



でのデータベース分解 AWS

# AWS 規範ガイド



# AWS 規範ガイド: でのデータベース分解 AWS

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは Amazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

# Table of Contents

序章 .....	1
対象者 .....	2
目的 .....	2
課題と責任 .....	3
一般的な課題 .....	3
役割と責任の定義 .....	3
範囲と要件 .....	6
コア分析フレームワーク .....	6
システム境界 .....	7
リリースサイクル .....	7
技術的な制約事項 .....	7
組織コンテキスト .....	8
リスク評価 .....	8
成功基準 .....	8
アクセスコントロール .....	10
データベースラッパーサービスパターン .....	11
利点と制限事項 .....	11
実装 .....	12
例 .....	13
CQRS パターン .....	15
結合と結合 .....	17
結合と結合について .....	17
一般的な結合パターン .....	18
実装結合パターン .....	19
一時的な結合パターン .....	19
デプロイ結合パターン .....	20
ドメイン結合パターン .....	20
一般的な結合パターン .....	21
機能結合パターン .....	21
連続結合パターン .....	22
コミュニケーションの結合パターン .....	22
手続き型結合パターン .....	23
一時的な結合パターン .....	23
論理的または偶発的な結合パターン .....	24

実装 .....	25
ベストプラクティス .....	25
フェーズ 1: データの依存関係をマッピングする .....	25
フェーズ 2: トランザクションの境界とアクセスパターンを分析する .....	25
フェーズ 3: 自己完結型テーブルを特定する .....	26
ビジネスロジック .....	28
フェーズ 1: 分析 .....	28
フェーズ 2: 分類 .....	29
フェーズ 3: 移行 .....	30
ロールバック戦略 .....	30
下位互換性を維持する .....	30
緊急ロールバック計画 .....	31
テーブルの関係 .....	32
非正規化戦略 .....	32
Reference-by-key戦略 .....	33
CQRS パターン .....	33
イベントベースのデータ同期 .....	34
テーブル結合に代わる方法の実装 .....	35
シナリオベースの例 .....	36
ベストプラクティス .....	39
成功の測定 .....	39
ドキュメント要件 .....	39
継続的改善戦略 .....	40
データベース分解における一般的な課題を克服する .....	40
よくある質問 .....	41
範囲と要件に関するよくある質問 .....	41
初期スコープ定義はどの程度詳細にする必要がありますか? .....	42
プロジェクトの開始後に追加の依存関係が見つかった場合はどうなりますか? .....	42
要件が競合するさまざまな部門のステークホルダーにどのように対処すればよいですか? ....	42
ドキュメントが古くなっている場合、技術的な制約を評価する最善の方法は何ですか? .....	42
当面のビジネスニーズと長期的な技術目標のバランスを取るにはどうすればよいですか? ....	43
サイレントステークホルダーから重要な要件が欠落していないことを確認するにはどうすればよいですか? .....	43
これらの推奨事項はモノリシックメインフレームデータベースに適用されますか? .....	43
データベースアクセスに関するよくある質問 .....	44
ラッパーサービスが新しいボトルネックにならないか。 .....	44

既存のストアドプロシージャはどうなりますか? .....	44
移行中にスキーマの変更を管理するにはどうすればよいですか? .....	44
結合と結合に関するよくある質問 .....	45
結合を分析するときに適切なレベルの詳細度を特定するにはどうすればよいですか? .....	45
データベースの結合と結合を分析するために使用できるツールは何ですか? .....	46
結合と結合の検出結果を文書化する最善の方法は何ですか? .....	46
最初に対処すべき結合の問題に優先順位を付けるにはどうすればよいですか? .....	47
複数のオペレーションにまたがるトランザクションを処理するにはどうすればよいですか? .....	47
ビジネスロジックの移行に関するよくある質問 .....	47
最初に移行するストアドプロシージャを特定するにはどうすればよいですか? .....	48
ロジックをアプリケーションレイヤーに移動するリスクは何ですか? .....	48
ロジックをデータベースから遠ざけるときにパフォーマンスを維持するにはどうすればよいですか? .....	49
複数のテーブルを含む複雑なストアドプロシージャではどうすればよいですか? .....	49
移行中にデータベーストリガーを処理するにはどうすればよいですか? .....	50
移行されたビジネスロジックをテストする最善の方法は何ですか? .....	50
データベースロジックとアプリケーションロジックの両方が存在する場合、移行期間を管理するにはどうすればよいですか? .....	50
データベースによって以前に管理されていたアプリケーションレイヤーのエラーシナリオを処理するにはどうすればよいですか? .....	51
次の手順 .....	52
増分戦略 .....	52
技術的考慮事項 .....	52
組織の変更 .....	53
リソース .....	54
AWS 規範ガイド .....	54
AWS ブログ投稿 .....	54
AWS のサービス .....	54
その他のツール .....	54
その他のリソース .....	55
ドキュメント履歴 .....	56
用語集 .....	57
# .....	57
A .....	58
B .....	60

C .....	62
D .....	65
E .....	69
F .....	72
G .....	73
H .....	74
I .....	76
L .....	78
M .....	79
O .....	83
P .....	86
Q .....	89
R .....	89
S .....	92
T .....	96
U .....	97
V .....	98
W .....	98
Z .....	99
.....	c

# でのデータベース分解 AWS

Philippe Wanner と Saurabh Sharma, Amazon Web Services

2025 年 10 月 ([ドキュメント履歴](#))

データベースのモダナイゼーション、特にモノリシックデータベースの分解は、データ管理システムの俊敏性、スケーラビリティ、パフォーマンスを向上させたい組織にとって重要なワークストリームです。ビジネスが成長し、そのデータニーズがより複雑になるにつれて、従来のモノリシックデータベースは多くの場合、ペースに追いつくのに苦労します。これにより、パフォーマンスのボトルネック、メンテナンスの課題、ビジネス要件の変化への適応が困難になります。

以下は、モノリシックデータベースの一般的な課題です。

- ビジネスドメインの不整合 – モノリシックデータベースは、多くの場合、テクノロジーを個別のビジネスドメインに合わせるできないため、組織の成長が制限される可能性があります。
- スケーラビリティの制約 — システムはスケーリング制限に頻繁にぶつかるため、事業拡大の障壁となります。
- アーキテクチャの剛性 – 構造が緊密に結合されているため、システム全体に影響を与えずに特定のコンポーネントを更新することが困難になります。
- パフォーマンスの低下 — データ負荷が増加し、ユーザーの同時実行数が増加すると、システムパフォーマンスが低下することがよくあります。

データベース分解の利点は次のとおりです。

- ビジネスの俊敏性の向上 – 分解により、変化するビジネスニーズに迅速に適応し、独立したスケーリングをサポートします。
- パフォーマンスの最適化 – 分解は、特定のユースケースに合わせてカスタマイズされた特殊なデータベースソリューションを作成し、各データベースを個別にスケールするのに役立ちます。
- コスト管理の向上 – 分解により、リソースの効率的な使用が可能になり、運用コストを削減できます。
- 柔軟なライセンスオプション – 分解により、コストのかかる独自のライセンスからオープンソースの代替ライセンスに移行する機会が生まれます。
- イノベーションの有効化 – 分解により、特定のワークロード専用のデータベースの導入が容易になります。

# 対象者

このガイドは、データベースアーキテクト、クラウドソリューションアーキテクト、アプリケーション開発チーム、エンタープライズアーキテクトに役立ちます。これは、モノリシックデータベースをマイクロサービスに沿ったデータストアに分解し、ドメイン駆動型データベースアーキテクチャを実装し、データベース移行戦略を計画し、増大するビジネス需要に合わせてデータベースオペレーションをスケールするのに役立つように設計されています。このガイドの概念と推奨事項を理解するには、リレーショナルデータベースと NoSQL データベースの原則、AWS マネージドデータベースサービス、マイクロサービスアーキテクチャパターンに精通している必要があります。このガイドは、データベース分解プロジェクトの初期段階にある組織を支援することを目的としています。

## 目的

このガイドは、組織が以下の目標を達成するのに役立ちます。

- ターゲットアーキテクチャを分解するための要件を収集します。
- リスクを評価し、コミュニケーションするための体系的な方法論を開発します。
- 分解計画を作成します。
- 成功メトリクス、主要業績評価指標 (KPIs)、緩和戦略、事業継続計画を定義します。
- ビジネス需要への対応に役立つワークロードの伸縮性を確立します。
- イノベーションを可能にする特定のユースケースに特殊なデータベースを採用する方法について説明します。
- 組織のデータセキュリティとガバナンスを強化します。
- 以下を通じてコストを削減します。
  - ライセンス料金の削減
  - ベンダーロックインの削減
  - より広範なコミュニティサポートとイノベーションへのアクセスを改善
  - コンポーネントごとに異なるデータベーステクノロジーを選択する機能
  - 段階的な移行により、リスクが軽減され、時間の経過とともにコストが分散する
  - リソース使用率の向上

# データベース分解に関する一般的な課題と責任の管理

データベースの分解は、慎重な計画、実行、管理を必要とする複雑なプロセスです。組織がデータインフラストラクチャをモダナイズしようとする、プロジェクトの成功に影響を与える可能性のあるさまざまな課題に直面することがよくあります。このセクションでは、一般的なハードルについて説明し、これらの障害を克服するための構造化されたアプローチを導入します。

## 一般的な課題

データベース分解プロジェクトは、技術面、人材面、ビジネス面においていくつかの課題に直面しています。技術的な面では、分散システム間でデータ整合性を確保するのは大きな障害となります。また、移行期間中にパフォーマンスや安定性に影響する可能性があるため、既存のシステムとシームレスに統合する必要があります。人材関連の課題には、新しいシステムに関連する学習曲線、従業員からの変更に対する潜在的な抵抗、必要なリソースの可用性などがあります。ビジネスの観点からは、プロジェクトは、タイムラインのオーバーラン、予算の制約、移行プロセス中のビジネス中断の可能性のリスクに対処する必要があります。

## 役割と責任の定義

技術的、人的、ビジネス上の側面にまたがるこれらの複雑な課題を考慮すると、プロジェクトの成功には明確な役割と責任を確立することが不可欠です。責任、説明責任、相談、情報 (RACI) マトリックスは、これらの課題をナビゲートするために必要な構造を提供します。意思決定を行う人、作業を実行する人、入力を提供する人、分解の各段階で情報を得る必要がある人を明確に定義します。この明確さは、あいまいな意思決定による遅延を防ぎ、適切なステークホルダーの関与を奨励し、主要な成果物に対する説明責任を生み出すのに役立ちます。このようなフレームワークがないと、チームは責任の重複、コミュニケーションの喪失、エスカレーションパスの不明確さに苦労する可能性があります。問題は、タイムラインと予算の超過のリスクを高めながら、既存の技術的複雑さや変更管理の課題を悪化させる可能性があります。

次のサンプル RACI マトリックスは、組織内の潜在的な役割と責任を明確にするのに役立つ出発点です。

タスクまたはアクティビティ	プロジェクトマネージャー	アーキテクト	デベロッパー	ステークホルダー
---------------	--------------	--------	--------	----------

ビジネス成果と課題を特定する	A/R	R	C	—
範囲を定義し、要件を特定する	A	R	C	C/I
プロジェクトの成功メトリクスを特定する	A	R	C	I
コミュニケーションプランを作成して実行する	A/R	C	C	I
ターゲットアーキテクチャを定義する	I	A/R	C	—
データベースアクセスの制御	I	A/R	R	—
事業継続計画を作成して実行する	A/R	C	I	—
結合と結合を分析する	I	A/R	R	I
ビジネスロジック (ストアードプロシージャなど) をデータベースからアプリケーションレイヤーに移動する	I	A	R	—

結合と呼ばれる  
テーブル関係を  
切り離す

I

A

R

-

# データベース分解の範囲と要件の定義

データベース分解プロジェクトの範囲を定義し、要件を特定するときは、組織のニーズから逆算する必要があります。これには、技術的実現可能性とビジネス価値のバランスを取る体系的なアプローチが必要です。この最初のステップは、プロセス全体の基盤を設定し、プロジェクトの目標が組織の目標と能力と一致していることを確認するのに役立ちます。

このセクションは、以下のトピックで構成されます。

- [コア分析フレームワークの確立](#)
- [データベース分解のシステム境界の定義](#)
- [リリースサイクルの検討](#)
- [データベース分解の技術的制約の評価](#)
- [組織コンテキストについて](#)
- [データベース分解のリスクの評価](#)
- [データベース分解の成功基準の定義](#)

## コア分析フレームワークの確立

スコープ定義は、4つの相互接続されたフェーズを通じて分析をガイドする体系的なワークフローから始まります。この包括的なアプローチにより、データベースの分解作業が、既存のシステムと運用要件を十分に理解していることが保証されます。以下は、コア分析フレームワークのフェーズです。

1. **アクター分析** – データベースとやり取りするすべてのシステムとアプリケーションを徹底的に特定します。これには、書き込みオペレーションを実行するプロデューサーと読み取りオペレーションを処理するコンシューマーの両方をマッピングし、アクセスパターン、頻度、ピーク使用時間をドキュメント化することが含まれます。この顧客中心のビューは、変更の影響を理解し、分解中に特別な注意が必要な重要なパスを特定するのに役立ちます。
2. **アクティビティ分析** – 各アクターが実行する特定のオペレーションについて詳しく説明します。システムごとに詳細な作成、読み取り、更新、削除 (CRUD) マトリックスを作成し、アクセスするテーブルとその方法を特定します。この分析は、分解の自然境界を発見し、現在のアーキテクチャを簡素化できる領域を強調するのに役立ちます。
3. **依存関係マッピング** – システム間の直接依存関係と間接依存関係の両方を文書化し、データフローと関係を明確に視覚化します。これにより、潜在的なブレークポイントや、信頼を得るために慎重な計画が必要な領域を特定できます。この分析では、共有テーブルや外部キーなどの技術

的な依存関係と、ワークフローシーケンスやレポート要件などのビジネスプロセスの依存関係の両方を考慮します。

4. 整合性要件 – 各オペレーションの整合性のニーズを高い基準で検証します。財務取引など、即時整合性が必要なオペレーションを決定します。分析の更新など、他のオペレーションは結果整合性で動作できます。この分析は、プロジェクト全体で分解パターンの選択とアーキテクチャ上の決定に直接影響します。

## データベース分解のシステム境界の定義

システム境界は、1つのシステムが終了する場所と別のシステムが開始する場所を定義する論理境界であり、データの所有権、アクセスパターン、統合ポイントが含まれます。システム境界を定義するときは、包括的な計画と実践的な実装ニーズのバランスを取るために、慎重かつ決定的な選択を行います。データベースを、複数の物理データベースまたはスキーマにまたがる論理単位と見なします。この境界定義は、次の重要な目的を達成します。

- すべての外部アクターとそのインタラクションパターンを識別します
- インバウンドとアウトバウンドの両方の依存関係を包括的にマッピングします
- 技術的制約と運用上の制約を文書化する
- 分解作業の範囲を明確に説明する

## リリースサイクルの検討

データベースの分解を計画するには、リリースサイクルを理解することが不可欠です。ターゲットシステムと依存システムの両方の更新時間を確認します。調整された変更の機会を特定します。接続されたシステムの計画的な廃止を検討してください。これは、分解戦略に影響する可能性があるためです。既存の変更ウィンドウとデプロイの制約を考慮して、ビジネスの中断を最小限に抑えます。実装計画が、接続されているすべてのシステムのリリーススケジュールと一致していることを確認します。

## データベース分解の技術的制約の評価

データベースの分解に進む前に、モダナイゼーションアプローチを形成する主要な技術的制限を評価してください。データベースバージョン、フレームワーク、パフォーマンス要件、サービスレベルアグリーメントなど、現在のテクノロジースタックの機能を調べます。特に規制された業界については、セキュリティとコンプライアンスの義務を考慮してください。現在のデータ量、成長予測、利用

可能な移行ツールを確認して、スケーリングの決定に役立ててください。最後に、ソースコードとシステムの変更に対するアクセス権を確認します。これらは実行可能な分解戦略を決定するためです。

## 組織コンテキストについて

データベースの分解を成功させるには、システムが動作するより広範な組織環境を理解する必要があります。部門間の依存関係をマッピングし、チーム間で明確なコミュニケーションチャネルを確立します。チームの技術的能力を評価し、対処する必要があるトレーニングニーズやスキルギャップを特定します。移行を管理し、ビジネス継続性を維持する方法など、変更管理の影響を考慮します。利用可能なリソースと、予算や人員の制限などの制約を評価します。最後に、分解戦略をステークホルダーの期待と優先順位に合わせ、プロジェクト全体で継続的なサポートを促進します。

## データベース分解のリスクの評価

データベースの分解を成功させるには、包括的なリスク評価が不可欠です。移行中のデータ整合性、システムパフォーマンスの低下の可能性、統合の失敗の可能性、セキュリティの脆弱性などのリスクを慎重に評価します。これらの技術的な課題は、潜在的な運用の中断、リソースの制限、タイムラインの遅延、予算の制約など、ビジネスリスクとのバランスを取る必要があります。特定されたリスクごとに、特定の緩和戦略と緊急時対応計画を策定して、事業運営を保護しながらプロジェクトの勢いを維持します。

潜在的な問題の影響と可能性の両方を評価するリスクマトリックスを作成します。技術チームやビジネスステークホルダーと協力してリスクを特定し、介入の明確なしきい値を設定し、特定の緩和戦略を策定します。たとえば、データ損失リスクを高い影響と低い確率で評価し、堅牢なバックアップ戦略が必要です。パフォーマンスの軽微な低下は、中程度の影響と高い確率で発生する可能性があり、プロアクティブモニタリングが必要です。

定期的なリスクレビューサイクルを確立して優先順位を再評価し、プロジェクトの進化に合わせて緩和計画を調整します。この体系的なアプローチにより、新たな問題の明確なエスカレーションパスを維持しながら、リソースが最も重要なリスクに集中できるようになります。

## データベース分解の成功基準の定義

データベース分解の成功基準は明確に定義され、複数の次元にわたって測定可能である必要があります。ビジネスの観点から、コスト削減、time-to-market短縮、システムの可用性、顧客満足度に関する具体的な目標を設定します。技術的成功は、システムパフォーマンス、デプロイ効率、データ整合性、および全体的な信頼性を定量化できる改善によって測定する必要があります。移行プ

プロセスでは、ゼロデータ損失、許容可能なビジネス中断制限、予算コンプライアンス、タイムラインの遵守に関する厳格な要件を定義します。

ベースラインメトリクスとターゲットメトリクス、明確な測定方法、定期的なレビュースケジュールを維持することで、これらの基準を徹底的に文書化します。成功メトリクスごとに明確な所有者を割り当て、異なるメトリクス間の依存関係をマッピングします。この成功を測定するための包括的なアプローチは、技術的成果とビジネス成果を統合させながら、分解ジャーニー全体で説明責任を維持します。

## 分解中のデータベースアクセスの制御

多くの組織は一般的なシナリオに直面しています。何年もかけて組織的に成長し、複数のサービスやチームから直接アクセスされる中央データベースです。これにより、いくつかの重大な問題が発生します。

- 制御されていない成長 — チームが継続的に新機能を追加し、スキーマを変更すると、データベースはますます複雑になり、管理が困難になります。
- パフォーマンスの懸念 — ハードウェアの改善にもかかわらず、負荷の増加は最終的にデータベースの機能を超越する可能性があります。スキーマの複雑さやスキル不足が原因でクエリをチューニングできない。システムパフォーマンスを予測または説明できません。
- 分解麻痺 — 複数のチームによって積極的に変更されている間、データベースを分割またはリファクタリングすることはほぼ不可能になります。

### Note

モノリシックデータベースシステムは、多くの場合、アプリケーションやサービス、管理に同じ認証情報を再利用します。これにより、データベースのトレーサビリティが低下します。[専用ロール](#)を設定し、[最小特権の原則](#)を採用することで、セキュリティと可用性を高めることができます。

扱いにくいモノリシックデータベースを扱う場合、アクセスを制御する最も効果的なパターンの1つはデータベースラッパーサービスと呼ばれます。複雑なデータベースシステムを管理するための戦略的最初のステップを提供します。制御されたデータベースアクセスを確立し、リスクを軽減しながら、段階的なモダナイゼーションを可能にします。このアプローチは、データ使用パターンと依存関係を明確に可視化することで、段階的な改善の基盤を構築します。これは、データベースを完全に分解するためのステップとして機能する移行アーキテクチャです。ラッパーサービスは、そのジャーニーを成功させるために必要な安定性と制御を提供します。

このセクションは、以下のトピックで構成されます。

- [データベースラッパーサービスパターンによるアクセスの制御](#)
- [CQRS パターンによるアクセスの制御](#)

## データベースラッパーサービスパターンによるアクセスの制御

ラッパーサービスは、データベースのファサードとして機能するサービスレイヤーです。このアプローチは、将来の分解に備えながら既存の機能を維持する必要がある場合に特に役立ちます。このパターンは単純な原則に従います。何かが混乱しすぎる場合は、混乱を含めることから始めます。ラッパーサービスは、データベースにアクセスするための唯一の認可された方法となり、基盤となる複雑さを隠しながら制御されたインターフェイスを提供します。

このパターンは、複雑なスキーマが原因でデータベースをすぐに分解できない場合や、複数のサービスで継続的なデータアクセスが必要な場合に使用します。システムの安定性を維持しながら慎重にリファクタリングする時間を提供するため、移行期間中は特に重要です。このパターンは、データの所有権を特定のチームに統合する場合や、新しいアプリケーションが複数のテーブルに集約されたビューを必要とする場合に適しています。

たとえば、次の場合にこのパターンを適用します。

- スキーマの複雑さにより、すぐに分離できない
- 複数のチームに継続的なデータアクセスが必要
- 段階的なモダナイゼーションが推奨されます
- チーム再構築には明確なデータ所有権が必要です
- 新しいアプリケーションには統合データビューが必要です

## データベースラッパーサービスパターンの利点と制限

データベースラッパーパターンの利点は次のとおりです。

- 制御された増加 – ラッパーサービスは、データベーススキーマへの制御されていない追加を防止します。
- 明確な境界 — 実装プロセスは、明確な所有権と責任の境界を確立するのに役立ちます。
- リファクタリングの自由 – ラッパーサービスを使用すると、コンシューマーに影響を与えることなく内部的な変更を行うことができます。
- オブザーバビリティの向上 – ラッパーサービスは、モニタリングとログ記録のための単一のポイントです。
- テストの簡素化 – ラッパーサービスを使用すると、サービスを消費してテスト用のモックバージョンを簡単に作成できます。

データベースラッパーパターンの制限は次のとおりです。

- テクノロジーカップリング – ラッパーサービスは、消費するサービスと同じテクノロジースタックを使用する場合に最適です。
- 初期オーバーヘッド – ラッパーサービスには、パフォーマンスに影響を与える可能性のある追加のインフラストラクチャが必要です。
- 移行作業 – ラッパーサービスを実装するには、チーム間で調整して直接アクセスから移行する必要があります。
- パフォーマンス – ラッピングサービスでトラフィックが多い、使用量が多い、または頻繁にアクセスする場合、サービスを消費するとパフォーマンスが低下する可能性があります。データベース上で、ラッパーサービスはページ分割、カーソル、データベース接続を処理する必要があります。ユースケースによっては、うまくスケールリングできず、抽出、変換、ロード (ETL) ワークロードに適していない可能性があります。

## データベースラッパーサービスパターンの実装

データベースラッパーサービスパターンを実装するには、2つのフェーズがあります。まず、データベースラッパーサービスを作成します。次に、それを通じてすべてのアクセスを指示し、アクセスパターンを文書化します。

### フェーズ 1: データベースラッパーサービスの作成

データベースのゲートキーパーとして機能する軽量サービスレイヤーを作成します。最初は、既存のすべての機能をミラーリングする必要があります。このラッパーサービスは、すべてのデータベースオペレーションの必須アクセスポイントになり、直接データベースの依存関係をサービスレベルの依存関係に変換します。このレイヤーに詳細なログ記録とモニタリングを実装して、使用パターン、パフォーマンスメトリクス、アクセス頻度を追跡します。既存のストアードプロシージャは維持しますが、この新しいサービスインターフェイスを介してのみアクセスされるようにしてください。

### フェーズ 2: アクセスコントロールの実装

ラッパーサービスを介してすべてのデータベースアクセスを体系的にリダイレクトし、データベースに直接アクセスする外部システムからデータベースの直接アクセス許可を取り消します。サービスが移行されるたびに、各アクセスパターンと依存関係を文書化します。この制御されたアクセスにより、外部コンシューマーを中断することなく、データベースコンポーネントの内部リファクタリングが可能になります。たとえば、複雑なトランザクションワークフローではなく、低リスクの読み取り専用オペレーションから始めます。

## フェーズ 3: データベースのパフォーマンスをモニタリングする

ラッパーサービスをデータベースパフォーマンスの一元的なモニタリングポイントとして使用します。クエリの応答時間、使用パターン、エラー率、リソース使用率などの主要なメトリクスを追跡します。パフォーマンスしきい値と異常なパターンのアラートを設定します。例えば、実行速度の遅いクエリ、接続プールの使用率、トランザクションスループットをモニタリングして、潜在的な問題を事前に特定します。

この統合ビューを使用して、クエリの調整、リソース割り当ての調整、使用パターン分析を通じてデータベースのパフォーマンスを最適化します。ラッパーサービスの一元化された性質により、一貫したパフォーマンス標準を維持しながら、改善を実装し、すべてのコンシューマーへの影響を検証することが容易になります。

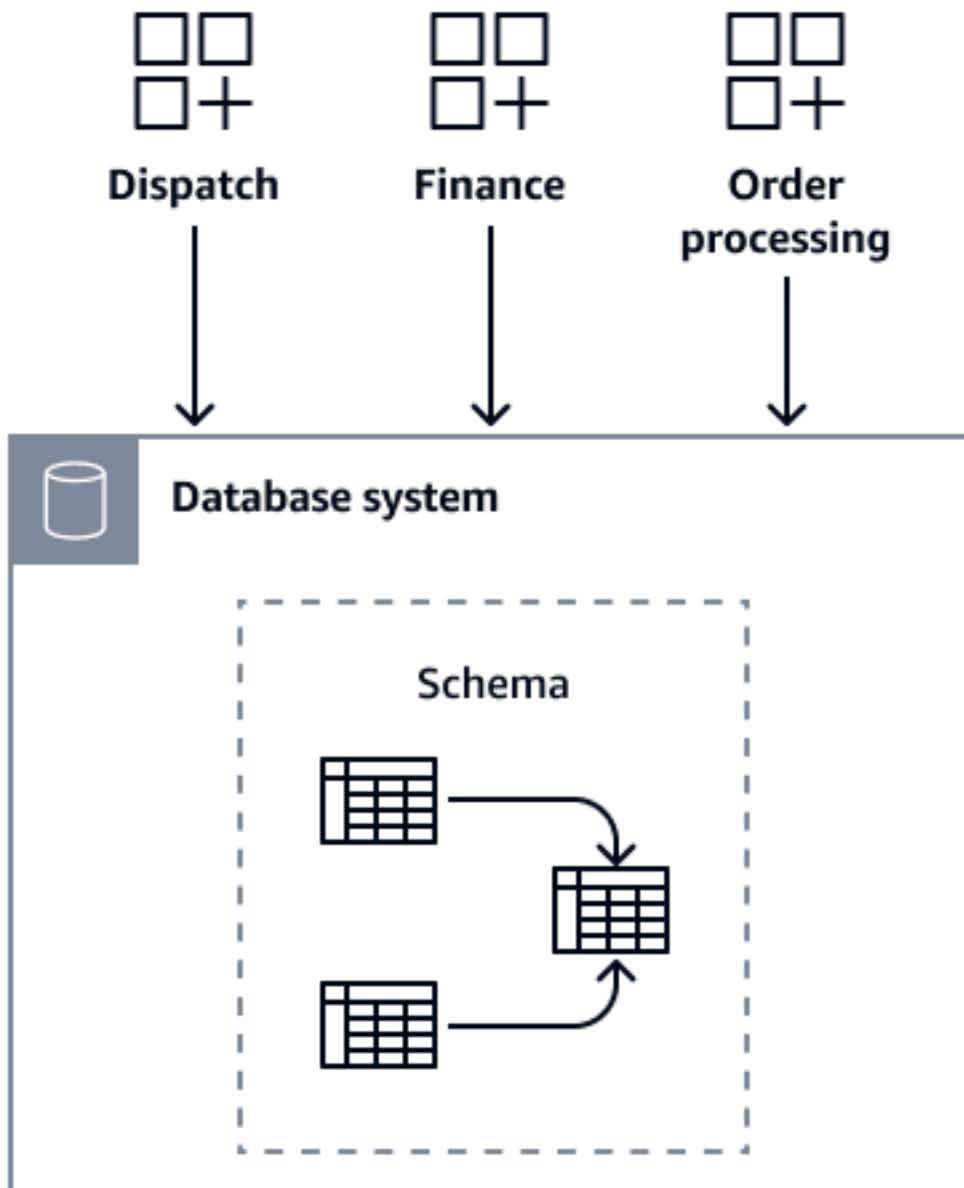
## データベースラッパーサービスを実装するためのベストプラクティス

以下のベストプラクティスは、データベースラッパーサービスの実装に役立ちます。

- Start small – 既存の機能を単純にプロキシする最小限のラッパーから始めます。
- 安定性の維持 – 内部改善を行いながら、サービスインターフェイスを安定させます。
- 使用状況のモニタリング – アクセスパターンを理解するための包括的なモニタリングを実装する
- 所有権のクリア – 専用チームを割り当てて、ラッパーと基盤となるスキーマの両方を維持します。
- ローカルストレージを奨励する – チームが自分のデータベースにデータを保存するよう促す

## シナリオベースの例

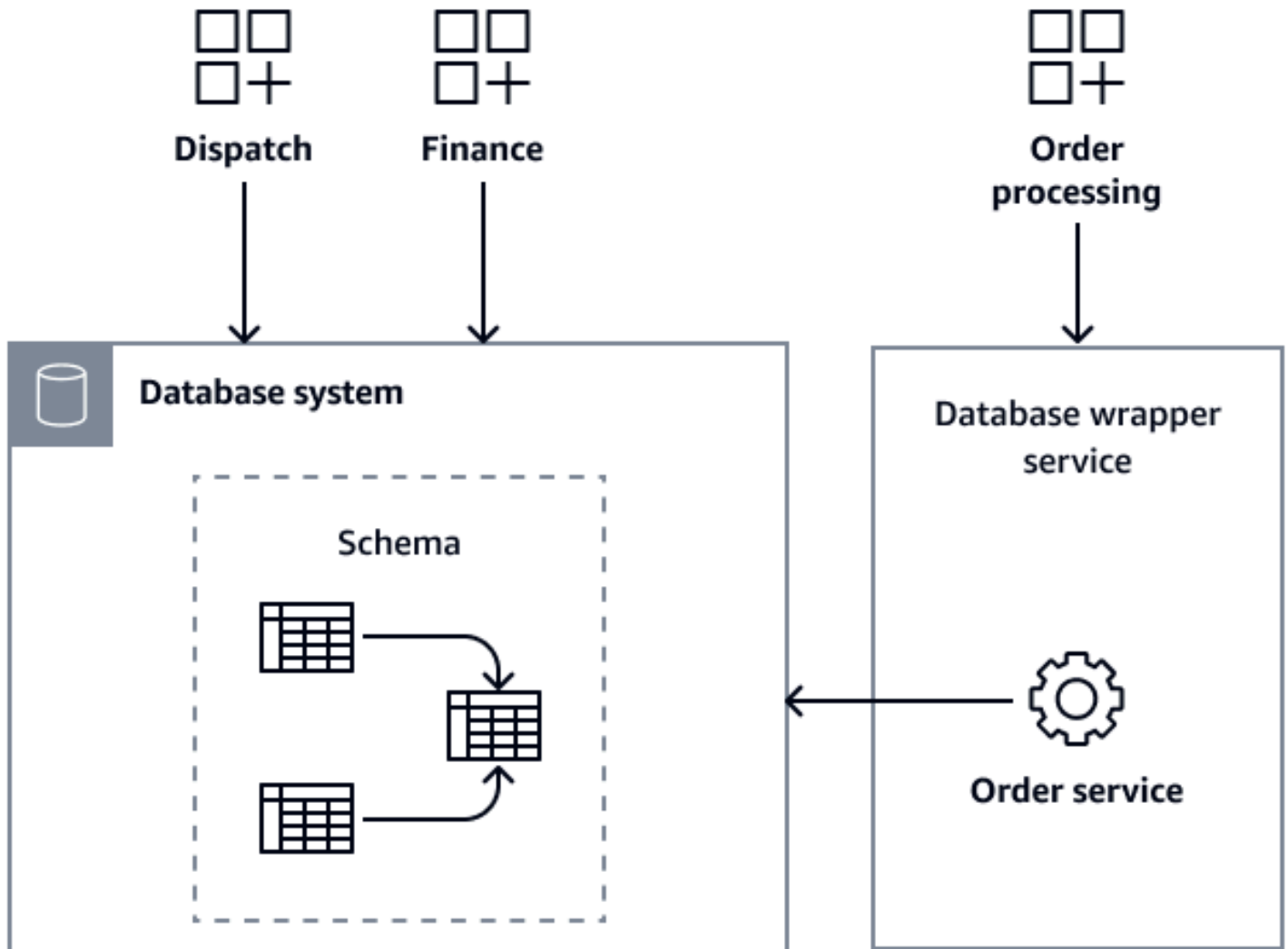
このセクションでは、AnyCompany Books という名前の架空の企業がデータベースラッパーパターンを使用してモノリシックデータベースシステムへのアクセスを制御する方法について説明します。AnyCompany Books には、ディスパッチ、財務、注文処理の 3 つの重要なサービスがあります。これらのサービスは、中央データベースへのアクセスを共有します。各サービスは、異なるチームによって管理されます。時間の経過とともに、特定のニーズに合わせてデータベーススキーマを個別に変更します。これにより、依存関係のウェブが複雑になり、データベース構造がますます複雑になっています。



会社のアプリケーションアーキテクトまたはエンタープライズアーキテクトは、このモノリシックデータベースを分解する必要があることを認識しています。目標は、メンテナンス性を向上させ、チーム間の依存関係を減らすために、各サービスに独自の専用データベースを提供することです。ただし、大きな課題に直面しています。3つのチームすべてが進行中のプロジェクトのためにデータベースを積極的に変更し続けている間、データベースを分解することはほぼ不可能です。スキーマの継続的な変更とチーム間の調整の欠如により、大規模な再構築を試みることは非常にリスクが高くなります。

アーキテクトは、データベースラッパーサービスパターンを使用して、モノリシックデータベースへのアクセスの制御を開始します。まず、Order サービスと呼ばれる特定のモジュールのデータベース

ラッパーサービスを設定します。次に、データベースに直接アクセスするのではなく、注文処理サービスをリダイレクトしてラッパーサービスにアクセスします。次の図は、変更されたインフラストラクチャを示しています。

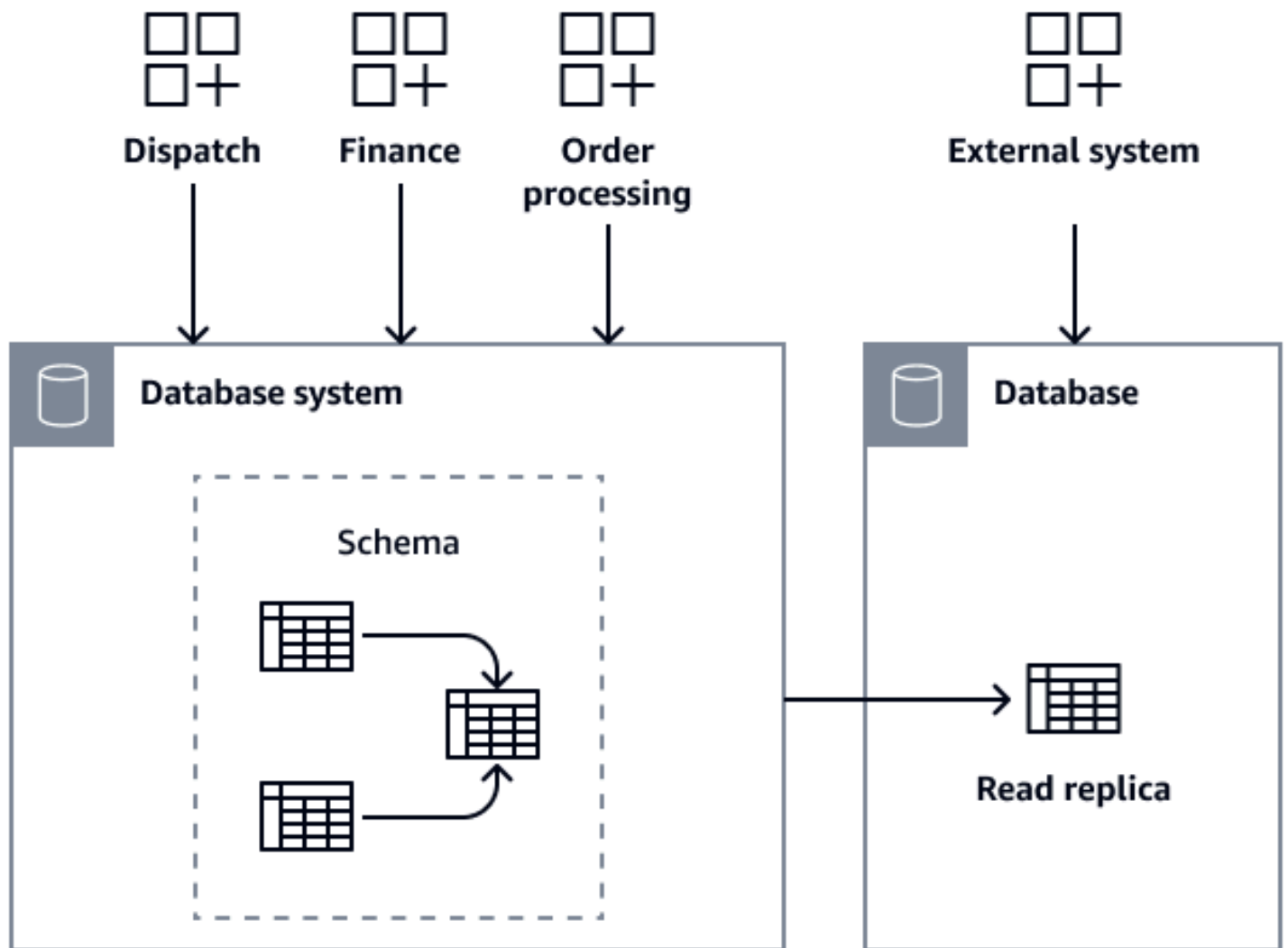


## CQRS パターンによるアクセスの制御

この中央データベースに接続する外部システムを分離するために使用できるもう 1 つのパターンは、コマンドクエリ責任分離 (CQRS) です。分析、レポート、その他の読み取り集約型オペレーションなど、一部の外部システムが主に読み取りのために中央データベースに接続している場合は、個別の読み取り最適化データストアを作成できます。

このパターンは、これらの外部システムをデータベースの分解やスキーマの変更の影響から効果的に分離します。特定のクエリパターンに対して専用のリードレプリカまたは専用データストアを維持することで、チームはプライマリデータベース構造の変更の影響を受けずに運用を継続できます。た

たとえば、モノリシックデータベースを分解している間、レポートシステムは既存のデータビューを引き続き操作でき、分析ワークロードは専用の分析ストアを通じて現在のクエリパターンを維持できます。このアプローチは技術的な分離を提供し、さまざまなチームがプライマリデータベースのトランスフォーメーションジャーニーに緊密に結合することなくシステムを個別に進化させることができるため、組織の自律性を可能にします。



このパターンの詳細とテーブルの関係を切り離すための使用例については、このガイドの [CQRS パターン](#) 後半の「」を参照してください。

# データベース分解のための結合と結合の分析

このセクションでは、モノリシックデータベースの結合パターンと結合パターンを分析して分解をガイドするのに役立ちます。データベースコンポーネントがどのように相互作用し、相互に依存するかを理解することは、自然なブレークポイントを特定し、複雑さを評価し、段階的な移行アプローチを計画するために不可欠です。この分析では、隠れた依存関係が明らかになり、即時の分離に適した領域が強調表示され、変換リスクを最小限に抑えながら分解作業の優先順位を付けるのに役立ちます。結合と結合の両方を調べることで、変換プロセス全体でシステムの安定性を維持するために、コンポーネントの分離シーケンスについて情報に基づいた決定を行うことができます。

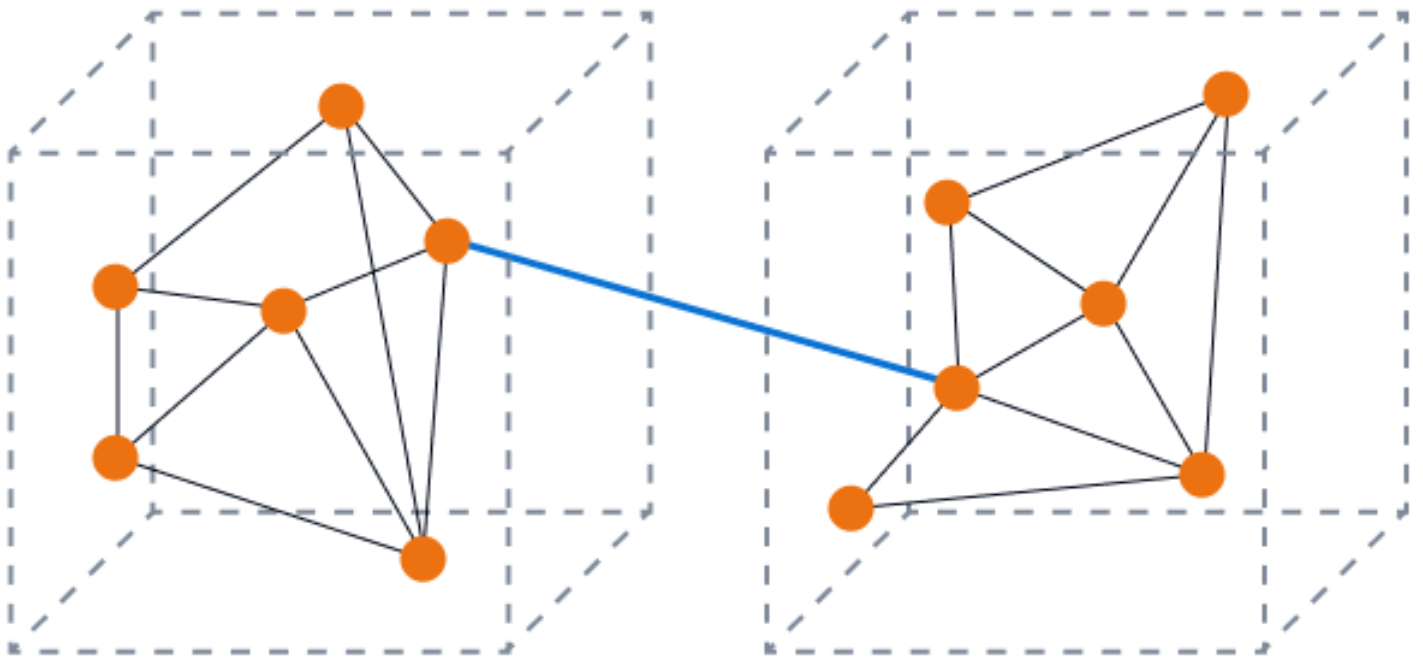
このセクションは、以下のトピックで構成されます。

- [結合と結合について](#)
- [モノリシックデータベースの一般的な結合パターン](#)
- [モノリシックデータベースの一般的な結合パターン](#)
- [低結合と高結合の実装](#)

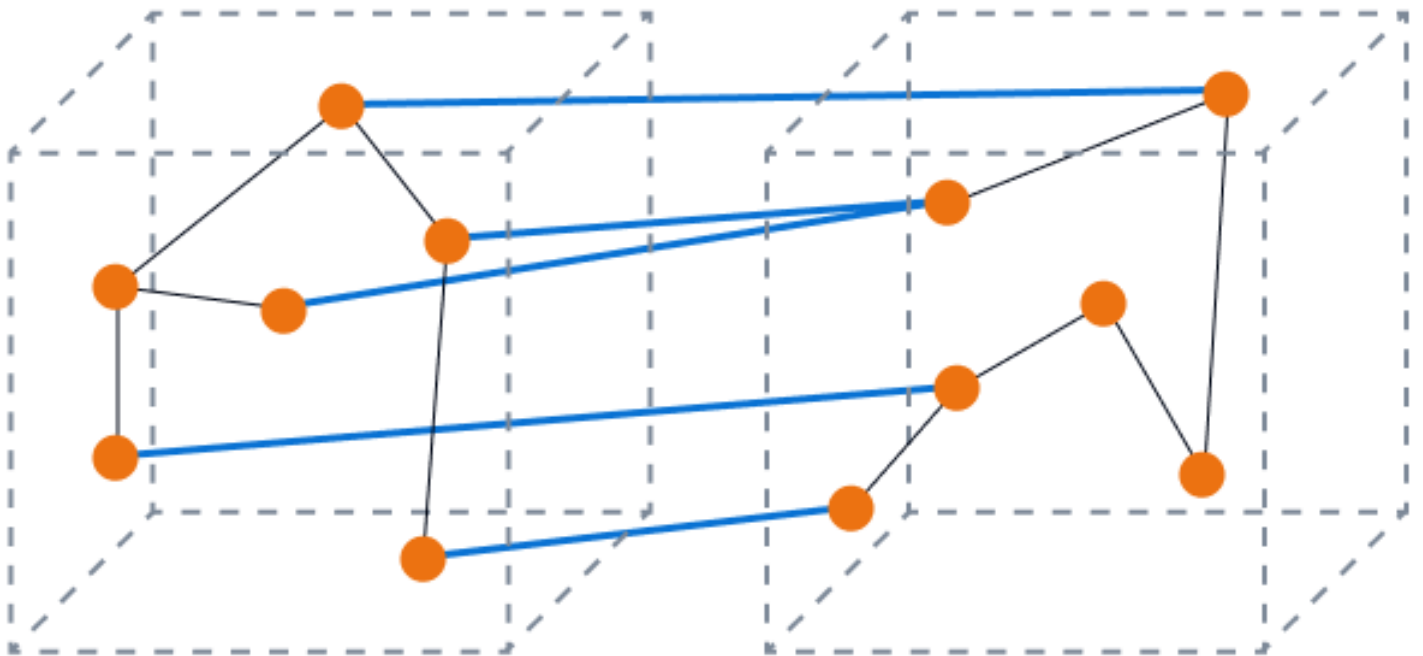
## 結合と結合について

カップリングは、データベースコンポーネント間の相互依存度を測定します。適切に設計されたシステムでは、1つのコンポーネントへの変更が他のコンポーネントに与える影響を最小限に抑えながら、疎結合を実現したいと考えています。結合は、データベースコンポーネント内の要素が連携して、明確に定義された1つの目的にどの程度役立つかを測定します。高い結合は、コンポーネントの要素が強く関連し、特定の関数に焦点を当てていることを示します。モノリシックデータベースを分解する場合は、個々のコンポーネント間の結合と結合の両方を分析する必要があります。この分析は、システムの整合性とパフォーマンスを維持しながらデータベースを分割する方法に関して、情報に基づいた意思決定を行うのに役立ちます。

次の図は、結合性が高い疎結合を示しています。データベース内のコンポーネントは連携して特定の関数を実行し、変更が1つのコンポーネントに与える影響を最小限に抑えます。これは理想的な状態です。



次の画像は、結合率が低く結合率が高いことを示しています。データベースコンポーネントは切断され、変更は他のコンポーネントに影響を与える可能性が高くなります。



## モノリシックデータベースの一般的な結合パターン

モノリシックデータベースをマイクロサービス固有のデータベースに分解するときによく見られる結合パターンがいくつかあります。これらのパターンを理解することは、データベースのモダナイゼー

シヨンの取り組みを成功させるために不可欠です。このセクションでは、各パターン、その課題、および結合を減らすためのベストプラクティスについて説明します。

## 実装結合パターン

定義: コンポーネントはコードレベルとスキーマレベルで緊密に相互接続されています。たとえば、customerテーブルの構造を変更するとorder、inventory、およびbillingのサービスに影響します。

モダナイゼーションの影響: 各マイクロサービスには、専用のデータベーススキーマとデータアクセスレイヤーが必要です。

課題:

- 共有テーブルの変更は複数のサービスに影響します
- 意図しない副作用のリスクが高い
- テストの複雑さの増大
- 個々のコンポーネントの変更が困難

結合を減らすためのベストプラクティス:

- コンポーネント間の明確なインターフェイスを定義する
- 抽象化レイヤーを使用して実装の詳細を非表示にする
- ドメイン固有のスキーマを実装する

## 一時的な結合パターン

定義: オペレーションは特定のシーケンスで実行する必要があります。たとえば、インベントリの更新が完了するまで、注文処理を続行することはできません。

モダナイゼーションの影響: 各マイクロサービスには自律的なデータ管理が必要です。

課題:

- サービス間の同期依存関係の解除
- パフォーマンスのボトルネック
- 最適化が困難

## • 制限された並列処理

結合を減らすためのベストプラクティス:

- 可能な場合は非同期処理を実装する
- イベント駆動型アーキテクチャを使用する
- 必要に応じて結果整合性を設計する

## デプロイ結合パターン

定義: システムコンポーネントは 1 つのユニットとしてデプロイする必要があります。たとえば、支払い処理ロジックを少し変更するには、データベース全体を再デプロイする必要があります。

モダナイゼーションの影響: サービスあたりの独立したデータベースデプロイ

課題:

- 高リスクデプロイ
- デプロイ頻度の制限
- 複雑なロールバック手順

結合を減らすためのベストプラクティス:

- 個別にデプロイ可能なコンポーネントに分割する
- データベースシャーディング戦略を実装する
- Blue-Green デプロイパターンを使用する

## ドメイン結合パターン

定義: ビジネスドメインはデータベース構造とロジックを共有します。たとえば、customer、order および inventory ドメインはテーブルとストアプロシージャを共有します。

モダナイゼーションの影響: ドメイン固有のデータ分離

課題:

- 複雑なドメイン境界
- 個々のドメインのスケーリングが困難
- 複雑なビジネスルール

結合を減らすためのベストプラクティス:

- 明確なドメイン境界を特定する
- ドメインコンテキストでデータを区切る
- ドメイン固有のサービスの実装

## モノリシックデータベースの一般的な結合パターン

データベースコンポーネントの分解を評価する際によく見られる結合パターンがいくつかあります。これらのパターンを理解することは、適切に構造化されたデータベースコンポーネントを特定する上で不可欠です。このセクションでは、各パターン、その特性、および結合を強化するためのベストプラクティスについて説明します。

### 機能結合パターン

**定義:** すべての要素が 1 つの明確に定義された関数を直接サポートし、実行に貢献します。たとえば、支払い処理モジュール内のすべてのストアードプロシージャとテーブルは、支払い関連のオペレーションのみを処理します。

**モダナイゼーションの影響:** マイクロサービスデータベース設計に最適なパターン

**課題:**

- 明確な機能境界の特定
- 多目的コンポーネントの分離
- 単一責任の維持

結合を強化するためのベストプラクティス:

- 関連する関数をグループ化する
- 関連しない機能を削除する
- コンポーネントの境界を明確に定義する

## 連続結合パターン

定義: ある要素からの出力は、別の要素の入力になります。たとえば、注文フィードの注文処理の検証結果などです。

モダナイゼーションの影響: 慎重なワークフロー分析とデータフローマッピングが必要

課題:

- ステップ間の依存関係の管理
- 失敗シナリオの処理
- プロセス順序の維持

結合を強化するためのベストプラクティス:

- 明確なデータフローを文書化する
- 適切なエラー処理を実装する
- ステップ間の明確なインターフェイスを設計する

## コミュニケーションの結合パターン

定義: 要素は同じデータで動作します。たとえば、顧客プロフィール管理関数はすべて顧客データを使用します。

モダナイゼーションの影響: サービス分離のデータ境界を特定し、モジュール間の結合を減らすのに役立ちます

課題:

- データ所有権の決定
- 共有データアクセスの管理
- データ整合性の維持

結合を強化するためのベストプラクティス:

- 明確なデータの所有権を定義する
- 適切なデータアクセスパターンを実装する

- 効果的なデータパーティショニングを設計する

## 手続き型結合パターン

定義: 要素は特定の順序で実行する必要があるため、グループ化されますが、機能的に関連しない場合があります。例えば、注文処理では、注文検証とユーザー通知の両方を処理するストアドプロシージャは、異なる目的を果たし、別々のサービスで処理できる場合でも、順番に行われるという理由だけでグループ化されます。

モダナイゼーションの影響: プロセスフローを維持しながら手順を慎重に分離する必要があります

課題:

- 分解後の正しいプロセスフローの維持
- 手続き型依存関係と比較した真の機能境界の特定

結合を強化するためのベストプラクティス:

- 実行順序ではなく、機能的な目的に基づいて手順を分離する
- オークストレーションパターンを使用してプロセスフローを管理する
- 複雑なシーケンスのワークフロー管理システムを実装する
- プロセスステップを個別に処理するようにイベント駆動型アーキテクチャを設計する

## 一時的な結合パターン

定義: 要素はタイミング要件に関連しています。たとえば、注文が行われると、複数のオペレーションを一緒に実行する必要があります。一貫した注文状態を維持するためには、在庫チェック、支払い処理、注文確認、配送通知がすべて特定の時間枠内に行われる必要があります。

モダナイゼーションへの影響: 分散システムでは特別な処理が必要になる場合があります

課題:

- 分散サービス間のタイミング依存関係の調整
- 分散トランザクションの管理
- 複数のコンポーネントにわたるプロセスの完了の確認

結合を強化するためのベストプラクティス:

- 適切なスケジューリングメカニズムとタイムアウトを実装する
- 明確なシーケンス処理でイベント駆動型アーキテクチャを使用する
- 補償パターンとの最終的な一貫性を実現する設計
- 分散トランザクションの saga パターンを実装する

## 論理的または偶発的な結合パターン

定義: 要素は、関係が弱い、または意味のある関係がない場合でも、同じことを行うように論理的に分類されます。例えば、顧客注文データ、倉庫在庫数、マーケティング E メールテンプレートは、アクセスパターン、ライフサイクル管理、スケーリング要件が異なるにもかかわらず、すべて販売オペレーションに関連しているため、同じデータベーススキーマに保存されます。もう 1 つの例は、同じデータベースコンポーネント内で注文支払い処理と製品カタログ管理を組み合わせることです。どちらも e コマースシステムの一部であり、運用上のニーズが異なる個別のビジネス機能を提供します。

モダナイゼーションの影響: リファクタリングまたは再編成する必要があります

課題:

- より良い組織パターンの特定
- 不要な依存関係の解消
- 任意にグループ化されたコンポーネントの再構築

結合を強化するためのベストプラクティス:

- 真の機能境界とビジネスドメインに基づいて再編成する
- 表面的な関係に基づいて任意のグループ化を削除する
- ビジネス能力に基づいて要素を適切に分離する
- データベースコンポーネントを特定の運用要件に合わせる

# 低結合と高結合の実装

## ベストプラクティス

以下のベストプラクティスは、低い結合を実現するのに役立ちます。

- データベースコンポーネント間の依存関係を最小限に抑える
- コンポーネントインタラクションに明確に定義されたインターフェイスを使用する
- 共有状態とグローバルデータ構造を避ける

次のベストプラクティスは、高い一貫性を実現するのに役立ちます。

- 関連するデータとオペレーションをグループ化する
- 各コンポーネントに1つの明確な責任があることを確認します。
- 異なるビジネスドメイン間の明確な境界を維持する

## フェーズ 1: データの依存関係をマッピングする

データ関係をマッピングし、自然境界を特定します。などのツールを使用して[SchemaSpy](#)、エンティティ関係 (ER) 図でテーブルを表示することで、データベースを視覚化できます。これにより、データベースの静的分析が提供され、データベース内の明確な境界と依存関係の一部が示されます。

データベーススキーマは、グラフデータベースまたはJupiterノートブックでエクスポートすることもできます。次に、クラスタリングまたは相互接続されたコンポーネントアルゴリズムを適用して、自然境界と依存関係を特定できます。などの他の AWS Partner ツールは[CAST Imaging](#)、データベースの依存関係を理解するのに役立ちます。

## フェーズ 2: トランザクションの境界とアクセスパターンを分析する

トランザクションパターンを分析して、アトミック性、一貫性、分離性、耐久性 (ACID) プロパティを維持し、データへのアクセス方法と変更方法を理解します。[Oracle Automatic Workload Repository \(AWR\)](#) やなどのデータベース分析と診断ツールを使用できます[PostgreSQL pg\\_stat\\_statements](#)。この分析は、データベースにアクセスするユーザーとトランザクションの境界を理解するのに役立ちます。また、実行時にテーブル間の結合と結合を理解するのにも役立ちます。また、などのコードとデータベース実行プロファイルをリンクできるモニタリングおよびプロファイリングツールを使用することもできます[Dynatrace AppEngine](#)。

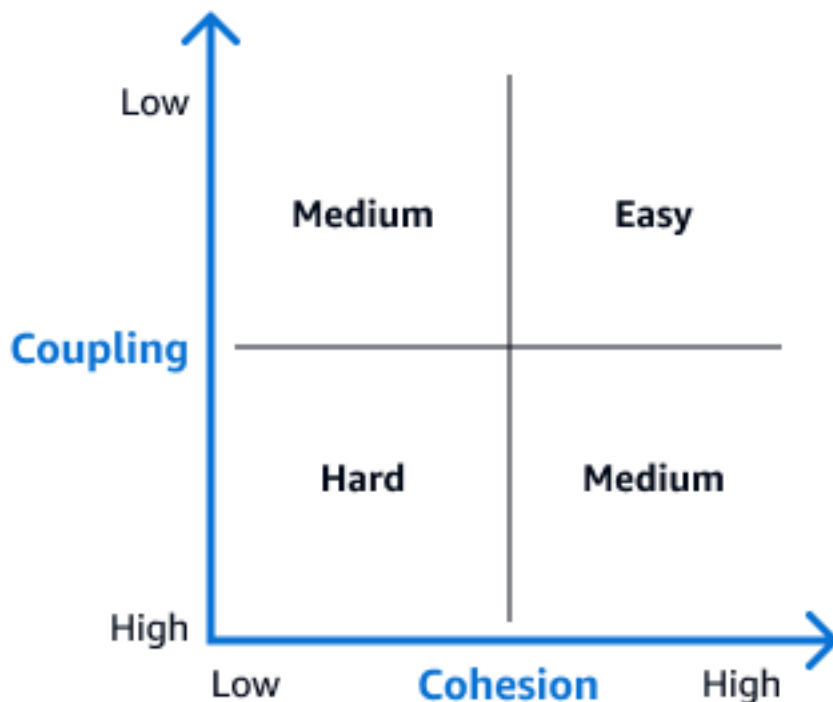
などの AI ツールは、アプリケーションの機能およびドメインの境界を分析することで、ドメインの境界を特定する [vFunction](#) のに役立ちます。vFunction 主にアプリケーションレイヤーを分析しますが、そのインサイトはアプリケーションとデータベースの両方の分解を導き、ビジネスドメインとの整合性をサポートします。

## フェーズ 3: 自己完結型テーブルを特定する

次の 2 つの主要な特性を示すテーブルを探します。

- 高結合 — テーブルの内容は相互に強く関連しています
- 低結合 — 他のテーブルへの依存関係は最小限です。

次の結合結合マトリックスは、各テーブルのデカップリングの難しさを特定するのに役立ちます。このマトリックスの右上の四分円に表示されるテーブルは、分離が最も簡単なため、最初のデカップリング作業の候補として最適です。ER 図では、これらのテーブルには外部キー関係やその他の依存関係はほとんどありません。これらのテーブルを分離したら、より複雑な関係を持つテーブルに進みます。



**Note**

データベース構造は、多くの場合、アプリケーションアーキテクチャを反映しています。データベースレベルで分離しやすいテーブルは、通常、アプリケーションレベルでマイクロサービスに変換しやすいコンポーネントに対応しています。

# データベースからアプリケーションレイヤーへのビジネスロジックの移行

データベースに保存されたプロシージャ、トリガー、関数からアプリケーションレイヤーサービスにビジネスロジックを移行することは、モノリシックデータベースを分解するための重要なステップです。この変換により、サービスの自律性が向上し、メンテナンスが簡素化され、スケーラビリティが向上します。このセクションでは、データベースロジックの分析、移行戦略の計画、ビジネス継続性を維持しながらの変革の実装に関するガイダンスを提供します。また、効果的なロールバック計画の策定についても説明します。

このセクションは、以下のトピックで構成されます。

- [フェーズ 1: ビジネスロジックの分析](#)
- [フェーズ 2: ビジネスロジックの分類](#)
- [フェーズ 3: ビジネスロジックの移行](#)
- [ビジネスロジックのロールバック戦略](#)

## フェーズ 1: ビジネスロジックの分析

モノリシックデータベースをモダナイズする場合は、まず既存のデータベースロジックの包括的な分析を行う必要があります。このフェーズでは、主に次の 3 つのカテゴリに焦点を当てます。

- ストアドプロシージャには、多くの場合、データ操作ロジック、ビジネスルール、検証チェック、計算など、重要なビジネスオペレーションが含まれます。アプリケーションビジネスロジックのコアコンポーネントとして、慎重に分解する必要があります。たとえば、金融機関のストアドプロシージャは、インタレスト計算、アカウント調整、コンプライアンスチェックを処理する場合があります。
- トリガーは、監査証跡、データ検証、計算、およびクロステーブル整合性を処理する主要なデータベースコンポーネントです。たとえば、小売組織はトリガーを使用して注文処理システム全体のインベントリ更新を管理する場合があります。これは、自動データベースオペレーションの複雑さを示しています。
- データベースの関数は、主にデータ変換、計算、ルックアップオペレーションを管理します。多くの場合、複数のプロシージャやアプリケーションに埋め込まれます。たとえば、医療組織は機能を使用して患者データを正規化したり、医療コードを検索したりできます。

各カテゴリは、データベースレイヤー内に埋め込まれているビジネスロジックのさまざまな側面を表します。アプリケーションレイヤーに移行するには、各を慎重に評価して計画する必要があります。

この分析フェーズでは、通常、お客様は3つの大きな課題に直面します。まず、複雑な依存関係は、ネストされたプロシージャ呼び出し、クロススキーマ参照、および暗黙的なデータ依存関係によって発生します。次に、特にマルチステップトランザクションを処理し、分散システム全体でデータの一貫性を維持する場合、トランザクション管理が重要になります。第3に、パフォーマンスに関する考慮事項を慎重に評価する必要があります。特に、バッチ処理オペレーション、一括データ更新、およびデータに近づくことで現在メリットを得ているリアルタイム計算については注意が必要です。

これらの課題に効果的に対処するには、初期分析に [AWS Schema Conversion Tool \(AWS SCT\)](#) を使用し、詳細な依存関係マッピングツールを使用できます。このアプローチは、データベースロジックの全範囲を理解し、分解中のビジネス継続性を維持する包括的な移行戦略を作成するのに役立ちます。

これらのコンポーネントと課題を徹底的に理解することで、モダナイゼーションジャーニーをよりの確に計画し、マイクロサービスベースのアーキテクチャへの移行中にどの要素を優先すべきかについて、情報に基づいた意思決定を行うことができます。

データベースコードコンポーネントを分析するときは、ストアードプロシージャ、トリガー、および関数ごとに包括的なドキュメントを作成します。まず、実装するビジネスルールを含め、その目的とコア機能を明確に記述します。すべての入力パラメータと出力パラメータを詳しく説明し、それらのデータ型と有効な範囲を書き留めます。他のデータベースオブジェクト、外部システム、ダウンストリームプロセスへの依存関係をマッピングします。データの整合性を維持するために、トランザクションの境界と分離要件を明確に定義します。応答時間の要件やリソース使用率パターンなど、期待されるパフォーマンスを文書化します。最後に、使用状況パターンを分析して、ピーク時の負荷、実行頻度、重要なビジネス期間を把握します。

## フェーズ 2: ビジネスロジックの分類

データベースを効果的に分解するには、複雑さ、ビジネスへの影響、依存関係、使用パターン、移行の難易度といった主要な側面にわたってデータベースロジックを体系的に分類する必要があります。この分類は、高リスクコンポーネントの特定、テスト要件の決定、移行の優先順位の確立に役立ちます。たとえば、ビジネスへの影響が大きく、頻繁に使用する複雑なストアードプロシージャには、慎重な計画と広範なテストが必要です。ただし、依存関係を最小限に抑えたシンプルでほとんど使用されない関数は、早期移行フェーズに適している場合があります。

この構造化されたアプローチは、システムの安定性を維持しながら、ビジネスの中断を最小限に抑えるバランスの取れた移行ロードマップを作成します。これらの相互関係を理解することで、分解作業のシーケンスを改善し、リソースを適切に割り当てることができます。

## フェーズ 3: ビジネスロジックの移行

ビジネスロジックを分析して分類したら、それを移行します。モノリシックデータベースからビジネスロジックを移行するときは、データベースロジックをアプリケーションレイヤーに移動するか、ビジネスロジックをマイクロサービスの一部である別のデータベースに移動するという 2 つのアプローチがあります。

ビジネスロジックをアプリケーションに移行すると、データベーステーブルにはデータのみが保存され、データベースにはビジネスロジックは含まれません。これは推奨されるアプローチです。[Ispirer](#) または [Amazon Q Developer](#) や などの生成 AI ツールを使用して [Kiro](#)、Java への変換など、アプリケーションレイヤーのデータベースビジネスロジックを変換できます。詳細については、「[Migrate business logic from database to application for faster innovation and flexibility](#)」(AWS ブログ記事) を参照してください。

ビジネスロジックを別のデータベースに移行する場合は、[AWS Schema Conversion Tool \(AWS SCT\)](#) を使用して既存のデータベーススキーマとコードオブジェクトをターゲットデータベースに変換できます。Amazon [DynamoDB](#)、[Amazon Aurora](#)、[Amazon Redshift](#) などの専用 AWS データベースサービスをサポートしています。包括的な評価レポートと自動変換機能を提供すること AWS SCT で、移行プロセスを合理化し、新しいデータベース構造の最適化に集中してパフォーマンスとスケーラビリティを向上させることができます。モダナイゼーションプロジェクトを進めるにつれて、AWS SCT は段階的なアプローチをサポートするために増分変換を処理できるため、データベース変換の各ステップを検証して微調整できます。

## ビジネスロジックのロールバック戦略

分解戦略の 2 つの重要な側面は、下位互換性を維持し、包括的なロールバック手順を実装することです。これらの要素は連携して、移行期間中のオペレーションを保護します。このセクションでは、分解プロセス中に互換性を管理し、潜在的な問題から保護する効果的な緊急ロールバック機能を確立する方法について説明します。

### 下位互換性を維持する

データベースの分解中、移行をスムーズにするには、下位互換性を維持することが不可欠です。新しい機能を徐々に実装しながら、既存のデータベース手順を一時的に実施してください。バージョン管

理を使用して、すべての変更を追跡し、複数のデータベースバージョンを同時に管理します。ソースシステムとターゲットシステムの両方を確実に動作させる必要がある、長期間の共存期間を計画します。これにより、レガシーコンポーネントを廃止する前に、新しいシステムをテストおよび検証する時間を確保できます。このアプローチは、ビジネスの中断を最小限に抑え、必要に応じてロールバックのための安全ネットを提供します。

## 緊急ロールバック計画

包括的なロールバック戦略は、データベースを安全に分解するために不可欠です。コードに機能フラグを実装して、アクティブなビジネスロジックのバージョンを制御します。これにより、デプロイを変更することなく、新しい実装と元の実装を即座に切り替えることができます。このアプローチは、移行をきめ細かく制御し、問題が発生した場合に迅速にロールバックするのに役立ちます。元のロジックを検証済みのバックアップとして保持し、トリガー、責任、復旧ステップを指定する詳細なロールバック手順を維持します。

これらのロールバックシナリオは、さまざまな条件下で定期的にテストして有効性を検証し、チームが緊急手順に精通していることを確認します。機能フラグは、特定のユーザーグループまたはトランザクションの新機能を選択的に有効にすることで、段階的なロールアウトも有効にします。これにより、移行中のリスク軽減のための追加のレイヤーが提供されます。

# データベース分解中のテーブル関係のデカップリング

このセクションでは、モノリシックデータベースの分解中に複雑なテーブル関係と JOIN オペレーションを分解するためのガイドを提供します。テーブル結合は、関連する列に基づいて 2 つ以上のテーブルの行を結合します。これらの関係を分離する目標は、マイクロサービス間でデータの整合性を維持しながら、テーブル間の高い結合を減らすことです。

このセクションは、以下のトピックで構成されます。

- [非正規化戦略](#)
- [Reference-by-key戦略](#)
- [CQRS パターン](#)
- [イベントベースのデータ同期](#)
- [テーブル結合に代わる方法の実装](#)
- [シナリオベースの例](#)

## 非正規化戦略

非正規化は、テーブル間でデータを結合または複製することで、意図的に冗長性を導入するデータベース設計戦略です。大規模なデータベースを小さなデータベースに分割する場合、サービス間で一部のデータを複製することは理にかなっている場合があります。例えば、名前や E メールアドレスなどの基本的な顧客の詳細をマーケティングサービスと注文サービスの両方に保存することで、サービス間の継続的な検索が不要になります。マーケティングサービスでは、キャンペーンターゲティングに顧客の好みと連絡先情報が必要になる場合がありますが、注文サービスでは注文処理と通知に同じデータが必要です。これによりデータの冗長性がある程度生じる一方で、サービスのパフォーマンスと独立性が大幅に向上するため、マーケティングチームはカスタマーサービスのリアルタイム検索に依存することなくキャンペーンを運用できます。

非正規化を実装するときは、データアクセスパターンを注意深く分析して識別する、頻繁にアクセスされるフィールドに焦点を当てます。Oracle AWR レポートやなどのツールを使用して `pg_stat_statements`、一般的に一緒に取得されるデータを把握できます。ドメインエキスパートは、自然データのグループ化に関する貴重なインサイトを提供することもできます。非正規化は all-or-nothing のアプローチではなく、システムパフォーマンスを向上させたり、複雑な依存関係を減らしたりする重複データのみであることに注意してください。

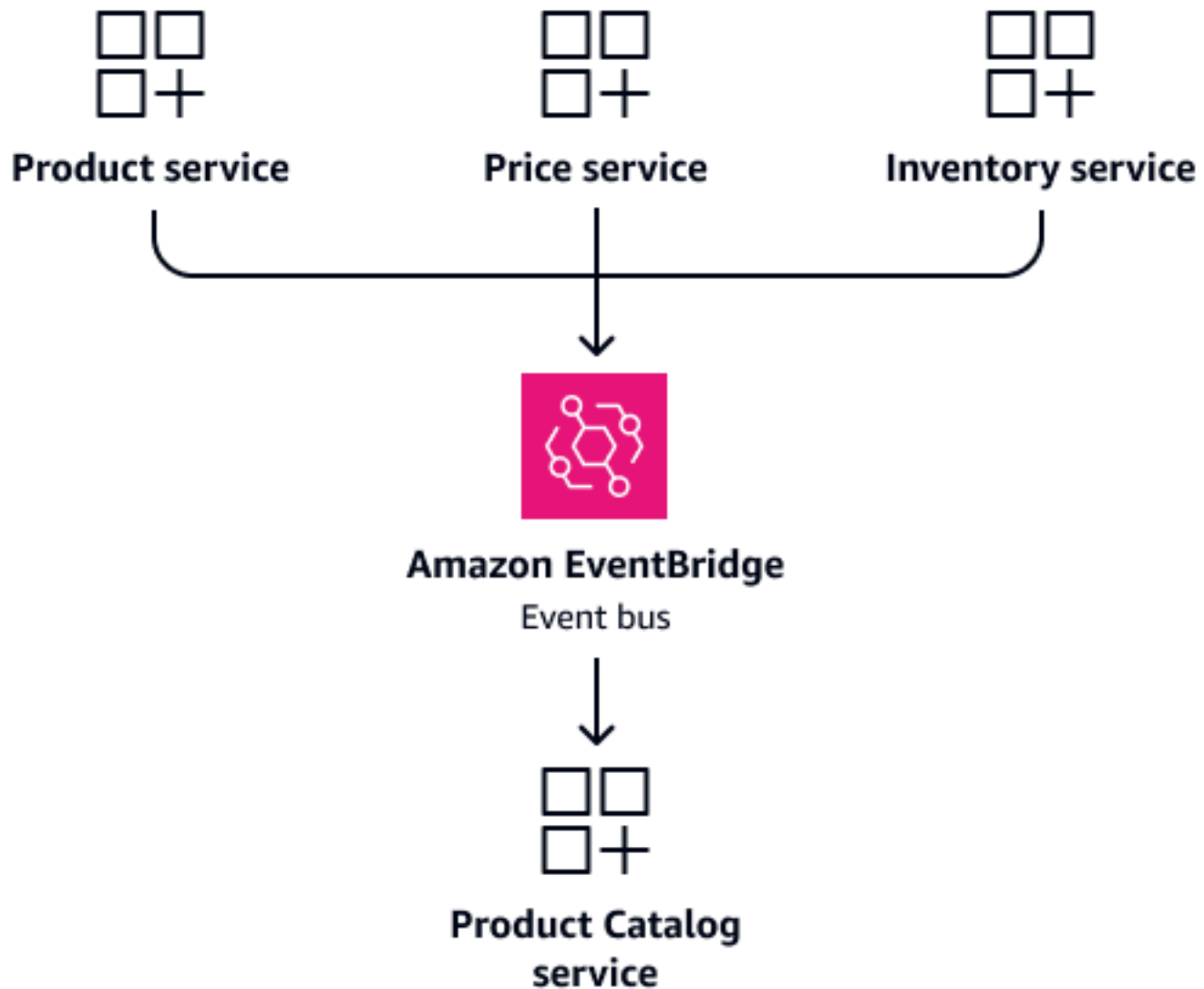
## Reference-by-key戦略

reference-by-key戦略は、実際の関連データを保存するのではなく、一意のキーを通じてエンティティ間の関係を維持するデータベース設計パターンです。従来の外部キー関係の代わりに、最新のマイクロサービスは多くの場合、関連するデータの一意の識別子のみを保存します。たとえば、注文テーブルにすべての顧客詳細を保持するのではなく、注文サービスは顧客 ID のみを保存し、必要に応じて API コールを通じて追加の顧客情報を取得します。このアプローチは、関連データへのアクセスを確保しながら、サービスの独立性を維持します。

## CQRS パターン

コマンドクエリ責任分離 (CQRS) パターンは、データストアの読み取りオペレーションと書き込みオペレーションを分離します。このパターンは、高性能要件を持つ複雑なシステム、特に非対称読み取り/書き込みロードを持つシステムで特に役立ちます。アプリケーションが複数のソースからのデータを頻繁に組み合わせる必要がある場合は、複雑な結合の代わりに専用の CQRS モデルを作成できます。たとえば、すべてのリクエストで Product、Pricing および Inventory テーブルを結合するのではなく、必要なデータを含む統合 Product Catalog テーブルを維持します。このアプローチの利点は、追加のテーブルのコストを上回る可能性があります。

Product、Price、および Inventory のサービスが頻繁に製品情報を必要とするシナリオを考えてみましょう。共有テーブルに直接アクセスするようにこれらのサービスを設定する代わりに、専用 Product Catalog サービスを作成します。このサービスは、統合された製品情報を含む独自のデータベースを保持します。製品関連のクエリの信頼できる単一のソースとして機能します。製品の詳細、価格、またはインベントリレベルが変更されると、各サービスはイベントを発行して Product Catalog サービスを更新できます。これにより、サービスの独立性を維持しながらデータの一貫性が得られます。次の図は、[Amazon EventBridge](#) がイベントバスとして機能するこの設定を示しています。



で説明されているように [イベントベースのデータ同期](#)、次のセクションでは、イベントを通じて CQRS モデルを更新し続けます。製品の詳細、価格、またはインベントリレベルが変更されると、各サービスはイベントを発行します。Product Catalog サービスはこれらのイベントをサブスクライブし、統合ビューを更新します。これにより、複雑な結合なしで高速読み取りが可能になり、サービスの独立性が維持されます。

## イベントベースのデータ同期

イベントベースのデータ同期は、データへの変更がキャプチャされ、イベントとして伝達されるパターンです。これにより、さまざまなシステムまたはコンポーネントが同期されたデータ状態を維持できます。データが変更されると、関連するすべてのデータベースをすぐに更新する代わりに、イベントを発行して、サブスクライブされたサービスに通知します。例えば、顧客が Customer サービス

スの配送先住所を変更すると、CustomerUpdatedイベントは各Orderサービスのスケジュールに従ってDeliveryサービスとサービスの更新を開始します。このアプローチは、リジッドテーブル結合を柔軟でスケラブルなイベント駆動型更新に置き換えます。一部のサービスは一時的に古いデータを持っているかもしれませんが、トレードオフはシステムのスケラビリティとサービスの独立性の向上です。

## テーブル結合に代わる方法の実装

通常、データベースの分解は移行と検証が簡単なため、読み取りオペレーションで開始します。読み取りパスが安定したら、より複雑な書き込みオペレーションに取り組みます。重要で高性能な要件については、[CQRS パターン](#)の実装を検討してください。読み取りには最適化された別のデータベースを使用し、書き込みには別のデータベースを使用します。

クロスサービス呼び出しの再試行ロジックを追加し、適切なキャッシュレイヤーを実装することで、回復力のあるシステムを構築します。サービスのやり取りを注意深くモニタリングし、データ整合性の問題に関するアラートを設定します。最終目標は、どこにいても完全な一貫性ではありません。ビジネスニーズに適したデータ精度を維持しながら、優れたパフォーマンスを発揮する独立したサービスを作成しています。

マイクロサービスの分離により、データ管理に次の新しい複雑さが導入されます。

- データは分散されます。データは独立したサービスによって管理される個別のデータベースに存在するようになりました。
- サービス間でのリアルタイムの同期は実用的ではないことが多く、結果整合性モデルが必要になります。
- 1つのデータバーストランザクション内で以前に発生したオペレーションが、複数のサービスにまたがるようになりました。

これらの課題に対処するには、以下を実行します。

- イベント駆動型アーキテクチャの実装 – メッセージキューとイベント発行を使用して、サービス間でデータ変更を伝達します。詳細については、「[サーバーレスランドでのイベント駆動型アーキテクチャの構築](#)」を参照してください。
- Saga オーケストレーションパターンを採用する – このパターンは、分散トランザクションを管理し、サービス間でデータの整合性を維持するのに役立ちます。詳細については、AWS ブログの「[Saga オーケストレーションパターンを使用してサーバーレス分散アプリケーションを構築する](#)」を参照してください。

- 障害の設計 — 再試行メカニズム、サーキットブレーカー、補償トランザクションを組み込み、ネットワークの問題やサービス障害を処理します。
- バージョンスタンプを使用する – データバージョンを追跡して競合を管理し、最新の更新が適用されていることを確認します。
- 定期的な調整 — 定期的なデータ同期プロセスを実装して、不整合をキャッチして修正します。

## シナリオベースの例

次のスキーマの例には、テーブルとCustomerテーブルの2つのOrderテーブルがあります。

```
-- Customer table
CREATE TABLE customer (
  customer_id INT PRIMARY KEY,
  first_name VARCHAR(100),
  last_name VARCHAR(100),
  email VARCHAR(255),
  phone VARCHAR(20),
  address TEXT,
  created_at TIMESTAMP
);

-- Order table
CREATE TABLE order (
  order_id INT PRIMARY KEY,
  customer_id INT,
  order_date TIMESTAMP,
  total_amount DECIMAL(10,2),
  status VARCHAR(50),
  FOREIGN KEY (customer_id) REFERENCES customers(id)
);
```

以下は、非正規化アプローチを使用する方法の例です。

```
CREATE TABLE order (
  order_id INT PRIMARY KEY,
  customer_id INT, -- Reference only
  customer_first_name VARCHAR(100), -- Denormalized
  customer_last_name VARCHAR(100), -- Denormalized
  customer_email VARCHAR(255), -- Denormalized
  order_date TIMESTAMP,
```

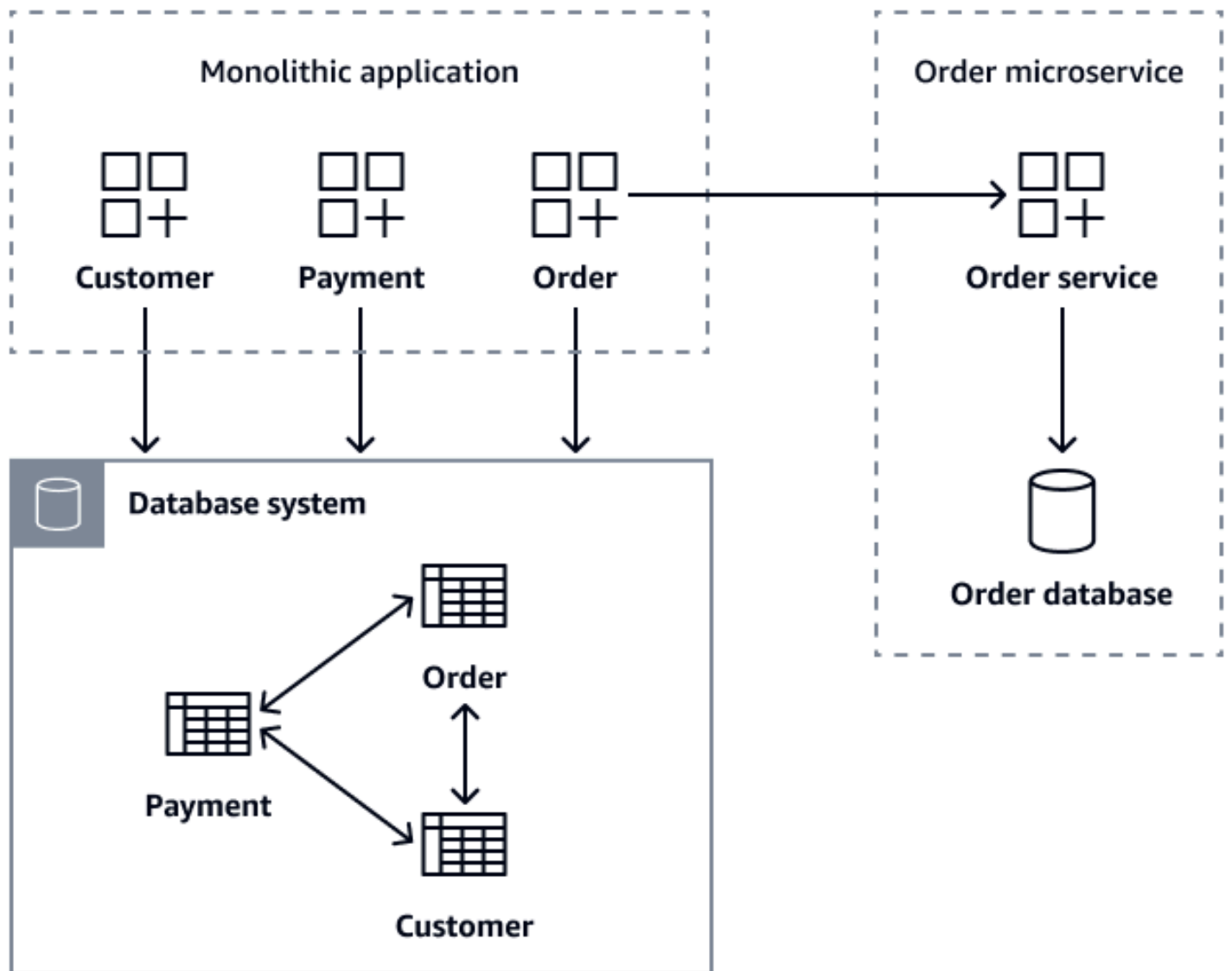
```
total_amount DECIMAL(10,2),
status VARCHAR(50)
);
```

新しいOrderテーブルには、非正規化された顧客名と E メールアドレスがあります。customer\_id が参照され、Customerテーブルに外部キーの制約はありません。この非正規化アプローチの利点は次のとおりです。

- Order サービスは注文履歴を顧客の詳細とともに表示でき、Customerマイクロサービスへの API コールは必要ありません。
- Customer サービスが停止している場合、Orderサービスは完全に機能し続けます。
- 注文処理とレポート作成のクエリの実行が高速化されます。

次の図は、Orderマイクロサービスへの

getOrder(customer\_id)、getOrder(order\_id)、getCustomerOrders(customer\_id)および createOrder(Order order) API 呼び出しを使用して注文データを取得するモノリシックアプリケーションを示しています。



マイクロサービスの移行中、移行時の安全対策としてモノリシックデータベースのOrderテーブルを維持し、レガシーアプリケーションが機能し続けるようにすることができます。ただし、新しい注文関連のオペレーションはすべてOrder、マイクロサービス API を介してルーティングすることが重要です。マイクロサービス API は、バックアップとしてレガシーデータベースに書き込むと同時に、独自のデータベースを維持します。このデュアル書き込みパターンは、セーフティネットを提供します。これにより、システムの安定性を維持しながら、段階的な移行が可能になります。すべてのお客様が新しいマイクロサービスに正常に移行したら、モノリシックデータベースのレガシーOrderテーブルを廃止できます。モノリシックアプリケーションとそのデータベースを個別のCustomerOrderマイクロサービスに分解した後、データの一貫性を維持することが主な課題になります。

# データベース分解のベストプラクティス

モノリシックデータベースを分解する場合、組織は進捗状況を追跡し、システム知識を維持し、新たな課題に対処するための明確なフレームワークを確立する必要があります。このセクションでは、分解の成功を測定し、重要なドキュメントを維持し、継続的な改善プロセスを実装し、一般的な課題に対処するためのベストプラクティスについて説明します。これらのガイドラインを理解して従うと、データベースの分解作業によって、運用上の中断や技術的負債を最小限に抑えながら、意図した利点が得られるようになります。

このセクションは、以下のトピックで構成されます。

- [成功の測定](#)
- [ドキュメント要件](#)
- [継続的改善戦略](#)
- [データベース分解における一般的な課題を克服する](#)

## 成功の測定

技術的メトリクス、運用メトリクス、ビジネスメトリクスを組み合わせ、分解の成功を追跡します。技術的には、クエリの応答時間、システムの稼働時間の改善、デプロイ頻度の増加をモニタリングします。運用上、インシデントの削減、問題解決の速度、リソース使用率の向上を測定します。開発では、機能実装の速度、リリースサイクルの加速、チーム間の依存関係の削減を追跡します。ビジネスへの影響により、運用コストの削減、time-to-marketの短縮、顧客満足度の向上につながります。これらのメトリクスは、多くの場合、スコープフェーズ中に定義されます。詳細については、このガイドの「[データベース分解の範囲と要件の定義](#)」を参照してください。

## ドキュメント要件

明確なサービス境界、データフロー、インターフェイス仕様を使用して、up-to-dateシステムアーキテクチャドキュメントを維持します。アーキテクチャ決定レコード (ADRs) を使用して、コンテキスト、結果、考慮すべき代替案など、主要な技術的決定をキャプチャします。たとえば、特定のサービスが最初に分離された理由や、特定のデータ整合性のトレードオフがどのように行われたかを文書化します。

毎月のアーキテクチャレビューをスケジュールして、パフォーマンスの傾向、セキュリティコンプライアンス、サービス間の依存関係などの主要なメトリクスを通じてシステムの状態を評価します。

統合の課題と運用上の問題に関する開発チームからのフィードバックを含めます。この定期的なレビューサイクルは、新たな問題を早期に特定し、分解作業がビジネス目標に沿っていることを検証するのに役立ちます。

## 継続的改善戦略

データベースの分解は、1 回限りのプロジェクトではなく、反復プロセスとして扱います。システムパフォーマンスメトリクスとサービスインタラクションをモニタリングして、最適化の機会を特定します。四半期ごとに、運用上の影響とメンテナンスコストに基づいて、技術的負債への対応を優先します。たとえば、頻繁に実行されるデータベースオペレーションを自動化し、モニタリングカバレッジを強化し、学習したパターンに基づいてデプロイ手順を絞り込みます。

## データベース分解における一般的な課題を克服する

パフォーマンスの最適化には、多面的なアプローチが必要です。サービス境界に戦略的キャッシュを実装し、実際の使用状況に基づいてクエリパターンを最適化し、主要なメトリクスを継続的にモニタリングします。傾向を分析し、介入の明確なしきい値を設定することで、パフォーマンスのボトルネックに積極的に対処します。

データ整合性の課題には、アーキテクチャ上の慎重な選択が必要です。クロスサービス更新のイベント駆動型パターンを実装し、複雑なトランザクションには Saga オーケストレーションパターンを使用します。明確なサービス境界を定義し、ビジネス要件が許す限り、結果整合性を受け入れます。分解を成功させるには、一貫性とサービスの自律性のバランスが不可欠です。

運用上の優秀性には、サービス全体でルーチンタスクと標準化された手順の自動化が必要です。明確なアラートしきい値で包括的なモニタリングを維持し、新しいパターンとツールの定期的なチームトレーニングに投資します。このオペレーションへの体系的なアプローチは、複雑さを管理しながら、信頼性の高いサービス提供を促進します。

# データベース分解に関するよくある質問

この包括的なよくある質問セクションでは、データベース分解プロジェクトを実行する際に組織が直面する最も一般的な質問と課題について説明します。これらの質問は、初期の範囲と要件の定義からストアドプロシージャの移行まで、チームがデータベースのモダナイゼーションジャーニーを正常に進めるのに役立つ実用的なインサイトと戦略的アプローチを提供します。計画段階にある場合でも、分解戦略をすでに実行している場合でも、これらの回答は一般的な落とし穴を回避し、最適な結果を得るためのベストプラクティスを実装するのに役立ちます。

このセクションは、以下のトピックで構成されます。

- [範囲と要件の定義に関するFAQs](#)
- [データベースアクセスの制御に関するFAQs](#)
- [結合と結合の分析に関するFAQs](#)
- [ビジネスロジックのアプリケーションレイヤーへの移行に関するFAQs](#)

## 範囲と要件の定義に関するFAQs

このガイドの [データベース分解の範囲と要件の定義](#) セクションでは、インタラクションを分析し、依存関係をマッピングし、成功基準を確立する方法について説明します。このよくある質問セクションでは、プロジェクトの境界の確立と管理に関する重要な質問について説明します。技術的な制約が不明確であるか、部門のニーズが競合しているか、ビジネス要件が進化しているかにかかわらず、これらのFAQsはバランスの取れたアプローチを維持するための実践的なガイダンスを提供します。

このセクションでは、以下の質問について説明します。

- [初期スコープ定義はどの程度詳細にする必要がありますか？](#)
- [プロジェクトの開始後に追加の依存関係が見つかった場合はどうなりますか？](#)
- [要件が競合するさまざまな部門のステークホルダーにどのように対処すればよいですか？](#)
- [ドキュメントが古くなっている場合、技術的な制約を評価する最善の方法は何ですか？](#)
- [当面のビジネスニーズと長期的な技術目標のバランスを取るにはどうすればよいですか？](#)
- [サイレントステークホルダーから重要な要件が欠落していないことを確認するにはどうすればよいですか？](#)
- [これらの推奨事項はモノリシックメインフレームデータベースに適用されますか？](#)

## 初期スコープ定義はどの程度詳細にする必要がありますか？

顧客のニーズから逆算して、検出の柔軟性を維持しながら、システムの境界と重要な依存関係を特定するのに十分な詳細でプロジェクトの範囲を定義します。システムインターフェイス、主要な利害関係者、主要な技術的制約など、重要な要素をマッピングします。測定可能な値を提供するシステムの境界付きでリスクの低い部分を選択して、小規模から始めます。このアプローチは、チームがより複雑なコンポーネントに取り組む前に戦略を学習して調整するのに役立ちます。

分解作業を推進する重要なビジネス要件を文書化しますが、実装中に変更される可能性のある詳細を過剰に指定することは避けてください。このバランスの取れたアプローチにより、チームはモダナイゼーションジャーニー中に出現する新しいインサイトや課題に適応しながら、明確に前進できます。

## プロジェクトの開始後に追加の依存関係が見つかった場合はどうなりますか？

プロジェクトの進行に伴い、追加の依存関係を明らかにすることが予想されます。ライブ依存関係ログを維持し、定期的なスコープレビューを実施して、タイムラインとリソースへの影響を評価します。明確な変更管理プロセスを実装し、予期しない検出を処理するためにプロジェクト計画にバッファ時間を含めます。目標は、変更を防ぐことではなく、効果的に管理することです。これにより、チームはプロジェクトの勢いを維持しながら迅速に適応できます。

## 要件が競合するさまざまな部門のステークホルダーにどのように対処すればよいですか？

ビジネス価値とシステムへの影響に基づく明確な優先順位付けを通じて、競合する部門の要件に対処します。主要な意思決定を推進し、競合を迅速に解決するためのエグゼクティブスポンサーシップを確保します。定期的なステークホルダー調整会議をスケジュールして、トレードオフについて話し合い、透明性を維持します。すべての決定とその根拠を文書化して、明確なコミュニケーションを促進し、プロジェクトの勢いを維持します。部門別の好みではなく、定量化可能なビジネス上の利点に焦点を当てて議論します。

## ドキュメントが古くなっている場合、技術的な制約を評価する最善の方法は何ですか？

ドキュメントが不十分な場合は、従来の分析と最新の AI ツールを組み合わせてください。大規模言語モデル (LLMs) を使用してコードリポジトリ、ログ、既存のドキュメントを分析し、パターンと潜在的な制約を特定します。経験豊富な開発者とデータベースアーキテクトにインタビューして AI の

検出結果を検証し、文書化されていない制約を発見します。AI 機能を強化したモニタリングツールをデプロイして、システム動作を観察し、潜在的な問題を予測します。

仮定を検証する小規模な技術実験を作成します。AI を活用したテストツールを使用して、プロセスを高速化できます。AI 支援の更新を通じて継続的に拡張できる結果をナレッジベースに文書化します。複雑な分野に対象分野のエキスパートを関与させ、AI ペアプログラミングツールを使用して分析とドキュメント作成の取り組みを加速することを検討してください。

## 当面のビジネスニーズと長期的な技術目標のバランスを取るにはどうすればよいですか？

当面のビジネスニーズを長期的な技術目標に合わせる、段階的なプロジェクトロードマップを作成します。ステークホルダーの信頼を構築できるように、具体的な価値を早期に提供するクイックウィン特定します。分解を明確なマイルストーンに分割します。各は、アーキテクチャ目標に向かって進みながら、測定可能なビジネス上の利点を提供する必要があります。定期的なロードマップのレビューと調整を通じて、緊急のビジネスニーズに対応するための柔軟性を維持します。

## サイレントステークホルダーから重要な要件が欠落していないことを確認するにはどうすればよいですか？

ダウンストリームシステム所有者や間接ユーザーなど、組織全体のすべての潜在的な利害関係者をマッピングします。構造化されたインタビュー、ワークショップ、定期的なレビューセッションを通じて、複数のフィードバックチャネルを作成します。proof-of-conceptsとプロトタイプを構築して、要件を具体的なものにし、有意義な議論を促します。たとえば、システムの依存関係を示すシンプルなダッシュボードでは、多くの場合、最初は明らかではなかった隠れた利害関係者や要件が明らかになります。

音声ステークホルダーとクワイエットステークホルダーの両方と定期的に検証セッションを行い、すべての視点がキャプチャされていることを確認します。重要なインサイトは、多くの場合、計画会議で最も大きな声ではなく、日常業務に最も近い人から得られます。

## これらの推奨事項はモノリシックメインフレームデータベースに適用されますか？

このガイドで説明されている方法は、モノリシックメインフレームデータベースの分解にも適用されます。これらのデータベースの主な課題は、さまざまな利害関係者の要件を管理することです。このガイドのテクノロジーに関する推奨事項は、モノリシックメインフレームデータベースに適用される

場合があります。メインフレームにオンライントランザクション処理 (OLTP) データベースなどのリレーショナルデータベースがある場合、多くの推奨事項が適用されます。ビジネスレポートの生成に使用されるデータベースなど、オンライン分析処理 (OLAP) データベースの場合、一部の推奨事項のみが適用されます。

## データベースアクセスの制御に関するFAQs

データベースラッパーサービスパターンを使用したデータベースアクセスの制御については、このガイドの [分解中のデータベースアクセスの制御](#) セクションで説明します。このよくある質問セクションでは、パフォーマンスへの潜在的な影響、既存のストアードプロシージャの処理、複雑なトランザクションの管理、スキーマの変更の監督など、データベースラッパーサービスの導入に関する一般的な懸念事項と質問について説明します。

このセクションでは、以下の質問について説明します。

- [ラッパーサービスが新しいボトルネックにならないか。](#)
- [既存のストアードプロシージャはどうなりますか？](#)
- [移行中にスキーマの変更を管理するにはどうすればよいですか？](#)

### ラッパーサービスが新しいボトルネックにならないか。

データベースラッパーサービスは追加のネットワークホップを追加しますが、影響は通常最小限です。サービスを水平方向にスケールできます。通常、制御されたアクセスの利点はパフォーマンスコストを上回ります。パフォーマンスと保守性の一時的なトレードオフと考える。

### 既存のストアードプロシージャはどうなりますか？

最初は、データベースラッパーサービスはストアードプロシージャをサービスメソッドとして公開できます。時間の経過とともに、ロジックをアプリケーションレイヤーに徐々に移行できるため、テストとバージョン管理が向上します。ビジネスロジックを段階的に移行して、リスクを最小限に抑えます。

### 移行中にスキーマの変更を管理するにはどうすればよいですか？

ラッパーサービスチームを通じてスキーマ変更管理を一元化します。このチームは、すべてのコンシューマーの包括的な可視性を維持する責任があります。このチームは、システム全体の影響について提案された変更を確認し、影響を受けるチームと調整し、制御されたデプロイプロセスを使用して

変更を実装します。たとえば、新しいフィールドを追加する場合、このチームはデフォルト値を実装するか、最初に null を許可することで下位互換性を維持する必要があります。

影響評価、テスト要件、ロールバック手順を含む明確な変更管理プロセスを確立します。データベースのバージョンニングツールを使用して、すべての変更を明確に文書化します。この一元化されたアプローチにより、スキーマの変更によって依存するサービスが中断されるのを防ぎ、システムの安定性を維持します。

## 結合と結合の分析に関するFAQs

データベースの結合と結合を理解して効果的に分析することは、データベースの分解を成功させるために不可欠です。結合と結合については、このガイドの [データベース分解のための結合と結合の分析](#) セクションで説明します。このよくある質問セクションでは、適切なレベルの詳細度の特定、適切な分析ツールの選択、結果の文書化、結合問題の優先順位付けに関する重要な質問について説明します。

このセクションでは、以下の質問について説明します。

- [結合を分析するときに適切なレベルの詳細度を特定するにはどうすればよいですか？](#)
- [データベースの結合と結合を分析するために使用できるツールは何ですか？](#)
- [結合と結合の検出結果を文書化する最善の方法は何ですか？](#)
- [最初に対処すべき結合の問題に優先順位を付けるにはどうすればよいですか？](#)
- [複数のオペレーションにまたがるトランザクションを処理するにはどうすればよいですか？](#)

## 結合を分析するときに適切なレベルの詳細度を特定するにはどうすればよいですか？

データベース関係の広範な分析から始め、体系的にドリルダウンして自然な分離ポイントを特定します。データベース分析ツールを使用して、テーブルレベルの関係、スキーマの依存関係、トランザクションの境界をマッピングします。たとえば、SQL クエリの結合パターンを調べて、データアクセスの依存関係を理解します。トランザクションログを分析して、ビジネスプロセスの境界を特定することもできます。

結合が自然に最小限である領域に焦点を当てます。これらは多くの場合、ビジネスドメインの境界と一致し、最適な分解ポイントを表します。適切なサービス境界を決定するときは、技術的結合 (共有テーブルや外部キーなど) とビジネス結合 (プロセスフローやレポートニーズなど) の両方を検討してください。

## データベースの結合と結合を分析するために使用できるツールは何ですか？

自動ツールと手動分析の組み合わせを使用して、データベースの結合と結合を評価できます。以下のツールは、この評価に役立ちます。

- スキーマ視覚化ツール – [SchemaSpy](#) やなどのツールを使用して ER 図 [pgAdmin](#) を生成できます。これらの図は、テーブルの関係と潜在的な結合ポイントを示しています。
- クエリ分析ツール – [pg\\_stat\\_statements](#) または [pg\\_stat\\_statements](#) を使用して [SQL Server Query Store](#)、頻繁に結合されるテーブルとアクセスパターンを識別できます。
- データベースプロファイリングツール – [Oracle SQL Developer](#) やなどのツールは、クエリのパフォーマンスとデータの依存関係に関するインサイト [MySQL Workbench](#) を提供します。
- 依存関係マッピングツール – [AWS Schema Conversion Tool \(AWS SCT\)](#) は、スキーマ関係を視覚化し、緊密に結合されたコンポーネントを識別するのに役立ちます。[vFunction](#) は、アプリケーションの機能境界とドメイン境界を分析することで、ドメイン境界を識別するのに役立ちます。
- トランザクションモニタリングツール – [Oracle Enterprise Manager](#) やなどのデータベース固有のツールを使用して [SQL Server Extended Events](#)、トランザクションの境界を分析できます。
- ビジネスロジック移行ツール – [Amazon Q Developer Inspire](#) やなどの または生成 AI ツールを使用して [Kiro](#)、Java への変換など、アプリケーションレイヤーのデータベースビジネスロジックを変換できます。

これらの自動分析をビジネスプロセスとドメインの知識の手動レビューと組み合わせて、システム結合を完全に理解します。この多面的なアプローチにより、技術的視点とビジネス上の視点の両方が分解戦略で考慮されます。

## 結合と結合の検出結果を文書化する最善の方法は何ですか？

データベースの関係と使用パターンを視覚化する包括的なドキュメントを作成します。以下は、検出結果を記録するために使用できるアセットのタイプです。

- 依存関係マトリックス – テーブルの依存関係をマッピングし、結合の多い領域を強調表示します。
- 関係図 – ER 図を使用して、スキーマ接続と外部キー関係を表示します。
- テーブル使用状況ヒートマップ – テーブル間のクエリ頻度とデータアクセスパターンを視覚化します。
- トランザクションフロー図 – マルチテーブルトランザクションとその境界を文書化します。
- ドメイン境界マップ – ビジネスドメインに基づいて潜在的なサービス境界を概説します。

これらのアーティファクトをドキュメントに結合し、分解の進行に合わせて定期的に更新します。図では、[draw.io](https://draw.io)やなどのツールを使用できます[Lucidchart](https://lucidchart.com)。チームアクセスとコラボレーションを容易にするために、Wikiの実装を検討してください。この多面的なドキュメントアプローチは、システムの結合と結合について明確で共通の理解を提供します。

## 最初に対処すべき結合の問題に優先順位を付けるにはどうすればよいですか？

ビジネス要因と技術的要因のバランスの取れた評価に基づいて、結合の問題に優先順位を付けます。各問題を、ビジネスへの影響(収益やカスタマーエクスペリエンスなど)、技術的リスク(システムの安定性やデータの整合性など)、実装作業、チームの能力に照らして評価します。これらのディメンション全体で各問題を1~5のスコア付けする優先順位付けマトリックスを作成します。このマトリックスは、管理可能なリスクを持つ最も重要な機会を特定するのに役立ちます。

まず、既存のチームの専門知識に沿った、影響の大きい低リスクの変更から始めます。これにより、より複雑な変更に対する組織の自信と勢いを構築できます。このアプローチは、現実的な実行を促進し、ビジネス価値を最大化します。変化するビジネスニーズとチームの能力との整合性を維持するために、優先順位を定期的に見直して調整します。

## 複数のオペレーションにまたがるトランザクションを処理するにはどうすればよいですか？

慎重に設計されたサービスレベルの調整を通じて、複数オペレーショントランザクションを処理します。複雑な分散トランザクションに saga パターンを実装します。それらを、個別に管理できる、より小さく、可逆的なステップに分割します。たとえば、注文処理フローは、インベントリチェック、支払い処理、注文作成の別々のステップに分割され、それぞれに独自の補償メカニズムがあります。

可能な場合は、オペレーションをよりアトミックに再設計し、分散トランザクションの必要性を減らします。分散トランザクションが避けられない場合は、堅牢な追跡および補償メカニズムを実装して、データの一貫性を促進します。トランザクション完了率をモニタリングし、システムの信頼性を維持するために明確なエラー復旧手順を実装します。

## ビジネスロジックのアプリケーションレイヤーへの移行に関するFAQs

データベースからアプリケーションレイヤーにビジネスロジックを移行することは、データベースのモダナイゼーションにおける重要で複雑な側面です。このビジネスロジックの移行については、こ

のガイドの [データベースからアプリケーションレイヤーへのビジネスロジックの移行セクション](#)で説明します。このよくある質問セクションでは、移行の初期候補の選択から複雑なストアドプロシージャやトリガーの処理まで、この移行を効果的に管理するための一般的な質問について説明します。

このセクションでは、以下の質問について説明します。

- [最初に移行するストアドプロシージャを特定するにはどうすればよいですか？](#)
- [ロジックをアプリケーションレイヤーに移動するリスクは何ですか？](#)
- [ロジックをデータベースから遠ざけるときのパフォーマンスを維持するにはどうすればよいですか？](#)
- [複数のテーブルを含む複雑なストアドプロシージャではどうすればよいですか？](#)
- [移行中にデータベーストリガーを処理するにはどうすればよいですか？](#)
- [移行されたビジネスロジックをテストする最善の方法は何ですか？](#)
- [データベースロジックとアプリケーションロジックの両方が存在する場合、移行期間を管理するにはどうすればよいですか？](#)
- [データベースによって以前に管理されていたアプリケーションレイヤーのエラーシナリオを処理するにはどうすればよいですか？](#)

## 最初に移行するストアドプロシージャを特定するにはどうすればよいですか？

まず、低リスクと高学習値の最適な組み合わせを提供するストアドプロシージャを特定します。最小限の依存関係、明確な機能、重要なビジネスへの影響がない手順に焦点を当てます。これらは、チームが信頼を構築し、パターンを確立するのに役立つため、初期移行の理想的な候補になります。たとえば、複雑なトランザクションや重要なビジネスロジックを管理する手順よりも、シンプルなデータオペレーションを処理する手順を選択します。

データベースモニタリングツールを使用して、使用パターンを分析し、アクセス頻度の低い手順を早期候補として特定します。このアプローチは、ビジネスリスクを最小限に抑えながら、後でより複雑な移行に取り組むための貴重な経験を提供します。各手順の複雑さ、ビジネス重要度、依存関係レベルを評価して、優先順位付けされた移行シーケンスを作成します。

## ロジックをアプリケーションレイヤーに移動するリスクは何ですか？

データベースロジックをアプリケーションレイヤーに移動すると、いくつかの重要な課題が生じます。ネットワーク呼び出しの増加、特にデータベース内で以前に処理されたデータ集約型オペレー

ションでは、システムパフォーマンスが低下する可能性があります。トランザクション管理はより複雑になり、分散オペレーション全体でデータの整合性を維持するために慎重な調整が必要です。データ整合性の確保は、特に以前にデータベースレベルの制約に依存していたオペレーションでは困難になります。

移行中の潜在的なビジネス中断とデベロッパーの学習曲線も大きな懸念事項です。徹底的な計画、ステージングされた環境での広範なテスト、重要度の低いコンポーネントから始まる段階的な移行を通じて、これらのリスクを軽減します。堅牢なモニタリングとロールバック手順を実装して、本番環境の問題をすばやく特定して対処します。

## ロジックをデータベースから遠ざけるときのパフォーマンスを維持するにはどうすればよいですか？

頻繁にアクセスされるデータに適したキャッシュメカニズムを実装し、データアクセスパターンを最適化してネットワーク呼び出しを最小限に抑え、一括オペレーションにバッチ処理を使用します。non-time-criticalオペレーションでは、システムの応答性を向上させるために非同期処理を検討してください。

アプリケーションのパフォーマンスメトリクスを注意深くモニタリングし、必要に応じて調整します。たとえば、複数の単一行オペレーションを一括処理に置き換えたり、頻繁に変更されない参照データをキャッシュしたり、クエリパターンを最適化してデータ転送を減らすことができます。定期的なパフォーマンステストとチューニングは、システムが許容可能な応答時間を維持し、保守性とスケーラビリティを向上させるのに役立ちます。

## 複数のテーブルを含む複雑なストアプロシージャではどうすればよいですか？

体系的な分解を通じて、複雑でマルチテーブルのストアプロシージャにアプローチします。まず、より小さく論理的に一貫性のあるコンポーネントに分割し、明確なトランザクションの境界とデータの依存関係を特定します。論理コンポーネントごとにサービスインターフェイスを作成します。これにより、既存の機能を中断することなく、徐々に移行できます。

最小結合コンポーネントから始めて、step-by-stepの移行を実装します。非常に複雑な手順については、よりシンプルなパートを移行しながら、それらをデータベースに一時的に保持することを検討してください。このハイブリッドアプローチは、アーキテクチャ目標に向かって進行しながら、システムの安定性を維持します。移行中はパフォーマンスと機能を継続的にモニタリングし、結果に基づいて戦略を調整する準備をします。

## 移行中にデータベーストリガーを処理するにはどうすればよいですか？

システム機能を維持しながら、データベーストリガーをアプリケーションレベルのイベントハンドラーに変換します。同期トリガーを、非同期オペレーションのメッセージキューを作成するイベント駆動型パターンに置き換えます。メッセージキューに [Amazon Simple Notification Service \(Amazon SNS\)](#) または [Amazon Simple Queue Service \(Amazon SQS\)](#) を使用することを検討してください。監査要件については、アプリケーションレベルのログ記録を実装するか、データベース変更データキャプチャ (CDC) 機能を使用します。

各トリガーの目的と重要度を分析します。一部のトリガーはアプリケーションロジックによって提供され、データ整合性を維持するためにイベントソーシングパターンが必要になる場合があります。監査ログなどの単純なトリガーから始めて、ビジネスルールやデータの整合性を管理する複雑なトリガーに対処します。移行中は慎重にモニタリングし、機能やデータ整合性が失われていないことを確認します。

## 移行されたビジネスロジックをテストする最善の方法は何ですか？

移行されたビジネスロジックをデプロイする前に、多層テストアプローチを実装します。新しいアプリケーションコードのユニットテストから始め、end-to-endのビジネスフローをカバーする統合テストを追加します。古い実装と新しい実装を並行して実行し、結果を比較して機能同等性を検証します。さまざまな負荷条件でパフォーマンステストを実行して、システム動作が以前の機能と一致するか、それを超えていることを確認します。

機能フラグを使用してデプロイを制御し、問題が発生した場合にすぐにロールバックできるようにします。特に重要なワークフローについては、検証にビジネスユーザーを関与させます。初期デプロイ中に主要なメトリクスをモニタリングし、新しい実装へのトラフィックを徐々に増やします。全体で、必要に応じて元のデータベースロジックに戻す機能を維持します。

## データベースロジックとアプリケーションロジックの両方が存在する場合、移行期間を管理するにはどうすればよいですか？

データベースロジックとアプリケーションロジックの両方が使用されている場合は、トラフィックフローを制御する機能フラグを実装し、古い実装と新しい実装をすばやく切り替えることができます。厳格なバージョン管理を維持し、実装とそれぞれの責任の両方を明確に文書化します。両方のシステムの包括的なモニタリングを設定して、不一致やパフォーマンスの問題をすばやく特定します。

必要に応じて元のロジックに戻すことができるように、移行されたコンポーネントごとに明確なロールバック手順を確立します。移行ステータス、潜在的な影響、エスカレーション手順について、すべ

でのステークホルダーと定期的にコミュニケーションを取ります。このアプローチは、システムの安定性とステークホルダーの信頼を維持しながら、徐々に移行するのに役立ちます。

## データベースによって以前に管理されていたアプリケーションレイヤーのエラーシナリオを処理するにはどうすればよいですか？

データベースレベルのエラー処理を堅牢なアプリケーションレイヤーメカニズムに置き換えます。サーキットブレーカーを実装し、一時的な障害に対してロジックを再試行します。補償トランザクションを使用して、分散オペレーション全体でデータの一貫性を維持します。たとえば、支払いの更新が失敗した場合、アプリケーションは定義された制限内で自動的に再試行し、必要に応じて補償アクションを開始する必要があります。

包括的なモニタリングとアラートを設定して問題をすばやく特定し、トラブルシューティングのための詳細な監査ログを維持します。エラー処理を可能な限り自動化するように設計し、人間の介入が必要なシナリオの明確なエスカレーションパスを定義します。この多層アプローチは、データの整合性とビジネスプロセスの継続性を維持しながら、システムの耐障害性を提供します。

## でのデータベース分解の次のステップ AWS

データベースラッパーサービスを通じて初期データベース分解戦略を実装し、ビジネスロジックをアプリケーションレイヤーに移行した後、組織は次の進化を計画する必要があります。このセクションでは、モダナイゼーションジャーニーを継続するための重要な考慮事項の概要を説明します。

このセクションは、以下のトピックで構成されます。

- [データベース分解のための増分戦略](#)
- [分散データベース環境に関する技術的な考慮事項](#)
- [分散アーキテクチャをサポートする組織変更](#)

### データベース分解のための増分戦略

データベースの分解は、3つの異なる段階を経て段階的に進化します。チームはまず、モノリシックデータベースをデータベースラッパーサービスでラップしてアクセスを制御します。その後、レガシーニーズに合わせてプライマリデータベースを維持しながら、データをサービス固有のデータベースに分割し始めます。最後に、完全に独立したサービスデータベースに移行するために、ビジネスロジックの移行を完了します。

このジャーニーを通じて、チームは慎重なデータ同期パターンを実装し、サービス間の一貫性を継続的に検証する必要があります。パフォーマンスモニタリングは、潜在的な問題を早期に特定して対処するために不可欠です。サービスが独立して進化するにつれて、実際の使用パターンに基づいてスキーマを最適化し、時間の経過とともに蓄積される冗長構造を削除する必要があります。

この増分アプローチは、変換プロセス全体でシステムの安定性を維持しながら、リスクを最小限に抑えるのに役立ちます。

### 分散データベース環境に関する技術的な考慮事項

分散データベース環境では、ボトルネックを早期に特定して対処するためにパフォーマンスモニタリングが不可欠です。チームは、パフォーマンスレベルを維持するために、包括的なモニタリングシステムとキャッシュ戦略を実装する必要があります。読み取り/書き込み分割により、システム全体の負荷を効果的に分散できます。

データ整合性には、分散サービス間の慎重なオーケストレーションが必要です。チームは、必要に応じて結果整合性パターンを実装し、明確なデータ所有権の境界を確立する必要があります。堅牢なモニタリングにより、すべてのサービスでデータの整合性が向上します。

さらに、分散アーキテクチャに対応するためにセキュリティを進化させる必要があります。各サービスにはきめ細かなセキュリティコントロールが必要であり、アクセスパターンには定期的なレビューが必要です。この分散環境では、モニタリングと監査の強化が重要になります。

## 分散アーキテクチャをサポートする組織変更

チーム構造は、所有権と説明責任を明確に定義するために、サービスの境界に沿っている必要があります。組織は、新しいコミュニケーションパターンを確立し、チーム内で追加の技術的能力を構築する必要があります。この構造は、既存のサービスのメンテナンスと継続的なアーキテクチャの進化の両方をサポートする必要があります。

分散アーキテクチャを処理するには、運用プロセスを更新する必要があります。チームは、デプロイ手順を変更し、インシデント対応プロセスを適応させ、変更管理プラクティスを進化させて、複数のサービス間で調整する必要があります。

# リソース

以下の追加のリソースとツールは、組織がデータベースの分解を進めるのに役立ちます。

## AWS 規範ガイドランス

- [Oracle データベースを に移行する AWS クラウド](#)
- [Oracle Databaseでの のリプラットフォームオプション AWS](#)
- [クラウド設計パターン、アーキテクチャ、実装](#)

## AWS ブログ投稿

- [イノベーションと柔軟性を高速化するために、ビジネスロジックをデータベースからアプリケーションに移行する](#)

## AWS のサービス

- [AWS Application Migration Service](#)
- [AWS Database Migration Service \(AWS DMS\)](#)
- [Migration Evaluator](#)
- [AWS Schema Conversion Tool \(AWS SCT\)](#)
- [AWS Transform](#)

## その他のツール

- [AppEngine](#) (Dynatrace ウェブサイト)
- [Oracle Automatic Workload Repository](#) (Oracle ウェブサイト)
- [CAST Imaging](#) (CAST ウェブサイト)
- [Kiro](#) (Kiro ウェブサイト)
- [pgAdmin](#) (pgAdmin ウェブサイト)
- [pg\\_stat\\_statements](#) (PostgreSQL ウェブサイト)
- [SchemaSpy](#) (SchemaSpy ウェブサイト)

- [SQL Developer](#) (Oracle ウェブサイト)
- [SQLWays](#) (Ispirer ウェブサイト)
- [vFunction](#) (vFunction ウェブサイト)

## その他のリソース

- [モノリスからマイクロサービス](#) (O'Reilly ウェブサイト)

## 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
<a href="#">メインフレームに関するよくある質問と AI ツール</a>	これらの <a href="#">推奨事項はモノリシックメインフレームデータベースに適用されますか?</a> よくある質問と、データベースの分解中に使用できる AI ツールに関する追加情報を追加しました。	2025 年 10 月 14 日
<a href="#">初版発行</a>	—	2025 年 9 月 30 日

# AWS 規範ガイドの用語集

以下は、AWS 規範ガイド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。

## アベイラビリティゾーン (AZ)

他のアベイラビリティゾーンの障害から 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一つ) 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なら、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。

## ブレイクグラスアクセス

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 ガイドの「[ブレイ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。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で利用できるサービス](#)」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 (ポストン: 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STREAM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利用できる、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。

## I

## 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/サブスクリプション](#) 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 内のすべてのイベント AWS CloudTrail をログに記録すること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ン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

読み取り専用で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)

サービスが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。

## レスポンスコントロール

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「[レスポンスコントロール](#)」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](#)、[レスポンス](#)、[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

のエンドポイントの 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が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と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 익스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。