステークホルダーを対象としています。この観点から、 AWS CAF は、クラウド導入を成功させるための組織の準備に役立つ人材開発、トレーニング、コミュニケーションに関するガイダンスを提供します。詳細については、AWS CAF ウェブサイトと AWS CAF のホワイトペーパー を参照してください。

AWS ワークロード認定フレームワーク (AWS WQF)

データベース移行ワークロードを評価し、移行戦略を推奨し、作業見積もりを提供するツール。AWS WQF は AWS Schema Conversion Tool (AWS SCT) に含まれています。データベーススキーマとコードオブジェクト、アプリケーションコード、依存関係、およびパフォーマンス特性を分析し、評価レポートを提供します。

B

不正なボット

個人や組織に混乱や損害を与えることを目的としたボット。

BCP

「ビジネス継続性計画 (BCP)」をご覧ください。

動作グラフ

リソースの動作とインタラクションを経時的に示した、一元的なインタラクティブビュー。Amazon Detective の動作グラフを使用すると、失敗したログオンの試行、不審な API 呼び出し、その他同様のアクションを調べることができます。詳細については、Detective ドキュメントの「動作グラフのデータ」を参照してください。

ビッグエンディアンシステム

最上位バイトを最初に格納するシステム。「エンディアン性」もご覧ください。

二項分類

バイナリ結果 (2 つの可能なクラスのうちの 1 つ) を予測するプロセス。例えば、お客様の機械学習モデルで「この E メールはスパムですか、それともスパムではありませんか」などの問題を予測する必要があるかもしれません。または「この製品は書籍ですか、車ですか」などの問題を予測する必要があるかもしれません。

ブルームフィルター

要素がセットのメンバーであるかどうかをテストするために使用される、確率的でメモリ効率の高いデータ構造。

ブルー/グリーンデプロイ

それぞれが独立しているが、同一の環境を 2 つ作成するデプロイ戦略。現在のアプリケーションバージョンを 1 つの環境 (ブルー) で実行し、新しいアプリケーションバージョンを別の環境 (グリーン) で実行します。この戦略は、最小限の影響で迅速にロールバックするのに役立ちます。

ボット

インターネット経由で自動タスクを実行し、人間のアクティビティややり取りをシミュレートするソフトウェアアプリケーション。インターネット上の情報のインデックスを作成するウェブクローラーなど、一部のボットは有用または有益です。悪質なボットと呼ばれる他のボットの中には、個人や組織を混乱させたり、損害を与えたりすることを意図したものもあります。

ボットネット

マルウェアに感染しており、ボットハーダーまたはボットオペレーターと呼ばれる単一の当事者によって制御されているボットのネットワーク。ボットネットは、ボットとその影響力を拡大する仕組みとして、非常によく知られています。

ブランチ

コードリポジトリに含まれる領域。リポジトリに最初に作成するブランチは、メインブランチといいます。既存のブランチから新しいブランチを作成し、その新しいブランチで機能を開発した

り、バグを修正したりできます。機能を構築するために作成するブランチは、通常、機能ブランチと呼ばれます。機能をリリースする準備ができたら、機能ブランチをメインブランチに統合します。詳細については、「ブランチの概要」(GitHub ドキュメント) を参照してください。

ブレークグラスアクセス

例外的な状況では、承認されたプロセスを通じて、ユーザーが AWS アカウント 通常アクセス許可を持たない にすばやくアクセスできるようにします。詳細については、 AWS Well-Architected ガイダンスの「ブレークグラス手順の実装」インジケータを参照してください。

ブラウンフィールド戦略

環境の既存インフラストラクチャ。システムアーキテクチャにブラウンフィールド戦略を導入する場合、現在のシステムとインフラストラクチャの制約に基づいてアーキテクチャを設計します。既存のインフラストラクチャを拡張している場合は、ブラウンフィールド戦略とグリーンフィールド戦略を融合させることもできます。

バッファキャッシュ

アクセス頻度が最も高いデータが保存されるメモリ領域。

ビジネス能力

価値を生み出すためにビジネスが行うこと (営業、カスタマーサービス、マーケティングなど)。マイクロサービスのアーキテクチャと開発の決定は、ビジネス能力によって推進できます。詳細については、AWSでのコンテナ化されたマイクロサービスの実行ホワイトペーパーの「ビジネス機能を中心に組織化」セクションを参照してください。

ビジネス継続性計画 (BCP)

大規模移行など、中断を伴うイベントが運用に与える潜在的な影響に対処し、ビジネスを迅速に再開できるようにする計画。

C

CAF

「AWS クラウド導入フレームワーク」を参照してください

カナリアデプロイ

エンドユーザーへのバージョンリリースを、時間をかけて段階的に行うこと。確信が持てたら新規バージョンをデプロイして、現在のバージョン全体を置き換えます。

CCoE

「Cloud Center of Excellence」を参照してください。

CDC

「変更データキャプチャ」を参照してください。

変更データキャプチャ (CDC)

データソース (データベーステーブルなど) の変更を追跡し、その変更に関するメタデータを記録するプロセス。CDC は、ターゲットシステムでの変更を監査またはレプリケートして同期を維持するなど、さまざまな目的に使用できます。

カオスエンジニアリング

障害や破壊的なイベントを意図的に導入して、システムの耐障害性をテストすること。AWS Fault Injection Service (AWS FIS) を使用して、 AWS ワークロードにストレスを与え、その応答を評価する実験を実行できます。

CI/CD

「継続的インテグレーションと継続的デリバリー」を参照してください。

分類

予測を生成するのに役立つ分類プロセス。分類問題の機械学習モデルは、離散値を予測します。離散値は、常に互いに区別されます。例えば、モデルがイメージ内に車があるかどうかを評価する必要がある場合があります。

クライアント側の暗号化

ターゲットがデータ AWS のサービス を受信する前のローカルでのデータの暗号化。

Cloud Center of Excellence (CCoE)

クラウドのベストプラクティスの作成、リソースの移動、移行のタイムラインの確立、大規模変革を通じて組織をリードするなど、組織全体のクラウド導入の取り組みを推進する学際的なチーム。詳細については、 AWS クラウド エンタープライズ戦略ブログの CCoE 投稿を参照してください。

クラウドコンピューティング

リモートデータストレージと IoT デバイス管理に通常使用されるクラウドテクノロジー。クラウドコンピューティングは、一般的に、エッジコンピューティングに接続されています。

クラウド運用モデル

IT 組織において、1 つ以上のクラウド環境を構築、成熟、最適化するために使用される運用モデル。詳細については、「クラウド運用モデルの構築」を参照してください。

導入のクラウドステージ

組織が、 AWS クラウドへの移行時に通常実行する 4 つの段階。

- プロジェクト — 概念実証と学習を目的として、クラウド関連のプロジェクトをいくつか実行する
- 基礎固め — お客様のクラウドの導入を拡大するための基礎的な投資 (ランディングゾーンの作成、CCoE の定義、運用モデルの確立など)
- 移行 — 個々のアプリケーションの移行
- 再発明 — 製品とサービスの最適化、クラウドでのイノベーション

これらのステージは、 AWS クラウド エンタープライズ戦略ブログのブログ記事「クラウドファーストへのジャーニー」と「導入のステージ」で Stephen Orban によって定義されました。移行戦略との関連性については、 AWS 「移行準備ガイド」を参照してください。

CMDB

「構成管理データベース (CMDB)」を参照してください。

コードリポジトリ

ソースコードやその他の資産 (ドキュメント、サンプル、スクリプトなど) が保存され、バージョン管理プロセスを通じて更新される場所。一般的なクラウドリポジトリには、GitHub や Bitbucket Cloud があります。コードの各バージョンはブランチと呼ばれます。マイクロサービスの構造では、各リポジトリは 1 つの機能専用です。1 つの CI/CD パイプラインで複数のリポジトリを使用できます。

コールドキャッシュ

空である、または、かなり空きがある、もしくは、古いデータや無関係なデータが含まれているバッファキャッシュ。データベースインスタンスはメインメモリまたはディスクから読み取る必要があり、バッファキャッシュから読み取るよりも時間がかかるため、パフォーマンスに影響します。

コールドデータ

めったにアクセスされず、通常は過去のデータです。この種類のデータをクエリする場合、通常は低速なクエリでも問題ありません。このデータを低パフォーマンスで安価なストレージ階層またはクラスに移動すると、コストを削減することができます。

コンピュータビジョン (CV)

機械学習を使用してデジタルイメージやビデオといった、ビジュアル形式の情報を分析および抽出する AI の分野。例えば、Amazon SageMaker AI では、CV 用の画像処理アルゴリズムを利用できます。

設定ドリフト

ワークロードにおいて、設定が想定した状態から変化すること。これによって、ワークロードが非準拠になる可能性がありますが、この状態は、徐々に生じ、意図的なものではありません。

構成管理データベース (CMDB)

データベースとその IT 環境 (ハードウェアとソフトウェアの両方のコンポーネントとその設定を含む) に関する情報を保存、管理するリポジトリ。通常、CMDB のデータは、移行のポートフォリオの検出と分析の段階で使用します。

コンフォーマンスパック

コンプライアンスチェックとセキュリティチェックをカスタマイズするためにアセンブルできる AWS Config ルールと修復アクションのコレクション。YAML テンプレートを使用して、コンフォーマンスパックを AWS アカウント および リージョンの単一のエンティティとしてデプロイすることも、組織全体にデプロイすることもできます。詳細については、 AWS Config ドキュメントの「コンフォーマンスパック」を参照してください。

継続的インテグレーションと継続的デリバリー (CI/CD)

ソフトウェアリリースプロセスのソース、ビルド、テスト、ステージング、本番の各ステージを自動化するプロセス。CI/CD は一般的にパイプラインと呼ばれます。プロセスの自動化、生産性の向上、コード品質の向上、配信の加速化を可能にします。詳細については、「継続的デリバリーの利点」を参照してください。CD は継続的デプロイ (Continuous Deployment) の略語でもあります。詳細については「継続的デリバリーと継続的なデプロイ」を参照してください。

CV

「コンピュータビジョン」を参照してください。

D

保管中のデータ

ストレージ内にあるデータなど、常に自社のネットワーク内にあるデータ。

データ分類

ネットワーク内のデータを重要度と機密性に基づいて識別、分類するプロセス。データに適した保護および保持のコントロールを判断する際に役立つため、あらゆるサイバーセキュリティのリスク管理戦略において重要な要素です。データ分類は、 AWS Well-Architected フレームワークのセキュリティの柱のコンポーネントです。詳細については、「データ分類」を参照してください。

データドリフト

実稼働データと ML モデルのトレーニングに使用されたデータとの間に有意な差異が生じたり、入力データが時間の経過と共に有意に変化したりすることです。データドリフトは、ML モデル予測の全体的な品質、精度、公平性を低下させる可能性があります。

転送中のデータ

ネットワーク内 (ネットワークリソース間など) を活発に移動するデータ。

データメッシュ

非一元的で分散型のデータ所有権を持つとともに、一元的な管理およびガバナンスを行えるアーキテクチャフレームワーク。

データ最小化

厳密に必要なデータのみを収集し、処理するという原則。でデータ最小化を実践 AWS クラウドすることで、プライバシーリスク、コスト、分析のカーボンフットプリントを削減できます。

データ境界

AWS 環境内の一連の予防ガードレール。信頼された ID のみが、期待されるネットワークから信頼されたリソースにアクセスできるようにします。詳細については、「 でのデータ境界の構築 AWS」を参照してください。

データの前処理

raw データをお客様の機械学習モデルで簡単に解析できる形式に変換すること。データの前処理とは、特定の列または行を削除して、欠落している、矛盾している、または重複する値に対処することを意味します。

データ出所

データの生成、送信、保存の方法など、データのライフサイクル全体を通じてデータの出所と履歴を追跡するプロセス。

データ件名

データを収集、処理している個人。

データウェアハウス

分析などのビジネスインテリジェンスをサポートするデータ管理システム。データウェアハウスには、一般的に、大量の履歴データが含まれており、多くの場合、それらはクエリや分析に使用されます。

データベース定義言語 (DDL)

データベース内のテーブルやオブジェクトの構造を作成または変更するためのステートメントまたはコマンド。

データベース操作言語 (DML)

データベース内の情報を変更 (挿入、更新、削除) するためのステートメントまたはコマンド。

DDL

「データベース定義言語」を参照してください。

ディープアンサンブル

予測のために複数の深層学習モデルを組み合わせます。ディープアンサンブルを使用して、より正確な予測を取得したり、予測の不確実性を推定したりできます。

深層学習

人工ニューラルネットワークの複数層を使用して、入力データと対象のターゲット変数の間のマッピングを識別する機械学習サブフィールド。

多層防御

一連のセキュリティメカニズムとコントロールをコンピュータネットワーク全体に層状に重ねて、ネットワークとその内部にあるデータの機密性、整合性、可用性を保護する情報セキュリティの手法。この戦略を に採用するときは AWS、リソースの保護に役立つように、 AWS Organizations 構造の異なるレイヤーに複数のコントロールを追加します。たとえば、多層防御アプローチでは、多要素認証、ネットワークセグメンテーション、暗号化を組み合わせることができます。

委任管理者

では AWS Organizations、互換性のあるサービスが AWS メンバーアカウントを登録して組織のアカウントを管理し、そのサービスのアクセス許可を管理できます。このアカウントを、そのサービスの委任管理者と呼びます。詳細、および互換性のあるサービスの一覧は、 AWS

Organizations ドキュメントの「AWS Organizationsで使用できるサービス」を参照してください。

トラブルシューティング

アプリケーション、新機能、コードの修正をターゲットの環境で利用できるようにするプロセス。デプロイでは、コードベースに変更を施した後、アプリケーションの環境でそのコードベースを構築して実行します。

開発環境

「環境」を参照してください。

検出管理

イベントが発生したときに、検出、ログ記録、警告を行うように設計されたセキュリティコントロール。これらのコントロールは副次的な防衛手段であり、実行中の予防的コントロールをすり抜けたセキュリティイベントをユーザーに警告します。詳細については、「 AWSでのセキュリティコントロールの実装」の「検出的コントロール」を参照してください。

開発バリューストリームマッピング (DVSM)

ソフトウェア開発ライフサイクルのスピードと品質に悪影響を及ぼす制約を特定し、優先順位を付けるために使用されるプロセス。DVSM は、もともとリーンマニュファクチャリング・プラクティスのために設計されたバリューストリームマッピング・プロセスを拡張したものです。ソフトウェア開発プロセスを通じて価値を創造し、動かすために必要なステップとチームに焦点を当てています。

デジタルツイン

建物、工場、産業機器、生産ラインなど、現実世界のシステムを仮想的に表現したものです。デジタルツインは、予知保全、リモートモニタリング、生産最適化をサポートします。

ディメンションテーブル

スタースキーマにおいて、ファクトテーブルの定量データに関するデータ属性が含まれる小さいテーブル。ディメンションテーブルの属性は、通常、テキストフィールド、またはテキストのように扱える個別の数値で示されます。これらの属性は、一般的に、クエリの制約、フィルタリング、結果セットのラベル付けに使用されます。

ディザスタ

ワークロードまたはシステムが、導入されている主要な場所でのビジネス目標の達成を妨げるイベント。これらのイベントは、自然災害、技術的障害、または意図しない設定ミスやマルウェア攻撃などの人間の行動の結果である場合があります。

ディザスタリカバリ (DR)

ディザスタによるダウンタイムとデータ損失を最小限に抑えるための戦略とプロセス。詳細については、 AWS Well-Architected フレームワークの「 でのワークロードのディザスタリカバリ AWS: クラウドでのリカバリ」を参照してください。

DML

「データベース操作言語」を参照してください。

ドメイン駆動型設計

各コンポーネントが提供している変化を続けるドメイン、またはコアビジネス目標にコンポーネントを接続して、複雑なソフトウェアシステムを開発するアプローチ。この概念は、エリック・エヴァンスの著書、Domain-Driven Design: Tackling Complexity in the Heart of Software (ドメイン駆動設計:ソフトウェアの中心における複雑さへの取り組み) で紹介されています (ボストン: Addison-Wesley Professional、2003)。strangler fig パターンでドメイン駆動型設計を使用する方法の詳細については、「コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET (ASMX) ウェブサービスを段階的にモダナイズ」を参照してください。

DR

「ディザスタリカバリ」を参照してください。

ドリフト検出

ベースライン設定からの偏差を追跡します。たとえば、 AWS CloudFormation を使用してシステムリソースのドリフトを検出したり、 を使用して AWS Control Tower 、ガバナンス要件への準拠に影響するランディングゾーンの変更を検出したりできます。

DVSM

「開発バリューストリームマッピング」を参照してください。

E

EDA

「探索的データ分析」を参照してください。

EDI

「電子データ交換」を参照してください。

エッジコンピューティング

IoT ネットワークのエッジにあるスマートデバイスの計算能力を高めるテクノロジー。クラウドコンピューティングと比較すると、エッジコンピューティングは通信レイテンシーを短縮し、応答時間を改善できます。

電子データ交換 (EDI)

組織間で行う、ビジネスドキュメントの自動交換。詳細については、「電子データ交換とは」を参照してください。

暗号化

人間が読み取り可能なプレーンテキストデータを暗号文に変換するコンピューティング処理。

暗号化キー

暗号化アルゴリズムが生成した、ランダム化されたビットからなる暗号文字列。キーの長さは決まっておらず、各キーは予測できないように、一意になるように設計されています。

エンディアン

コンピュータメモリにバイトが格納される順序。ビッグエンディアンシステムでは、最上位バイトが最初に格納されます。リトルエンディアンシステムでは、最下位バイトが最初に格納されます。

エンドポイント

「サービスエンドポイント」を参照してください。

エンドポイントサービス

仮想プライベートクラウド (VPC) 内でホストして、他のユーザーと共有できるサービス。を使用してエンドポイントサービスを作成し AWS PrivateLink 、他の AWS アカウント または AWS Identity and Access Management (IAM) プリンシパルにアクセス許可を付与できます。これらのアカウントまたはプリンシパルは、インターフェイス VPC エンドポイントを作成することで、エンドポイントサービスにプライベートに接続できます。詳細については、Amazon Virtual Private Cloud (Amazon VPC) ドキュメントの「エンドポイントサービスを作成する」を参照してください。

エンタープライズリソースプランニング (ERP)

エンタープライズの主要なビジネスプロセス (会計、MES、プロジェクト管理など) を自動化および管理するシステム。

エンベロープ暗号化

暗号化キーを、別の暗号化キーを使用して暗号化するプロセス。詳細については、 AWS Key Management Service (AWS KMS) ドキュメントの「エンベロープ暗号化」を参照してください。

環境

実行中のアプリケーションのインスタンス。クラウドコンピューティングにおける一般的な環境の種類は以下のとおりです。

- 開発環境 — アプリケーションのメンテナンスを担当するコアチームのみが使用できる、実行中のアプリケーションのインスタンス。開発環境は、上位の環境に昇格させる変更をテストするときに使用します。このタイプの環境は、テスト環境と呼ばれることもあります。
- 下位環境 — 初期ビルドやテストに使用される環境など、アプリケーションのすべての開発環境。
- 本番環境 — エンドユーザーがアクセスできる、実行中のアプリケーションのインスタンス。CI/CD パイプラインでは、本番環境が最後のデプロイ環境になります。
- 上位環境 — コア開発チーム以外のユーザーがアクセスできるすべての環境。これには、本番環境、本番前環境、ユーザー承認テスト環境などが含まれます。

エピック

アジャイル方法論で、お客様の作業の整理と優先順位付けに役立つ機能カテゴリ。エピックでは、要件と実装タスクの概要についてハイレベルな説明を提供します。例えば、 AWS CAF セキュリティエピックには、ID とアクセスの管理、検出コントロール、インフラストラクチャセキュリティ、データ保護、インシデント対応が含まれます。 AWS 移行戦略のエピックの詳細については、プログラム実装ガイドを参照してください。

ERP

「エンタープライズリソース計画」を参照してください。

探索的データ分析 (EDA)

データセットを分析してその主な特性を理解するプロセス。お客様は、データを収集または集計してから、パターンの検出、異常の検出、および前提条件のチェックのための初期調査を実行します。EDA は、統計の概要を計算し、データの可視化を作成することによって実行されます。

F

ファクトテーブル

スタースキーマの中央にあるテーブル。ビジネスオペレーションに関する定量的データが保存されます。一般的に、ファクトテーブルは、2 種類の列で構成されます。1 つは測定値が含まれる列、もう 1 つはディメンションテーブルへの外部キーが含まれる列です。

フェイルファスト

開発ライフサイクルを短縮するために、頻繁かつ段階的にテストを行う哲学であり、アジャイルアプローチでは、この考え方がきわめて重要です。

障害分離境界

では AWS クラウド、障害の影響を制限し、ワークロードの耐障害性を高めるのに役立つアベイラビリティーゾーン AWS リージョン、コントロールプレーン、データプレーンなどの境界。詳細については、「AWS 障害分離境界」を参照してください。

機能ブランチ

「ブランチ」を参照してください。

特徴量

お客様が予測に使用する入力データ。例えば、製造コンテキストでは、特徴量は製造ラインから定期的にキャプチャされるイメージの可能性もあります。

特徴量重要度

モデルの予測に対する特徴量の重要性。これは通常、Shapley Additive Deskonations (SHAP) や積分勾配など、さまざまな手法で計算できる数値スコアで表されます。詳細については、「 を使用した機械学習モデルの解釈可能性 AWS」を参照してください。

機能変換

追加のソースによるデータのエンリッチ化、値のスケーリング、単一のデータフィールドからの複数の情報セットの抽出など、機械学習プロセスのデータを最適化すること。これにより、機械学習モデルはデータの恩恵を受けることができます。例えば、「2021-05-27 00:15:37」の日付を「2021 年」、「5 月」、「木」、「15」に分解すると、学習アルゴリズムがさまざまなデータコンポーネントに関連する微妙に異なるパターンを学習するのに役立ちます。

数ショットプロンプト

LLM に、タスクと望ましい出力を示す例を少数提示した後に、類似のタスクを実行させること。この手法は、プロンプトに記述された例 (ショット) からモデルが学習する「インコンテキスト学

習」の一種です。数ショットプロンプトは、特定のフォーマット、推論、専門知識が必要なタスクに効果的です。「ゼロショットプロンプト」も参照してください。

FGAC

「きめ細かなアクセス制御」を参照してください。

きめ細かなアクセス制御 (FGAC)

複数の条件を使用してアクセス要求を許可または拒否すること。

フラッシュカット移行

変更データのキャプチャによる継続的なデータ複製を利用して、段階的なアプローチではなく、可能な限り短時間でデータを移行するデータベース移行方法。目的はダウンタイムを最小限に抑えることです。

FM

「基盤モデル」を参照してください。

基盤モデル (FM)

大規模な深層学習ニューラルネットワークであり、一般化およびラベル付けされていないデータからなる大規模データセットでトレーニングされています。FM により、言語理解、テキストおよび画像生成、自然言語での会話といった、一般的な各種タスクを実行できます。詳細については、「基盤モデルとは何ですか?」を参照してください。

G

生成 AI

AI モデルのサブセット。大量のデータでトレーニングされており、シンプルなテキストプロンプトを使用して、画像、動画、テキスト、オーディオなどの新しいコンテンツやアーティファクトを作成できます。詳細については、「生成 AI とは何ですか?」を参照してください。

ジオブロッキング

「地理的制限」を参照してください。

地理的制限 (ジオブロッキング)

特定の国のユーザーがコンテンツ配信にアクセスできないようにするための、Amazon CloudFront のオプション。アクセスを許可する国と禁止する国は、許可リストまたは禁止リスト

を使って指定します。詳細については、CloudFront ドキュメントの「コンテンツの地理的ディストリビューションの制限」を参照してください。

Gitflow ワークフロー

下位環境と上位環境が、ソースコードリポジトリでそれぞれ異なるブランチを使用する方法。Gitflow ワークフローは古いと見なされている方法であり、トランクベースのワークフローは推奨されている新しい方法です。

ゴールデンイメージ

システムまたはソフトウェアのスナップショットであり、システムまたはソフトウェアの新規インスタンスをデプロイするテンプレートとして使用されます。製造の例で言えば、ゴールデンイメージを使用すると、複数のデバイスにソフトウェアをプロビジョニングして、デバイス製造オペレーションの速度、スケーラビリティ、生産性を向上させることができます。

グリーンフィールド戦略

新しい環境に既存のインフラストラクチャが存在しないこと。システムアーキテクチャにグリーンフィールド戦略を導入する場合、既存のインフラストラクチャ (別名ブラウンフィールド) との互換性の制約を受けることなく、あらゆる新しいテクノロジーを選択できます。既存のインフラストラクチャを拡張している場合は、ブラウンフィールド戦略とグリーンフィールド戦略を融合させることもできます。

ガードレール

組織単位 (OU) 全般のリソース、ポリシー、コンプライアンスを管理するのに役立つ概略的なルール。予防ガードレールは、コンプライアンス基準に一致するようにポリシーを実施します。これらは、サービスコントロールポリシーと IAM アクセス許可の境界を使用して実装されます。検出ガードレールは、ポリシー違反やコンプライアンス上の問題を検出し、修復のためのアラートを発信します。これらは AWS Config、、 AWS Security Hub CSPM、Amazon GuardDuty、、 AWS Trusted Advisor Amazon Inspector、およびカスタム AWS Lambda チェックを使用して実装されます。

H

HA

「高可用性」を参照してください。

異種混在データベースの移行

別のデータベースエンジンを使用するターゲットデータベースへお客様の出典データベースの移行 (例えば、Oracle から Amazon Aurora)。異種間移行は通常、アーキテクチャの再設計作業の一部であり、スキーマの変換は複雑なタスクになる可能性があります。AWS は、スキーマの変換に役立つ AWS SCTを提供します。

高可用性 (HA)

課題や災害が発生した場合に、介入なしにワークロードを継続的に運用できること。HA システムは、自動的にフェイルオーバーし、一貫して高品質のパフォーマンスを提供し、パフォーマンスへの影響を最小限に抑えながらさまざまな負荷や障害を処理するように設計されています。

ヒストリアンのモダナイゼーション

製造業のニーズによりよく応えるために、オペレーションテクノロジー (OT) システムをモダナイズし、アップグレードするためのアプローチ。ヒストリアンは、工場内のさまざまなソースからデータを収集して保存するために使用されるデータベースの一種です。

ホールドアウトデータ

機械学習モデルのトレーニング用データセットから保留される、ラベル付き履歴データの一部。ホールドアウトデータを使用すると、モデル予測をホールドアウトデータと比較して、モデルのパフォーマンスを評価できます。

同種データベースの移行

お客様の出典データベースを、同じデータベースエンジンを共有するターゲットデータベース (Microsoft SQL Server から Amazon RDS for SQL Server など) に移行する。同種間移行は、通常、リホストまたはリプラットフォーム化の作業の一部です。ネイティブデータベースユーティリティを使用して、スキーマを移行できます。

ホットデータ

リアルタイムデータや最近の翻訳データなど、頻繁にアクセスされるデータ。通常、このデータには高速なクエリ応答を提供する高性能なストレージ階層またはクラスが必要です。

ホットフィックス

本番環境の重大な問題を修正するために緊急で配布されるプログラム。緊急性が高いため、通常の DevOps のリリースワークフローからは外れた形で実施されます。

ハイパーケア期間

カットオーバー直後、移行したアプリケーションを移行チームがクラウドで管理、監視して問題に対処する期間。通常、この期間は 1～4 日です。ハイパーケア期間が終了すると、アプリケーションに対する責任は一般的に移行チームからクラウドオペレーションチームに移ります。

I

IaC

「Infrastructure as Code」を参照してください。

ID ベースのポリシー

AWS クラウド 環境内のアクセス許可を定義する 1 つ以上の IAM プリンシパルにアタッチされたポリシー。

アイドル状態のアプリケーション

90 日間の平均的な CPU およびメモリ使用率が 5～20% のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するか、オンプレミスに保持するのが一般的です。

IIoT

「インダストリアル IoT」を参照してください。

イミュータブルインフラストラクチャ

既存インフラストラクチャの更新、パッチ適用、変更などを行わずに、本番環境ワークロードに使用する新規インフラストラクチャをデプロイするモデル。本質的に、イミュータブルインフラストラクチャは、ミュータブルインフラストラクチャよりも一貫性、信頼性、予測性に優れています。詳細については、 AWS Well-Architected フレームワークにある「イミュータブルインフラストラクチャを使用してデプロイする」のベストプラクティスを参照してください。

インバウンド (受信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーションの外部からネットワーク接続を受け入れ、検査し、ルーティングする VPC。AWS Security Reference Architecture では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

増分移行

アプリケーションを 1 回ですべてカットオーバーするのではなく、小さい要素に分けて移行するカットオーバー戦略。例えば、最初は少数のマイクロサービスまたはユーザーのみを新しいシステムに移行する場合があります。すべてが正常に機能することを確認できたら、残りのマイクロサービスやユーザーを段階的に移行し、レガシーシステムを廃止できるようにします。この戦略により、大規模な移行に伴うリスクが軽減されます。

インダストリー 4.0

2016 年に Klaus Schwab 氏が提唱した用語で、接続、リアルタイムデータ、オートメーション、分析、AI/ML の進歩による、ビジネスプロセスのモダナイズを意味します。

インフラストラクチャ

アプリケーションの環境に含まれるすべてのリソースとアセット。

Infrastructure as Code (IaC)

アプリケーションのインフラストラクチャを一連の設定ファイルを使用してプロビジョニングし、管理するプロセス。IaC は、新しい環境を再現可能で信頼性が高く、一貫性のあるものにするため、インフラストラクチャを一元的に管理し、リソースを標準化し、スケールを迅速に行えるように設計されています。

インダストリアル IoT (IIoT)

製造、エネルギー、自動車、ヘルスケア、ライフサイエンス、農業などの産業部門におけるインターネットに接続されたセンサーやデバイスの使用。詳細については、「インダストリアル IoT (IIoT) デジタルトランスフォーメーション戦略の構築」を参照してください。

インスペクション VPC

AWS マルチアカウントアーキテクチャでは、VPC (同一または異なる 内 AWS リージョン)、インターネット、オンプレミスネットワーク間のネットワークトラフィックの検査を管理する一元化された VPCs。AWS Security Reference Architecture では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

IoT

インターネットまたはローカル通信ネットワークを介して他のデバイスやシステムと通信する、センサーまたはプロセッサが組み込まれた接続済み物理オブジェクトのネットワーク。詳細については、「IoT とは」を参照してください。

解釈可能性

機械学習モデルの特性で、モデルの予測がその入力にどのように依存するかを人間が理解できる度合いを表します。詳細については、「 を使用した機械学習モデルの解釈可能性 AWS」を参照してください。

IoT

「IoT」を参照してください。

IT 情報ライブラリ (ITIL)

IT サービスを提供し、これらのサービスをビジネス要件に合わせるための一連のベストプラクティス。ITIL は ITSM の基盤を提供します。

IT サービス管理 (ITSM)

組織の IT サービスの設計、実装、管理、およびサポートに関連する活動。クラウドオペレーションと ITSM ツールの統合については、オペレーション統合ガイドを参照してください。

ITIL

「IT 情報ライブラリ」を参照してください。

ITSM

「IT サービス管理」を参照してください。

L

ラベルベースアクセス制御 (LBAC)

強制アクセス制御 (MAC) の実装で、ユーザーとデータ自体にそれぞれセキュリティラベル値が明示的に割り当てられます。ユーザーセキュリティラベルとデータセキュリティラベルが交差する部分によって、ユーザーに表示される行と列が決まります。

ランディングゾーン

ランディングゾーンは、スケーラブルで安全な、適切に設計されたマルチアカウント AWS 環境です。これは、組織がセキュリティおよびインフラストラクチャ環境に自信を持ってワークロードとアプリケーションを迅速に起動してデプロイできる出発点です。ランディングゾーンの詳細については、「安全でスケーラブルなマルチアカウント AWS 環境のセットアップ」を参照してください。

大規模言語モデル (LLM)

大量のデータで事前トレーニングされた深層学習 AI モデル。LLM では、質問への回答、ドキュメントの要約、他言語へのテキスト翻訳、文を完成させるなど、さまざまなタスクを実行できます。詳細については、「大規模言語モデル (LLM) とは何ですか?」を参照してください。

大規模な移行

300 台以上のサーバの移行。

LBAC

「ラベルベースアクセス制御」を参照してください。

最小特権

タスクの実行には必要最低限の権限を付与するという、セキュリティのベストプラクティス。詳細については、IAM ドキュメントの「最小特権アクセス許可を適用する」を参照してください。

リフトアンドシフト

「7 Rs」を参照してください。

リトルエンディアンシステム

最下位バイトを最初に格納するシステム。「エンディアン性」もご覧ください。

LLM

「大規模言語モデル」を参照してください。

下位環境

「環境」を参照してください。

M

機械学習 (ML)

パターン認識と学習にアルゴリズムと手法を使用する人工知能の一種。ML は、モノのインターネット (IoT) データなどの記録されたデータを分析して学習し、パターンに基づく統計モデルを生成します。詳細については、「機械学習」を参照してください。

メインブランチ

「ブランチ」を参照してください。

マルウェア

コンピュータのセキュリティやプライバシーを侵害するように設計されたソフトウェア。マルウェアは、コンピュータシステムの中断、機密情報の漏洩、不正アクセスを招く可能性があります。マルウェアの例には、ウイルス、ワーム、ランサムウェア、トロイの木馬、スパイウェア、キーロガーなどがあります。

マネージドサービス

AWS のサービス はインフラストラクチャレイヤー、オペレーティングシステム、プラットフォーム AWS を運用し、エンドポイントにアクセスしてデータを保存および取得します。マネージドサービスの例として、Amazon Simple Storage Service (Amazon S3) と Amazon DynamoDB が挙げられます。このサービスは、抽象化されたサービスとも呼ばれます。

製造実行システム (MES)

生産プロセスを追跡、モニタリング、文書化、制御するソフトウェアシステムであり、工場では、これによって、原材料から製品を完成させます。

MAP

「Migration Acceleration Program」を参照してください。

メカニズム

ツールを作成してその導入を推進し、導入結果を調べて調整を行うための包括的なプロセス。メカニズムとは、運用中にそれ自体を強化し改善するサイクルを意味します。詳細については、AWS 「 Well-Architected フレームワーク」の「メカニズムの構築」を参照してください。

メンバーアカウント

組織の一部である管理アカウント AWS アカウント 以外のすべて AWS Organizations。 アカウントが 組織のメンバーになることができるのは、一度に 1 つのみです。

MES

「製造実行システム」を参照してください。

Message Queuing Telemetry Transport (MQTT)

発行/サブスクライブのパターンに基づく、軽量のマシンツーマシン (M2M) 通信プロトコルであり、リソースに限りのある IoT デバイスに使用されます。

マイクロサービス

明確に定義された API を介して通信し、通常は小規模な自己完結型のチームが所有する、小規模で独立したサービスです。例えば、保険システムには、販売やマーケティングなどのビジネス

機能、または購買、請求、分析などのサブドメインにマッピングするマイクロサービスが含まれる場合があります。マイクロサービスの利点には、俊敏性、柔軟なスケーリング、容易なデプロイ、再利用可能なコード、回復力などがあります。詳細については、AWS「サーバーレスサービスを使用したマイクロサービスの統合」を参照してください。

マイクロサービスアーキテクチャ

各アプリケーションプロセスをマイクロサービスとして実行する独立したコンポーネントを使用してアプリケーションを構築するアプローチ。これらのマイクロサービスは、軽量 API を使用して、明確に定義されたインターフェイスを介して通信します。このアーキテクチャの各マイクロサービスは、アプリケーションの特定の機能に対する需要を満たすように更新、デプロイ、およびスケーリングできます。詳細については、「 でのマイクロサービスの実装 AWS」を参照してください。

Migration Acceleration Program (MAP)

組織がクラウドに移行するための強力な運用基盤を構築し、移行の初期コストを相殺するのに役立つコンサルティングサポート、トレーニング、サービスを提供する AWS プログラム。MAP には、組織的な方法でレガシー移行を実行するための移行方法論と、一般的な移行シナリオを自動化および高速化する一連のツールが含まれています。

大規模な移行

アプリケーションポートフォリオの大部分を次々にクラウドに移行し、各ウェーブでより多くのアプリケーションを高速に移動させるプロセス。この段階では、以前の段階から学んだベストプラクティスと教訓を使用して、移行ファクトリー チーム、ツール、プロセスのうち、オートメーションとアジャイルデリバリーによってワークロードの移行を合理化します。これは、AWS 移行戦略 の第 3 段階です。

移行ファクトリー

自動化された俊敏性のあるアプローチにより、ワークロードの移行を合理化する部門横断的なチーム。移行ファクトリーチームには、通常、運用、ビジネスアナリストおよび所有者、移行エンジニア、デベロッパー、およびスプリントで作業する DevOps プロフェッショナルが含まれます。エンタープライズアプリケーションポートフォリオの 20～50% は、ファクトリーのアプローチによって最適化できる反復パターンで構成されています。詳細については、このコンテンツセットの移行ファクトリーに関する解説とCloud Migration Factory ガイドを参照してください。

移行メタデータ

移行を完了するために必要なアプリケーションおよびサーバーに関する情報。移行パターンごとに、異なる一連の移行メタデータが必要です。移行メタデータの例としては、ターゲットサブネット、セキュリティグループ、 AWS アカウントなどがあります。

移行パターン

移行戦略、移行先、および使用する移行アプリケーションまたはサービスを詳述する、反復可能な移行タスク。例: AWS Application Migration Service を使用して Amazon EC2 への移行をリホストします。

Migration Portfolio Assessment (MPA)

オンラインツール。これによって、 AWS クラウドに移行するビジネスケースの検証に必要な情報を得られます。MPA は、詳細なポートフォリオ評価 (サーバーの適切なサイジング、価格設定、TCO 比較、移行コスト分析) および移行プラン (アプリケーションデータの分析とデータ収集、アプリケーションのグループ化、移行の優先順位付け、およびウェーブプランニング) を提供します。MPA ツール (ログインが必要) は、すべての AWS コンサルタントと APN パートナーコンサルタントが無料で利用できます。

移行準備状況評価 (MRA)

AWS CAF を使用して、組織のクラウド準備状況に関するインサイトを取得し、長所と短所を特定し、特定されたギャップを埋めるためのアクションプランを構築するプロセス。詳細については、移行準備状況ガイドを参照してください。MRA は、AWS 移行戦略の第一段階です。

移行戦略

ワークロードを AWS クラウドに移行するために使用するアプローチ。詳細については、この用語集の 7 Rs エントリと、「組織を動員して大規模な移行を加速する」を参照してください。

ML

「機械学習」を参照してください。

モダナイゼーション

古い (レガシーまたはモノリシック) アプリケーションとそのインフラストラクチャをクラウド内の俊敏で弾力性のある高可用性システムに変換して、コストを削減し、効率を高め、イノベーションを活用します。詳細については、「AWS クラウドでのアプリケーションのモダナイズ戦略」を参照してください。

モダナイゼーション準備状況評価

組織のアプリケーションのモダナイゼーションの準備状況を判断し、利点、リスク、依存関係を特定し、組織がこれらのアプリケーションの将来の状態をどの程度適切にサポートできるかを決定するのに役立つ評価。評価の結果として、ターゲットアーキテクチャのブループリント、モダナイゼーションプロセスの開発段階とマイルストーンを詳述したロードマップ、特定されたギャップに対処するためのアクションプランが得られます。詳細については、「AWS クラウドでのアプリケーションのモダナイゼーションの準備状況を評価する」を参照してください。

モノリシックアプリケーション (モノリス)

緊密に結合されたプロセスを持つ単一のサービスとして実行されるアプリケーション。モノリシックアプリケーションにはいくつかの欠点があります。1 つのアプリケーション機能エクスペリエンスの需要が急増する場合は、アーキテクチャ全体をスケーリングする必要があります。モノリシックアプリケーションの特徴を追加または改善することは、コードベースが大きくなると複雑になります。これらの問題に対処するには、マイクロサービスアーキテクチャを使用できます。詳細については、「モノリスをマイクロサービスに分解する」を参照してください。

MPA

「Migration Portfolio Assessment」を参照してください。

MQTT

「Message Queuing Telemetry Transport」を参照してください。

多クラス分類

複数のクラスの予測を生成するプロセス (2 つ以上の結果の 1 つを予測します)。例えば、機械学習モデルが、「この製品は書籍、自動車、電話のいずれですか?」 または、「このお客様にとって最も関心のある商品のカテゴリはどれですか?」と聞くかもしれません。

ミュータブルなインフラストラクチャ

本番ワークロードに使用する既存のインフラストラクチャを更新および変更するためのモデル。Well-Architected AWS フレームワークでは、一貫性、信頼性、予測可能性を向上させるために、イミュータブルインフラストラクチャの使用をベストプラクティスとして推奨しています。

O

OAC

「オリジンアクセス制御」を参照してください。

OAI

「オリジンアクセスアイデンティティ」を参照してください。

OCM

「組織変更管理」を参照してください。

オフライン移行

移行プロセス中にソースワークロードを停止させる移行方法。この方法はダウンタイムが長くなるため、通常は重要ではない小規模なワークロードに使用されます。

OI

「オペレーション統合」を参照してください。

Ola

「オペレーショナルレベルアグリーメント」を参照してください。

オンライン移行

ソースワークロードをオフラインにせずにターゲットシステムにコピーする移行方法。ワークロードに接続されているアプリケーションは、移行中も動作し続けることができます。この方法はダウンタイムがゼロから最小限で済むため、通常は重要な本番稼働環境のワークロードに使用されます。

OPC-UA

「Open Process Communications - Unified Architecture」を参照してください。

Open Process Communications - Unified Architecture (OPC-UA)

産業オートメーション用のマシンツーマシン (M2M) 通信プロトコル。OPC-UA により、相互運用の際に、データ暗号化、認証、認可の各スキームを標準化できます。

オペレーショナルレベルアグリーメント (OLA)

サービスレベルアグリーメント (SLA) をサポートするために、どの機能的 IT グループが互いに提供することを約束するかを明確にする契約。

運用準備状況レビュー (ORR)

質問と関連するベストプラクティスのチェックリスト。インシデントや起こり得る障害を理解、評価、防止したり、その範囲を縮小したりする際に役立ちます。詳細については、 AWS Well-Architected フレームワークの「Operational Readiness Reviews (ORR)」を参照してください。

運用テクノロジー (OT)

産業オペレーション、機器、インフラストラクチャを制御するために物理環境と連携させるハードウェアおよびソフトウェアシステム。製造分野では、Industry 4.0 への変革を進める上で、OT と情報技術 (IT) システムの統合に焦点が当てられています。

オペレーション統合 (OI)

クラウドでオペレーションをモダナイズするプロセスには、準備計画、オートメーション、統合が含まれます。詳細については、オペレーション統合ガイドを参照してください。

組織の証跡

組織 AWS アカウント 内のすべての のすべてのイベント AWS CloudTrail を ログに記録する によって作成された証跡 AWS Organizations。証跡は、組織に含まれている各 AWS アカウント に作成され、各アカウントのアクティビティを追跡します。詳細については、CloudTrail ドキュメントの「組織の証跡の作成」を参照してください。

組織変更管理 (OCM)

人材、文化、リーダーシップの観点から、主要な破壊的なビジネス変革を管理するためのフレームワーク。OCM は、変化の導入を加速し、移行問題に対処し、文化や組織の変化を推進することで、組織が新しいシステムと戦略の準備と移行するのを支援します。 AWS 移行戦略では、クラウド導入プロジェクトに必要な変化のスピードにより、このフレームワークは人材アクセラレーションと呼ばれます。詳細については、OCM ガイドを参照してください。

オリジンアクセス制御 (OAC)

Amazon Simple Storage Service (Amazon S3) コンテンツを保護するための、CloudFront のアクセス制限の強化オプション。OAC は AWS リージョン、すべての S3 バケット、 AWS KMS (SSE-KMS) によるサーバー側の暗号化、S3 バケットへの動的 PUT および DELETEリクエストをサポートします。

オリジンアクセスアイデンティティ (OAI)

CloudFront の、Amazon S3 コンテンツを保護するためのアクセス制限オプション。OAI を使用すると、CloudFront が、Amazon S3 に認証可能なプリンシパルを作成します。認証されたプリンシパルは、S3 バケット内のコンテンツに、特定の CloudFront ディストリビューションを介してのみアクセスできます。OAC も併せて参照してください。OAC では、より詳細な、強化されたアクセス制御が可能です。

ORR

「運用準備状況レビュー」を参照してください。

OT

「運用テクノロジー」を参照してください。

アウトバウンド (送信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーション内から開始されたネットワーク接続を処理する VPC。AWS Security Reference Architecture では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

P

アクセス許可の境界

ユーザーまたはロールが使用できるアクセス許可の上限を設定する、IAM プリンシパルにアタッチされる IAM 管理ポリシー。詳細については、IAM ドキュメントのアクセス許可の境界を参照してください。

個人を特定できる情報 (PII)

直接閲覧した場合、または他の関連データと組み合わせた場合に、個人の身元を合理的に推測するために使用できる情報。PII の例には、氏名、住所、連絡先情報などがあります。

PII

「個人を特定できる情報」を参照してください。

プレイブック

クラウドでのコアオペレーション機能の提供など、移行に関連する作業を取り込む、事前定義された一連のステップ。プレイブックは、スクリプト、自動ランブック、またはお客様のモダナイズされた環境を運用するために必要なプロセスや手順の要約などの形式をとることができます。

PLC

「プログラマブルロジックコントローラー」を参照してください。

PLM

「製品ライフサイクル管理」を参照してください。

ポリシー

次の操作を可能にするオブジェクト: アクセス許可を定義する (ID ベースのポリシーを参照)。アクセス条件を指定する (リソースベースのポリシーを参照)。 AWS Organizations の組織における全アカウントにアクセス許可の上限を定義する (サービスコントロールポリシーを参照)。

多言語の永続性

データアクセスパターンやその他の要件に基づいて、マイクロサービスのデータストレージテクノロジーを個別に選択します。マイクロサービスが同じデータストレージテクノロジーを使用している場合、実装上の問題が発生したり、パフォーマンスが低下する可能性があります。マイクロサービスは、要件に最も適合したデータストアを使用すると、より簡単に実装でき、パフォーマンスとスケーラビリティが向上します。

ポートフォリオ評価

移行を計画するために、アプリケーションポートフォリオの検出、分析、優先順位付けを行うプロセス。詳細については、「移行の準備状況の評価」を参照してください。

述語

`true` または `false` を返すためのクエリ条件。一般的に、`WHERE` 句に記述されます。

述語プッシュダウン

データベースクエリを最適化する手法。これによって、転送前にクエリ内のデータをフィルタリングします。この手法を取ると、リレーショナルデータベースから取得し処理する必要のあるデータの量が減少するため、クエリのパフォーマンスが向上します。

予防的コントロール

イベントの発生を防ぐように設計されたセキュリティコントロール。このコントロールは、ネットワークへの不正アクセスや好ましくない変更を防ぐ最前線の防御です。詳細については、「AWSでのセキュリティコントロールの実装」の「予防的コントロール」を参照してください。

プリンシパル

アクションを実行し AWS 、リソースにアクセスできる のエンティティ。このエンティティは通常、、IAM AWS アカウントロール、または ユーザーのルートユーザーです。詳細については、IAM ドキュメントの「ロールに関する用語と概念」にあるプリンシパルを参照してください。

プライバシーバイデザイン

開発プロセス全体を通してプライバシーが考慮されているシステムエンジニアリングのアプローチ。

プライベートホストゾーン

1 つ以上の VPC 内のドメインとそのサブドメインへの DNS クエリに対し、Amazon Route 53 がどのように応答するかに関する情報を保持するコンテナ。詳細については、Route 53 ドキュメントの「プライベートホストゾーンの使用」を参照してください。

プロアクティブコントロール

非準拠リソースのデプロイ防止を目的としたセキュリティコントロール。このコントロールにより、プロビジョニング前にリソースをスキャンします。コントロールに準拠していないリソースは、プロビジョニングされません。詳細については、 AWS Control Tower ドキュメントの「コントロールリファレンスガイド」および「 セキュリティコントロールの実装」の「プロアクティブコントロール」を参照してください。 AWS

製品ライフサイクル管理 (PLM)

製品の設計、開発、発売から、成長、成熟、衰退、廃棄に至る、製品のライフサイクル全体を通してデータとプロセスを管理すること。

本番環境

「環境」を参照してください。

プログラマブルロジックコントローラー (PLC)

製造分野で使用される、信頼性と適応性に優れたコンピュータであり、これによって、マシンをモニタリングするとともに、製造プロセスを自動化します。

プロンプトチェイニング

1 つの LLM プロンプトによる出力を次のプロンプトの入力に使用して、より良いレスポンスを生成します。この手法を使用すると、複雑なタスクをサブタスクに分割したり、事前レスポンスを繰り返し改良または拡張したりできます。これによって、モデルのレスポンスの精度と関連性が向上し、粒度の高いパーソナライズされた結果を得られます。

仮名化

データセット内の個人識別子をプレースホルダー値に置き換えるプロセス。仮名化は個人のプライバシー保護に役立ちます。仮名化されたデータは、依然として個人データとみなされます。

発行/サブスクライブ (pub/sub)

マイクロサービス間の非同期通信を可能にするパターン。これにより、スケーラビリティと応答性を向上させます。例えば、マイクロサービスベースの MES の場合、マイクロサービスは、他のマイクロサービスがサブスクライブ可能なチャネルにイベントメッセージを発行できます。このシステムでは、発行サービスの変更なしに、新規マイクロサービスを追加できます。

Q

クエリプラン

手順などの一連のステップであり、SQL リレーショナルデータベースシステムのデータにアクセスするために使用されます。

クエリプランのリグレッション

データベースサービスのオプティマイザーが、データベース環境に特定の変更が加えられる前に選択されたプランよりも最適性の低いプランを選択すること。これは、統計、制限事項、環境設定、クエリパラメータのバインディングの変更、およびデータベースエンジンの更新などが原因である可能性があります。

R

RACI マトリックス

「実行責任者、説明責任者、協業先、報告先 (RACI)」を参照してください。

RAG

「検索拡張生成」を参照してください。

ランサムウェア

決済が完了するまでコンピュータシステムまたはデータへのアクセスをブロックするように設計された、悪意のあるソフトウェア。

RASCI マトリックス

「実行責任者、説明責任者、協業先、報告先 (RACI)」を参照してください。

RCAC

「行と列のアクセス制御」を参照してください。

リードレプリカ

読み取り専用に使用されるデータベースのコピー。クエリをリードレプリカにルーティングして、プライマリデータベースへの負荷を軽減できます。

リアーキテクト

「7 Rs」を参照してください。

目標復旧時点 (RPO)

最後のデータリカバリポイントからの最大許容時間です。これにより、最後の回復時点からサービスが中断されるまでの間に許容できるデータ損失の程度が決まります。

目標復旧時間 (RTO)

サービスの中断から復旧までの最大許容遅延時間。

リファクタリング

「7 Rs」を参照してください。

リージョン

地理的エリア内の AWS リソースのコレクション。各 AWS リージョン は、耐障害性、安定性、耐障害性を提供するために、他の から分離され、独立しています。詳細については、「アカウントが使用できる AWS リージョン を指定する」を参照してください。

リグレッション

数値を予測する機械学習手法。例えば、「この家はどれくらいの値段で売れるでしょうか?」という問題を解決するために、機械学習モデルは、線形回帰モデルを使用して、この家に関する既知の事実 (平方フィートなど) に基づいて家の販売価格を予測できます。

リホスト

「7 Rs」を参照してください。

リリース

デプロイプロセスで、変更を本番環境に昇格させること。

再配置

「7 Rs」を参照してください。

リプラットフォーム

「7 Rs」を参照してください。

再購入

「7 Rs」を参照してください。

回復性

中断に抵抗または中断から回復するアプリケーションの機能。 AWS クラウドでの回復力を計画する際には、一般的に、高可用性とディザスタリカバリが考慮されます。詳細については、「AWS クラウド の耐障害性」を参照してください。

リソースベースのポリシー

Amazon S3 バケット、エンドポイント、暗号化キーなどのリソースにアタッチされたポリシー。このタイプのポリシーは、アクセスが許可されているプリンシパル、サポートされているアクション、その他の満たすべき条件を指定します。

実行責任者、説明責任者、協業先、報告先 (RACI) に基づくマトリックス

移行活動とクラウド運用に関わるすべての関係者の役割と責任を定義したマトリックス。マトリックスの名前は、マトリックスで定義されている責任の種類、すなわち責任 (R) 、説明責任 (A) 、協議 (C) 、情報提供 (I) に由来します。サポート (S) タイプはオプションです。サポートが含まれる場合は RASCI マトリックスと呼ばれ、含まれない場合は RACI マトリックスと呼ばれます。

レスポンシブコントロール

有害事象やセキュリティベースラインからの逸脱について、修復を促すように設計されたセキュリティコントロール。詳細については、「 AWSでのセキュリティコントロールの実装」の「レスポンシブコントロール」を参照してください。

保持

「7 Rs」を参照してください。

廃止

「7 Rs」を参照してください。

検索拡張生成 (RAG)

生成 AI の技術。これにより、LLM では、レスポンスの生成前に、トレーニングデータソースの外部にある信頼できるデータソースが参照されます。例えば、RAG モデルによって、組織のナレッジベースまたはカスタムデータのセマンティック検索を実行できる場合があります。細については、「RAG (検索拡張生成) とは何ですか?」を参照してください。

ローテーション

定期的にシークレット情報を更新して、攻撃者が認証情報にアクセスするのをより困難にするプロセス。

行と列のアクセス制御 (RCAC)

アクセスルールが定義された、基本的で柔軟な SQL 表現の使用。RCAC は行権限と列マスクで構成されています。

RPO

「目標復旧時点」を参照してください。

RTO

「目標復旧時間」を参照してください。

ランブック

特定のタスクを実行するために必要な手動または自動化された一連の手順。これらは通常、エラー率の高い反復操作や手順を合理化するために構築されています。

S

SAML 2.0

多くの ID プロバイダー (IdP) が使用しているオープンスタンダード。この機能を使用すると、フェデレーティッドシングルサインオン (SSO) が有効になるため、ユーザーは組織内のすべてのユーザーを IAM で作成しなくても、 AWS マネジメントコンソール にログインしたり AWS 、API オペレーションを呼び出すことができます。SAML 2.0 ベースのフェデレーションの詳細については、IAM ドキュメントの「SAML 2.0 ベースのフェデレーションについて」を参照してください。

SCADA

「監視制御とデータ取得」を参照してください。

SCP

「サービスコントロールポリシー」を参照してください。

シークレット

暗号化された形式で保存する AWS Secrets Managerパスワードやユーザー認証情報などの機密情報または制限付き情報。シークレット値とそのメタデータで構成されます。シークレット値には、バイナリ、1 つの文字列、複数の文字列を指定できます。詳細については、Secrets Manager ドキュメントの「Secrets Manager シークレットの概要」を参照してください。

セキュリティバイデザイン

開発プロセス全体を通してセキュリティが考慮されているシステムエンジニアリングのアプローチ。

セキュリティコントロール

脅威アクターによるセキュリティ脆弱性の悪用を防止、検出、軽減するための、技術上または管理上のガードレール。セキュリティコントロールには、主に 4 つの種類があります。4 つとは、予防、検出、レスポンシブ、プロアクティブです。

セキュリティ強化

アタックサーフェスを狭めて攻撃への耐性を高めるプロセス。このプロセスには、不要になったリソースの削除、最小特権を付与するセキュリティのベストプラクティスの実装、設定ファイル内の不要な機能の無効化、といったアクションが含まれています。

Security Information and Event Management (SIEM) システム

セキュリティ情報管理 (SIM) とセキュリティイベント管理 (SEM) のシステムを組み合わせたツールとサービス。SIEM システムは、サーバー、ネットワーク、デバイス、その他ソースからデータを収集、モニタリング、分析して、脅威やセキュリティ違反を検出し、アラートを発信します。

セキュリティレスポンスの自動化

セキュリティイベントへの自動レスポンスまたは自動修復を目的として、事前定義およびプログラムされたアクション。これらの自動化は、セキュリティのベストプラクティスを実装するのに役立つ検出的または応答的な AWS セキュリティコントロールとして機能します。自動レスポンスアクションの例には、VPC セキュリティグループの変更、Amazon EC2 インスタンスへのパッチ適用、認証情報の更新などがあります。

サーバー側の暗号化

送信先で、それ AWS のサービス を受け取る によるデータの暗号化。

サービスコントロールポリシー (SCP)

AWS Organizationsの組織内の、すべてのアカウントのアクセス許可を一元的に管理するポリシー。SCP は、管理者がユーザーまたはロールに委任するアクションに、ガードレールを定義したり、アクションの制限を設定したりします。SCP は、許可リストまたは拒否リストとして、許可または禁止するサービスやアクションを指定する際に使用できます。詳細については、 AWS Organizations ドキュメントの「サービスコントロールポリシー」を参照してください。

サービスエンドポイント

のエントリポイントの URL AWS のサービス。ターゲットサービスにプログラムで接続するには、エンドポイントを使用します。詳細については、「AWS 全般のリファレンス」の「AWS のサービス エンドポイント」を参照してください。

サービスレベルアグリーメント (SLA)

サービスのアップタイムやパフォーマンスなど、IT チームがお客様に提供すると約束したものを明示した合意書。

サービスレベルインジケータ (SLI)

エラー率、可用性、スループットといった、サービスパフォーマンス面の指標。

サービスレベル目標 (SLO)

サービスレベルインジケータによって測定され、サービスの状態を表すターゲットメトリクス。

責任共有モデル

クラウドのセキュリティとコンプライアンス AWS について と共有する責任を説明するモデル。AWS はクラウドのセキュリティを担当しますが、 はクラウドのセキュリティを担当します。詳細については、「責任共有モデル」を参照してください。

SIEM

「Security Information and Event Management システム」を参照してください。

単一障害点 (SPOF)

特定のアプリケーションを構成する単一の重要なコンポーネントで発生し、システム稼働に支障をきたす可能性のある障害。

SLA

「サービスレベルアグリーメント」を参照してください。

SLI

「サービスレベルインジケータ」を参照してください。

SLO

「サービスレベルの目標」を参照してください。

スプリットアンドシードモデル

モダナイゼーションプロジェクトのスケーリングと加速のためのパターン。新機能と製品リリースが定義されると、コアチームは解放されて新しい製品チームを作成します。これにより、お客様の組織の能力とサービスの拡張、デベロッパーの生産性の向上、迅速なイノベーションのサポートに役立ちます。詳細については、「AWS クラウドでのアプリケーションをモダナイズするための段階的アプローチ」を参照してください。

SPOF

「単一障害点」を参照してください。

スタースキーマ

データベースの編成構造を意味し、1 つの大きいファクトテーブルにトランザクションデータまたは測定データが保存され、1 つ以上の小さいディメンションテーブルにデータ属性が保存されます。この構造は、データウェアハウスやビジネスインテリジェンスを用途とするように設計されています。

strangler fig パターン

レガシーシステムが廃止されるまで、システム機能を段階的に書き換えて置き換えることにより、モノリシックシステムをモダナイズするアプローチ。このパターンは、宿主の樹木から根を成長させ、最終的にその宿主を包み込み、宿主に取って代わるイチジクのつるを例えています。そのパターンは、モノリシックシステムを書き換えるときのリスクを管理する方法として Martin Fowler により提唱されました。このパターンの適用方法の例については、「コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET (ASMX) ウェブサービスを段階的にモダナイズ」を参照してください。

サブネット

VPC 内の IP アドレスの範囲。サブネットは、1 つのアベイラビリティーゾーンに存在する必要があります。

監視制御とデータ取得 (SCADA)

製造分野において、ハードウェアとソフトウェアを使用して物理アセットと本番運用をモニタリングするシステム。

対称暗号化

データの暗号化と復号に同じキーを使用する暗号化のアルゴリズム。

合成テスト

ユーザーとのやり取りをシミュレートして、起こり得る問題を検出したり、パフォーマンスをモニタリングしたりすることで、システムをテストします。Amazon CloudWatch Synthetics を使用すると、こうしたテストを作成できます。

システムプロンプト

コンテキスト、指示、ガイドラインなどを提示して、LLM に動作を指示する手法。システムプロンプトは、コンテキストを設定して、ユーザーとやり取りするルールを確立するのに有用です。

T

タグ

AWS リソースを整理するためのメタデータとして機能するキーと値のペア。タグは、 リソースの管理、識別、整理、検索、フィルタリングに役立ちます。詳細については、「AWS リソースのタグ付け」を参照してください。

ターゲット変数

監督された機械学習でお客様が予測しようとしている値。これは、結果変数のことも指します。例えば、製造設定では、ターゲット変数が製品の欠陥である可能性があります。

タスクリスト

ランブックの進行状況を追跡するために使用されるツール。タスクリストには、ランブックの概要と完了する必要のある一般的なタスクのリストが含まれています。各一般的なタスクには、推定所要時間、所有者、進捗状況が含まれています。

テスト環境

「環境」を参照してください。

トレーニング

お客様の機械学習モデルに学習するデータを提供すること。トレーニングデータには正しい答えが含まれている必要があります。学習アルゴリズムは入力データ属性をターゲット (お客様が予測したい答え) にマッピングするトレーニングデータのパターンを検出します。これらのパターンをキャプチャする機械学習モデルを出力します。そして、お客様が機械学習モデルを使用して、ターゲットがわからない新しいデータでターゲットを予測できます。

トランジットゲートウェイ

VPC とオンプレミスネットワークを相互接続するために使用できる、ネットワークの中継ハブ。詳細については、 AWS Transit Gateway ドキュメントの「トランジットゲートウェイとは」を参照してください。

トランクベースのワークフロー

デベロッパーが機能ブランチで機能をローカルにビルドしてテストし、その変更をメインブランチにマージするアプローチ。メインブランチはその後、開発環境、本番前環境、本番環境に合わせて順次構築されます。

信頼されたアクセス

ユーザーに代わって AWS Organizations およびそのアカウントで組織内でタスクを実行するために指定したサービスにアクセス許可を付与します。信頼されたサービスは、サービスにリンクされたロールを必要なときに各アカウントに作成し、ユーザーに代わって管理タスクを実行します。詳細については、 ドキュメントの「Using AWS Organizations with other AWS services AWS Organizations 」を参照してください。

チューニング

機械学習モデルの精度を向上させるために、お客様のトレーニングプロセスの側面を変更する。例えば、お客様が機械学習モデルをトレーニングするには、ラベル付けセットを生成し、ラベルを追加します。これらのステップを、異なる設定で複数回繰り返して、モデルを最適化します。

ツーピザチーム

2 枚のピザを分け合えることができるくらい小さな DevOps チーム。ツーピザチームの規模では、ソフトウェア開発におけるコラボレーションに最適な機会が確保されます。

U

不確実性

予測機械学習モデルの信頼性を損なう可能性がある、不正確、不完全、または未知の情報を指す概念。不確実性には、次の 2 つのタイプがあります。認識論的不確実性は、限られた、不完全なデータによって引き起こされ、弁論的不確実性 は、データに固有のノイズとランダム性によって引き起こされます。

未分化なタスク

ヘビーリフティングとも呼ばれ、アプリケーションの作成と運用には必要だが、エンドユーザーに直接的な価値をもたらさなかったり、競争上の優位性をもたらしたりしない作業です。未分化なタスクの例としては、調達、メンテナンス、キャパシティプランニングなどがあります。

上位環境

「環境」を参照してください。

V

バキューミング

ストレージを再利用してパフォーマンスを向上させるために、増分更新後にクリーンアップを行うデータベースのメンテナンス操作。

バージョンコントロール

リポジトリ内のソースコードへの変更など、変更を追跡するプロセスとツール。

VPC ピアリング

プライベート IP アドレスを使用してトラフィックをルーティングできる、2 つの VPC 間の接続。詳細については、Amazon VPC ドキュメントの「VPC ピア機能とは」を参照してください。

脆弱性

システムのセキュリティを脅かすソフトウェアまたはハードウェアの欠陥。

W

ウォームキャッシュ

頻繁にアクセスされる最新の関連データを含むバッファキャッシュ。データベースインスタンスはバッファキャッシュから、メインメモリまたはディスクからよりも短い時間で読み取りを行うことができます。

ウォームデータ

アクセス頻度の低いデータ。この種類のデータをクエリする場合、通常は適度に遅いクエリでも問題ありません。

ウィンドウ関数

現在のレコードに何らかの形で関連している行のグループに計算を実行する SQL 関数。ウィンドウ関数は、移動平均を計算したり、現在の行の相対位置に基づいて他の行の値にアクセスするといったタスクの処理に役立ちます。

ワークロード

ビジネス価値をもたらすリソースとコード (顧客向けアプリケーションやバックエンドプロセスなど) の総称。

ワークストリーム

特定のタスクセットを担当する移行プロジェクト内の機能グループ。各ワークストリームは独立していますが、プロジェクト内の他のワークストリームをサポートしています。たとえば、ポートフォリオワークストリームは、アプリケーションの優先順位付け、ウェーブ計画、および移行メタデータの収集を担当します。ポートフォリオワークストリームは、これらの設備を移行ワークストリームで実現し、サーバーとアプリケーションを移行します。

WORM

「Write-Once-Read-Many」を参照してください。

WQF

「AWS ワークロード資格フレームワーク」を参照してください

Write-Once-Read-Many (WORM)

データを 1 回のみ書き込むことで、データの削除や変更を防ぐストレージモデル。承認済みユーザーは、必要な回数だけデータを読み取ることができますが、変更することはできません。このデータストレージインフラストラクチャは、イミュータブルと見なされます。

Z

ゼロデイエクスプロイト

ゼロデイ脆弱性を悪用した攻撃 (一般的にマルウェアによる)。

ゼロデイ脆弱性

実稼働システムにおける未解決の欠陥または脆弱性。脅威アクターは、このような脆弱性を利用してシステムを攻撃する可能性があります。開発者は、よく攻撃の結果で脆弱性に気付きます。

ゼロショットプロンプト

LLM にタスク実行の手順は提示するが、実行のガイドとして役立つ例 (ショット) は提示しない方法。LLM は、事前トレーニング済みの知識を使用してタスクを処理する必要があります。ゼロショットプロンプトの有効性は、タスクの複雑さとプロンプトの品質によって異なります。「数ショットプロンプト」も参照してください。

ゾンビアプリケーション

平均 CPU およびメモリ使用率が 5% 未満のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するのが一般的です。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。