



インラインコード生成とアシスタントコード生成のための Amazon Q Developer のベストプラクティス

AWS 規範ガイドンス



AWS 規範ガイド: インラインコード生成とアシスタントコード生成のための Amazon Q Developer のベストプラクティス

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 |
| 目的 | 1 |
| デベロッパーワークフロー | 3 |
| 設計と計画 | 4 |
| コーディング | 4 |
| コードレビュー | 5 |
| 統合とデプロイ | 5 |
| 高度な機能 | 6 |
| Amazon Q Developer コード変換 | 6 |
| Amazon Q Developer のカスタマイズ | 6 |
| コーディングのベストプラクティス | 8 |
| オンボーディングのベストプラクティス | 8 |
| Amazon Q Developer の前提条件 | 8 |
| Amazon Q Developer を使用する際のベストプラクティス | 8 |
| Amazon Q Developer のデータプライバシーとコンテンツの使用状況 | 9 |
| コード生成のベストプラクティス | 9 |
| コードレコメンデーションのベストプラクティス | 10 |
| コードの例 | 12 |
| Python の例 | 12 |
| クラスと関数を生成する | 12 |
| ドキュメントコード | 14 |
| アルゴリズムの生成 | 15 |
| ユニットテストを生成する | 17 |
| Java の例 | 18 |
| クラスと関数を生成する | 19 |
| ドキュメントコード | 21 |
| アルゴリズムの生成 | 23 |
| ユニットテストを生成する | 25 |
| チャットの例 | 27 |
| について質問する AWS のサービス | 27 |
| コードの生成 | 28 |
| ユニットテストを生成する | 30 |
| コードの説明 | 32 |
| トラブルシューティング | 35 |

| | |
|--|----|
| 空のコード生成 | 35 |
| 継続的なコメント | 36 |
| インラインコード生成が正しくない | 37 |
| チャットの結果が不十分 | 42 |
| よくある質問 | 46 |
| Amazon Q Developer とは? | 46 |
| Amazon Q Developer にアクセスするにはどうすればよいですか? | 46 |
| Amazon Q Developer はどのプログラミング言語をサポートしていますか? | 46 |
| より良いコード生成のために Amazon Q Developer にコンテキストを提供するにはどうすればよいですか? | 46 |
| Amazon Q Developer でのインラインコード生成が正確でない場合はどうすればよいですか? | 47 |
| Amazon Q Developer チャット機能を使用してコードの生成とトラブルシューティングを行うにはどうすればよいですか? | 47 |
| Amazon Q Developer を使用する際のベストプラクティスにはどのようなものがありますか? | 47 |
| Amazon Q Developer をカスタマイズして、自分のコードに基づいてレコメンデーションを生成できますか? | 47 |
| 次のステップ | 48 |
| リソース | 49 |
| AWS ブログ | 49 |
| AWS ドキュメント | 49 |
| AWS ワークショップ | 49 |
| 寄稿者 | 50 |
| ドキュメント履歴 | 51 |
| 用語集 | 52 |
| # | 52 |
| A | 53 |
| B | 55 |
| C | 57 |
| D | 60 |
| E | 64 |
| F | 67 |
| G | 68 |
| H | 69 |
| I | 71 |

| | |
|---------|-----|
| L | 73 |
| M | 74 |
| O | 78 |
| P | 81 |
| Q | 84 |
| R | 84 |
| S | 87 |
| T | 91 |
| U | 92 |
| V | 93 |
| W | 93 |
| Z | 94 |
| | XCV |

インラインコード生成とアシスタントコード生成のための Amazon Q Developer のベストプラクティス

アマゾン ウェブ サービス ([寄稿者](#))

2024 年 8 月 ([ドキュメント履歴](#))

従来、デベロッパーは独自の専門知識、ドキュメント、さまざまなソースからのコードスニペットを利用してコードを記述し、維持してきました。これらの方法は業界でうまく機能していますが、時間がかかり、人為的ミスが発生しやすくなり、非効率性や潜在的なバグにつながる可能性があります。

ここでは、Amazon Q Developer が開発者のジャーニーを改善します。Amazon Q Developer は、インテリジェントなコード生成とレコメンデーションを提供することでコード開発タスクを加速するように設計された強力な AWS 生成 AI を活用したアシスタントです。

ただし、他の新しいテクノロジーと同様に、課題が発生する可能性があります。非現実的な期待、オンボーディングの問題、不正確なコード生成のトラブルシューティング、Amazon Q 機能の適切な使用は、デベロッパーが直面する可能性のある一般的な障害です。この包括的なガイドでは、これらの課題に対処し、最も広く採用されているプログラミング言語の 2 つであると専用の実際のシナリオ、詳細なベストプラクティス、トラブルシューティング Java、実際のコード例を提供します。

このガイドでは、Amazon Q Developer を使用して、次のようなコード開発タスクを実行することに重点を置いています。

- コード補完 – デベロッパーコードとしてインライン提案をリアルタイムで生成します。
- コードの改善とアドバイス – ソフトウェア開発について話し合い、自然言語で新しいコードを生成し、既存のコードを改善します。

目的

このガイドの目的は、Amazon Q Developer の新規ユーザーまたは継続的なユーザーであるデベロッパーをサポートし、毎日のコーディングタスクでサービスを正常に使用できるようにすることです。開発チームのマネージャーは、このガイドを読むことでもメリットが得られます。

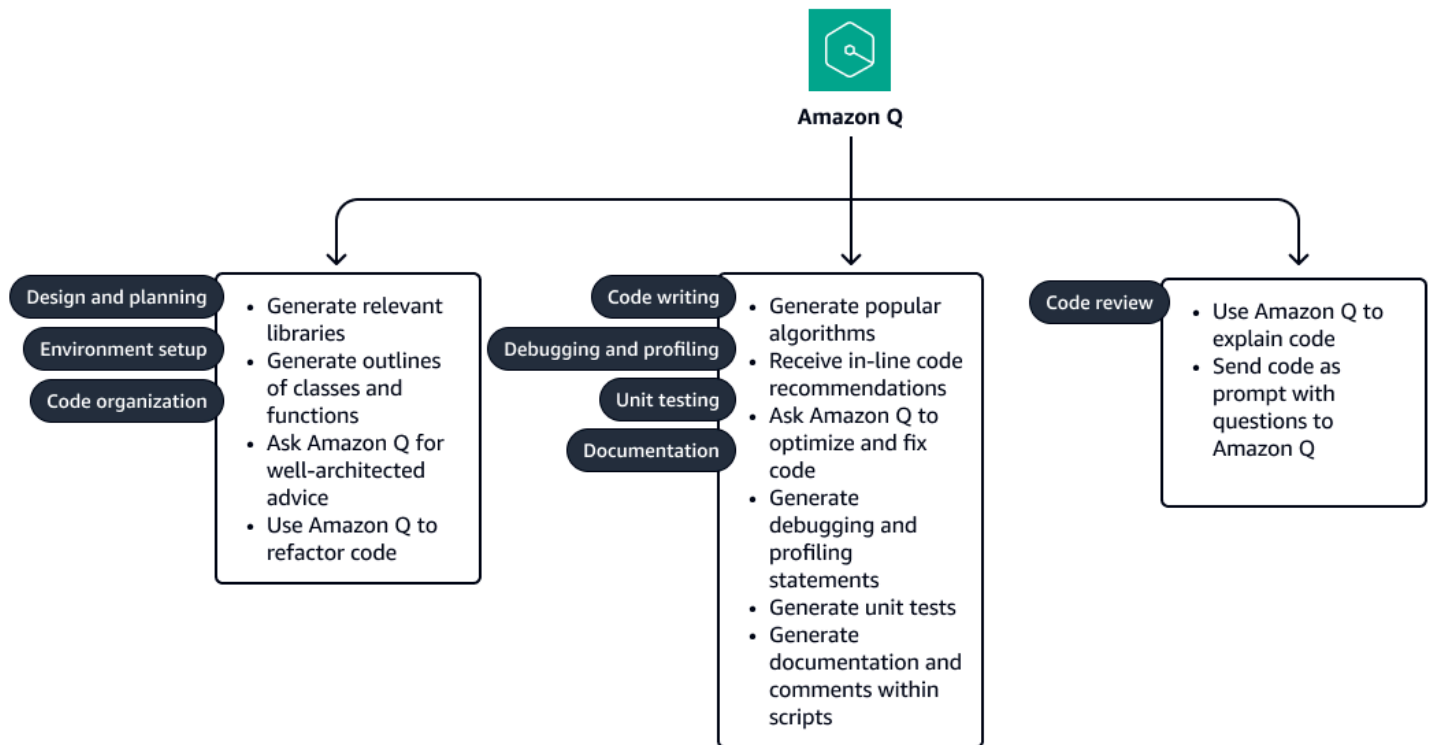
このガイドでは、Amazon Q Developer の使用に関する以下のインサイトを提供します。

- コード開発のための Amazon Q Developer の効果的な使用方法を理解する

- Amazon Q Developer を [開発者のワークフロー](#) に統合するためのベストプラクティスを示します。
- [コード生成](#) と [レコメン](#) デーションを成功させるための例を示して、step-by-step ガイダンスを提供します。
- Amazon Q Developer を使用する際の一般的な課題を軽減し、開発者の明確さを促進する
 - 開発者の期待を満たし、コード生成の精度とパフォーマンスに関連するハードルを克服するための [戦略](#) とインサイトを提供します。
- トラブルシューティングとエラー処理を提供する
 - 不正確な結果や予期しない動作に対処するための Amazon Q Developer コード生成 [トラブルシューティングガイド](#) をデベロッパーに提供します。
 - Python とに固有の実際の [例とシナリオ](#) を提供します Java。
- ワークフローと生産性を最適化する
 - Amazon Q Developer を使用してコード開発ワークフローを最適化します。
 - [開発者の生産性](#) を向上させる戦略について説明します。

デベロッパーワークフローでの Amazon Q デベロッパーの使用

デベロッパーは、要件収集、[設計と計画](#)、[コーディング](#)、テスト、[コードレビュー](#)、[デプロイ](#)の各段階を含む標準ワークフローに従います。このセクションでは、Amazon Q デベロッパー機能を使用して主要な開発ステップを最適化する方法に焦点を当てます。



前の図は、Amazon Q Developer がコード開発の段階で次の一般的なタスクを高速化および合理化する方法を示しています。

- 設計と計画 | 環境設定 | コードの整理
 - 関連するライブラリを生成する
 - クラスと関数の概要を生成する
 - Amazon Q に適切に設計されたアドバイスを求める
 - Amazon Q を使用してコードをリファクタリングする
- コード書き込み | デバッグとプロファイリング | ユニットテスト | ドキュメント
 - 一般的なアルゴリズムを生成する
 - インラインコードレコメンデーションを受信する

- Amazon Q にコードの最適化と修正を依頼する
- デバッグステートメントとプロファイリングステートメントを生成する
- ユニットテストの生成
- スクリプト内でドキュメントとコメントを生成する
- コードレビュー
 - Amazon Q にコードの説明を求める
 - Amazon Q に質問を含むコードをプロンプトとして送信する

設計と計画

ビジネス要件と技術要件を収集した後、デベロッパーは新しいコードベースを設計するか、既存のコードベースを拡張します。このフェーズでは、Amazon Q デベロッパーはデベロッパーが以下のタスクを実行するのを支援できます。

- 適切な設計のアドバイスのために、関連するライブラリとクラスと関数の概要を生成します。
- エンジニアリング、互換性、アーキテクチャ設計クエリに関するガイダンスを提供します。

コーディング

コーディングプロセスは Amazon Q Developer を使用して、以下の方法で開発を加速します。

- 環境設定 - 統合開発環境 (IDE) AWS Toolkit に をインストールします (VS Code や IntelliJ など)。次に、Amazon Q を使用してライブラリを生成したり、プロジェクト目標に基づいてセットアップの提案を受け取ったりします。詳細については、[「Amazon Q Developer のオンボーディングのベストプラクティス」](#)を参照してください。
- コード組織 - プロジェクトの目的に沿ったコードをリファクタリングするか、Amazon Q から組織のレコメンデーションを取得します。
- コード書き込み - 開発中にインラインの提案を使用してコードを生成するか、 の Amazon Q チャットパネルを使用してコードを生成するように Amazon Q に依頼しますIDE。詳細については、[「Amazon Q Developer を使用したコード生成のベストプラクティス」](#)を参照してください。
- デバッグとプロファイリング - プロファイリングコマンドを生成するか、Fix や Explain などの Amazon Q オプションを使用して問題をデバッグします。

- ユニットテスト - チャットセッション中に Amazon Q にプロンプトとしてコードを提供し、該当するユニットテスト生成をリクエストします。詳細については、[「Amazon Q Developer を使用したコード例」](#)を参照してください。
- ドキュメント - インライン提案を使用してコメントとドキュメントを作成するか、Explain オプションを使用してコード選択の詳細な概要を生成します。詳細については、[「Amazon Q Developer を使用したコード例」](#)を参照してください。

コードレビュー

レビュアーは、開発コードを本番環境に昇格させる前に開発コードを理解する必要があります。このプロセスを高速化するには、Amazon Q Explain and Optimize オプションを使用するか、チャットセッションでカスタムプロンプトの指示とともにコード選択を Amazon Q に送信します。詳細については、[「チャットの例」](#)を参照してください。

統合とデプロイ

プロジェクトのアーキテクチャに固有の継続的統合、配信パイプライン、デプロイのベストプラクティスに関するガイドンスについては、Amazon Q にお問い合わせください。

これらの推奨事項を使用すると、Amazon Q デベロッパーの機能を効果的に活用し、ワークフローを最適化し、開発ライフサイクル全体で生産性を向上させる方法を学ぶことができます。

Amazon Q Developer の高度な機能

このガイドでは、実践的なプログラミングタスクでの Amazon Q Developer の使用に焦点を当てていますが、以下の高度な機能に注意することが重要です。

- Amazon Q Developer コード変換
- Amazon Q Developer のカスタマイズ

Amazon Q Developer コード変換

コード変換用の Amazon Q Developer Agent は、コードを手動で書き直すことなく、ファイルのコード言語バージョンをアップグレードできます。これは、既存のコードファイルを分析して、新しいバージョンの言語を使用するように自動的に書き換えることによって機能します。例えば、のような IDE で作業している場合、Amazon Q は単一のモジュールを変換し、Eclipse、Visual Studio Code を使用している場合、Amazon Q はプロジェクトまたはワークスペース全体を変換できます。

次のような一般的なコードアップグレードタスクを実行する場合は、Amazon Q を使用します。

- 言語バージョンの新しい構文で動作するようにコードを更新します。
- ユニットテストを実行して、コンパイルと実行の成功を検証します。
- デプロイの問題を確認して解決します。

Amazon Q は、開発者がコードベースをアップグレードするための面倒で反復的な作業を数日から数か月節約できます。

2024 年 6 月現在、Amazon Q Developer は Java コードのアップグレードをサポートし、8 Java 7 のコードを 11 や Java 17 などの新しいバージョンに変換できます。

Amazon Q Developer のカスタマイズ

カスタマイズ機能により、Amazon Q Developer は会社独自のコードベースに基づいてインライン提案を提供できます。同社は、コードリポジトリを Amazon Simple Storage Service (Amazon S3) または以前 AWS CodeConnections に提供します。AWS CodeStar 次、Amazon Q はセキュリティが有効になっているカスタムコードリポジトリを使用して、その組織の開発者に関連するコーディングパターンを推奨します。

Amazon Q Developer のカスタマイズを使用する場合は、次の点に注意してください。

- 2024 年 6 月現在、Amazon Q Developer Customizations 機能はプレビューモードです。その結果、この機能の可用性とサポートが制限される可能性があります。
- カスタムインラインコード提案は、提供されているコードリポジトリの品質を考慮してのみ正確です。作成するカスタマイズごとに[評価スコア](#)を確認することをお勧めします。
- パフォーマンスを最適化するには、すべてのソースファイルが 10MB 未満であることをお勧めします。リポジトリが参照可能なソースコードで構成され、メタデータファイル (設定ファイル、プロパティファイル、readme ファイルなど) で構成されていることを確認します。

Amazon Q Developer のカスタマイズを使用すると、次の方法で時間を節約できます。

- 独自の独自のコードに基づくレコメンデーションを使用します。
- 既存のコードベースの再利用性を高めます。
- 組織全体に一般化される反復可能なパターンを作成します。

Amazon Q Developer のベストプラクティス

このセクションでは、Amazon Q Developer によるコーディングの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には、次のカテゴリが含まれます。

- [オンボーディング](#) - オンボーディング時の方法と考慮事項
- [コード生成](#) - コード生成を正常に使用するガイド
- [コードレコメンデーション](#) - コードを改善するためのテクニック

Amazon Q Developer をオンボーディングするためのベストプラクティス

Amazon Q Developer は、Visual Studio Code や JetBrains などの一般的な IDEs を通じて利用できる強力な生成 AI コーディングアシスタントです。JetBrains このセクションでは、Amazon Q Developer にアクセスしてコーディング開発環境にオンボーディングするためのベストプラクティスに焦点を当てます。

Amazon Q Developer の前提条件

Amazon Q Developer は、AWS Toolkit for Visual Studio Code およびの一部として使用できます AWS Toolkit for JetBrains (IntelliJ や PyCharm など)。Visual Studio Code および JetBrains IDEs、Amazon Q Developer は Python、Java、JavaScript、TypeScript、C#、Go、Rust、PHP、Ruby、Kotlin、C、C++、Shell スクリプト、SQL、Scala をサポートしています。

Visual Studio Code と JetBrains IDE の両方 AWS Toolkit にインストールする詳細な手順については、[「Amazon Q Developer User Guide」の「Installing the Amazon Q Developer extension or plugin in your IDE」](#)を参照してください。

Amazon Q Developer を使用する際のベストプラクティス

Amazon Q Developer を使用する際の一般的なベストプラクティスは次のとおりです。

- 関連するコンテキストを提供して、使用中のプログラミング言語、フレームワーク、ツールなど、より正確なレスポンスを取得します。複雑な問題を小さなコンポーネントに分割します。
- プロンプトと質問を試して繰り返します。多くの場合、プログラミングにはさまざまなアプローチを試す必要があります。

- コード提案を受け入れる前に必ず確認し、必要に応じて編集して、意図したとおりに実行されていることを確認します。
- [カスタマイズ機能](#)を使用して、Amazon Q Developer が内部ライブラリ、APIs、ベストプラクティス、アーキテクチャパターンを認識し、より関連性の高いレコメンデーションを実現できるようにします。

Amazon Q Developer のデータプライバシーとコンテンツの使用状況

Amazon Q Developer の使用を決定するときは、データとコンテンツの使用方法を理解する必要があります。以下は、重要なポイントです。

- Amazon Q Developer Pro ユーザーの場合、コードコンテンツはサービスの改善やモデルトレーニングには使用されません。
- Amazon Q Developer 無料利用枠のユーザーの場合、IDE 設定または AWS Organizations ポリシーを使用して、サービスの改善にコンテンツを使用することをオプトアウトできます。
- 送信されたコンテンツは暗号化され、保存されたコンテンツは保管時の暗号化とアクセスコントロールで保護されます。詳細については、[「Amazon Q Developer ユーザーガイド」の「Amazon Q Developer のデータ暗号化」](#)を参照してください。

Amazon Q Developer を使用したコード生成のベストプラクティス

Amazon Q Developer は、自動コード生成、自動補完、自然言語コードの提案を提供します。以下は、Amazon Q Developer インラインコーディング支援を使用するためのベストプラクティスです。

- レスポンスの精度向上に役立つコンテキストを提供する

既存のコードから開始する、ライブラリをインポートする、クラスと関数を作成する、またはコードスケルトンを確立する。このコンテキストは、コード生成の品質を大幅に向上させるのに役立ちます。

- 自然にコードする

堅牢な自動補完エンジンのように Amazon Q Developer コード生成を使用します。通常どおりコーディングし、入力または一時停止するときに Amazon Q に提案してもらいます。コード生成が利用できない場合、またはコードの問題で停止している場合は、PC で Alt+C、MacOS で Option+C と入力して Amazon Q を開始します。インライン提案の使用中に実行できる一般的なア

クシヨンの詳細については、「Amazon Q Developer [ユーザーガイド](#)」の「[ショートカットキーの使用](#)」を参照してください。

- スクリプトの目的に関連するインポートライブラリを含める

Amazon Q がコンテキストを理解し、それに応じてコードを生成するのに役立つ、関連するインポートライブラリを含めます。関連するインポートステートメントを提案するように Amazon Q に依頼することもできます。

- 明確で焦点を絞ったコンテキストを維持する

スクリプトを特定の目標に集中させ、個別の機能に関連するコンテキストを持つ個別のスクリプトにモジュール化します。ノイズの多いコンテキストや紛らわしいコンテキストは避けてください。

- プロンプトを試す

さまざまなプロンプトを調べて Amazon Q をナッジし、コード生成に役立つ結果を生成します。たとえば、次のアプローチを試してください。

- 自然言語プロンプトには標準のコメントブロックを使用します。
 - クラスと関数を入力するコメントを含むスケルトンを作成します。
 - プロンプトでは、一般化ではなく詳細を具体的に指定します。
- Amazon Q Developer とチャットしてサポートを依頼する

Amazon Q Developer が正確な提案をしていない場合は、IDE で Amazon Q Developer とチャットします。コードスニペットまたはフルクラスと関数を提供して、コンテキストを開始できます。詳細については、「[Amazon Q Developer ユーザーガイド](#)」の「[コードに関する Amazon Q Developer とのチャット](#)」を参照してください。

Amazon Q Developer を使用したコードレコメンデーションのベストプラクティス

Amazon Q Developer は、開発者からの質問を受け、コードを評価して、コード生成やバグ修正から自然言語を使用したガイダンスまで、さまざまなレコメンデーションを提供できます。Amazon Q でチャットを使用するためのベストプラクティスを以下に示します。

- ゼロからコードを生成する

新しいプロジェクトや一般的な関数 (Amazon S3 からのファイルのコピーなど) が必要な場合は、自然言語プロンプトを使用してコード例を生成するように Amazon Q Developer に依頼してくだ

さい。Amazon Q は、さらなる検証と調査のために、パブリックリソースへの関連リンクを提供できます。

- コーディングの知識とエラーの説明を求める

コーディングの問題やエラーメッセージに直面した場合は、Amazon Q Developer へのプロンプトとしてコードブロック (該当する場合はエラーメッセージを含む) と質問を指定します。このコンテキストは、Amazon Q が正確で関連するレスポンスを提供するのに役立ちます。

- 既存のコードの改善

既知のエラーを修正したり、コードを最適化したりするには (複雑さを減らすためなど)、関連するコードブロックを選択し、リクエストとともに Amazon Q Developer に送信します。より良い結果を得るには、プロンプトに特に注意してください。

- コード機能の説明

新しいコードリポジトリを調べるときは、コードブロックまたはスクリプト全体を選択し、Amazon Q Developer に送信して説明を求めます。より具体的な説明については、選択サイズを小さくします。

- ユニットテストの生成

コードブロックをプロンプトとして送信したら、Amazon Q Developer にユニットテストの生成を依頼します。このアプローチにより、コードカバレッジと DevOps に関連する時間と開発コストを削減できます。

- AWS 回答の検索

Amazon Q Developer は、に関連する膨大な知識が含まれているため、を使用する AWS のサービス 開発者にとって貴重なリソースです AWS。特定の で課題に直面しているか AWS のサービス、特定のエラーメッセージに遭遇しているか AWS、新しい を学習しようとしているかにかかわらず AWS のサービス、Amazon Q は多くの場合、関連性の高い有用な情報を提供します。

Amazon Q Developer が提供する推奨事項を必ず確認してください。次に、必要な編集を行い、テストを実行して、コードが意図した機能を満たしていることを確認します。

Amazon Q Developer のコード例

このセクションでは、Pythonおよび Java 言語に焦点を当ててエクスペリエンスとコード生成を改善する方法の現実的な例を示します。インラインの例に加えて、Amazon Q Developer のチャット機能を使用して、プログラミング中にコードのレコメンデーションを求める方法を示すシナリオもあります。

詳細については、以下の各トピックを参照してください。

- [Python 例](#)
- [Java 例](#)
- [チャットの例](#)

Python の例

次のPython例は、Amazon Q Developer でインラインコード提案の生成を開始する方法を示しています。これらの例に示しているのは、独自のプロジェクトのコーディングアシスタントをトリガーする方法です。以下の例の大部分は、他の言語に適用されます。

- [クラスと関数を生成する](#)
- [ドキュメントコード](#)
- [アルゴリズムの生成](#)
- [ユニットテストを生成する](#)

クラスと関数を生成する

クラス全体または部分的な実装を生成するには、コードコメントを使用します。コメントは、クラスの意図を記述します。シンプルに開始し、必要に応じて詳細を追加します。クラスを生成した後、スケルトン内で関数を引き続き生成できます。

コメントを作成するときは、言語のコメント構造を使用することをお勧めします。ではPython、1行のコメントには # を使用します。複数行のコメントの場合は、`''' '''` または `""" """` を使用します。

入力

```
...
DocumentReader class. has a name, children and optional parent.
Includes I/O functions to read and
write document
...
```

出力:

```
class DocumentReader:
    def __init__(self, name, children, parent=None):
        self.name = name
        self.parent = parent
        self.children = children

    def read(self):
        """Reads current document and returns text"""
        pass

    def write(self, text):
        """Writes text to current document"""
        pass
```

初期クラスができたので、追加のコメントを使用して追加の関数を生成できます。

入力

```
...
function to read document from local directory path and print the first line of the
document
...
```

出力:

```
def read_document(self, path):
    """Reads document from path and prints first line"""
    import os
    if os.path.isfile(path):
        with open(path, 'r') as f:
            print(f.readline())
    else:
        print(f"File not found at {path}")
```

ドキュメントコード

インラインドキュメントに依存することで、コードドキュメントの時間を短縮します。Python コメントブレイクを追加します''' '''。

入力

```
class S3Utils:
    """
    <place cursor here>
    """
    def __init__(self):
        self.base_url = "https://s3"

    def generate_s3_url(self, bucket_name, region):
        return f"{self.base_url}.{region}.amazonaws.com/{bucket_name}"

# Example usage:
if __name__ == "__main__":
    s3_util = S3Utils()
    bucket_name = "my-example-bucket"
    region = "us-west-1"
    s3_url = s3_util.generate_s3_url(bucket_name, region)
    print(f"S3 URL for bucket '{bucket_name}' in region '{region}': {s3_url}")
```

出力:

```
class S3Utils:
    """
    Class to generate S3 URL from bucket name and region
    """
    def __init__(self):
        self.base_url = "https://s3"

    def generate_s3_url(self, bucket_name, region):
        return f"{self.base_url}.{region}.amazonaws.com/{bucket_name}"

# Example usage:
if __name__ == "__main__":
    s3_util = S3Utils()
    bucket_name = "my-example-bucket"
    region = "us-west-1"
    s3_url = s3_util.generate_s3_url(bucket_name, region)
```

```
print(f"S3 URL for bucket '{bucket_name}' in region '{region}': {s3_url}")
```

アルゴリズムの生成

Amazon Q Developer は、特に AWS SDKs に関連する一般的なアルゴリズムを生成できます。スクリプトで、必要な関数を指定するプロンプトを追加します。

この例では、値に名前を付けるユーザー ID キーのディクショナリを生成し、バブルソートアルゴリズムを使用してソート関数を生成する方法を示します。

入力

```
class SortList:
    @staticmethod
    def get_user_id_name_map():
        user_id_name_map = {
            1: "Amit",
            2: "Rahul",
            3: "Vijay",
            4: "Raj",
            5: "Kumar"
        }
        return user_id_name_map

...
Sort the list of user id's alphabetically using bubble sort
...
```

出力:

```
@staticmethod
def sort_user_id_name_map():
    user_id_name_map = SortList.get_user_id_name_map()
    list_of_user_ids = list(user_id_name_map.keys())
    list_of_user_names = list(user_id_name_map.values())
    print("Before sort: ", list_of_user_ids)
    for i in range(len(list_of_user_ids)):
        for j in range(len(list_of_user_ids) - i - 1):
            if list_of_user_names[j] > list_of_user_names[j + 1]:
                list_of_user_ids[j], list_of_user_ids[j + 1] = list_of_user_ids[j + 1], list_of_user_ids[j]
```

```
        list_of_user_names[j], list_of_user_names[j + 1] =
list_of_user_names[j + 1], list_of_user_names[j]
    print("After sort: ", list_of_user_ids)
```

さらに、標準手順のプロンプトを作成できます。この例では、ユーザーはディクショナリを Amazon DynamoDB テーブルに保存するためのプロンプトを作成します。スクリプトの先頭に、ユーザーは Amazon Q Developer に追加コンテキストを提供するのに役立つ DynamoDB ライブラリを追加するプロンプトを追加します。

入力

```
...
Add libraries for Amazon Dynamo DB
...
```

出力:

```
...
Add libraries for Amazon Dynamo DB and JSON.

import necessary libraries for working with Amazon DynamoDB and JSON data.
...

import boto3
import json
from botocore.exceptions import ClientError

def load_data_to_dynamodb(table_name, data_file):
    # Create a DynamoDB resource
    dynamodb = boto3.resource('dynamodb')

    # Get a reference to the table
    table = dynamodb.Table(table_name)

    # Load data from the JSON file
    with open(data_file, 'r') as file:
        data = json.load(file)

    # Iterate over the data and load it to the table
    for item in data:
        try:
            table.put_item(Item=item)
```

```
        print(f"Successfully loaded data: {item}")
    except ClientError as e:
        print(f"Error loading data: {e.response['Error']['Message']}")

# Example usage
load_data_to_dynamodb('myTable', 'data.json')
```

これで、ユーザーは DynamoDB にディクショナリを保存するのに役立つように Amazon Q に簡単に依頼できるようになりました。

入力

```
'''
    function to store the above sorted dict to dynamodb table named users
'''
```

出力:

```
@staticmethod
def store_user_id_name_map_in_dynamodb():
    user_id_name_map = SortList.get_user_id_name_map()
    list_of_user_ids = list(user_id_name_map.keys())
    list_of_user_names = list(user_id_name_map.values())
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('users')
    for i in range(len(list_of_user_ids)):
        try:
            response = table.put_item(
                Item={
                    'user_id': list_of_user_ids[i],
                    'user_name': list_of_user_names[i]
                }
            )
        except ClientError as e:
            print(e.response['Error']['Message'])
```

ユニットテストを生成する

インラインコメントを使用すると、Amazon Q Developer はアサーションやその他のテストを実行するためのユニットテストを生成できます。この例では、ユーザーは Amazon Q にユニットテスト用の適切なライブラリを生成して、テストスクリプトのコンテキストを設定するよう求めます。次に、

ユーザーは Amazon Q に `sort_user_id_name_map` 関数のテスト関数を記述するように要求します。

入力

```
...
Import all the required unit test libraries for python and then generate a unit test
for sort_user_id_name_map function.
...
```

出力:

```
import unittest
from unittest.mock import patch
from sort_user_id_name_map import sort_user_id_name_map

class TestSortUserNameMap(unittest.TestCase):
    @patch('sort_user_id_name_map.sort_user_id_name_map')
    def test_sort_user_id_name_map(self, mock_sort):
        mock_sort.return_value = {'id1': 'name1', 'id2': 'name2'}
        result = sort_user_id_name_map({'id2': 'name2', 'id1': 'name1'})
        self.assertEqual(result, {'id1': 'name1', 'id2': 'name2'})

if __name__ == '__main__':
    unittest.main()
```

さらに、ユーザーと Amazon Q Developer の間のチャットでは、ユーザーのコード入力プロンプトに基づいてユニットテストを提案して生成します。詳細については、[「チャットの例」](#)を参照してください。

Java の例

次の Java 例は、ユーザーがインラインコード提案の生成を開始する方法を示しています。これらの例に示しているのは、独自のプロジェクトのコーディングアシスタントをトリガーする方法です。以下の例の大部分は、他の言語に適用されます。

- [クラスと関数を生成する](#)
- [ドキュメントコード](#)
- [アルゴリズムの生成](#)
- [ユニットテストを生成する](#)

クラスと関数を生成する

クラス全体または部分的な実装を生成するには、コードコメントを使用します。コメントは、クラスの意図を記述します。シンプルに開始し、必要に応じて詳細を追加します。クラスを生成した後も、スケルトン内で関数を生成し続けることができます。

コメントを書き込むときは、言語のコメント構造を使用することをお勧めします。ではJava、これはです/** */。

入力

```
/**
 * DocumentReader class. has a name, children and optional parent.
 * Includes I/O functions to read and
 * write document
 */
```

出力:

```
/**
 * DocumentReader class. has a name, children and optional parent.
 * Includes I/O functions to read and
 * write document
 */
import java.util.ArrayList;
public class DocumentReader {
    private String name;
    private DocumentReader parent;
    private ArrayList<DocumentReader> children;

    /**
     * Constructor
     * @param name name of the document
     * @param parent parent of the document
     */
    public DocumentReader(String name, DocumentReader parent) {
        this.name = name;
        this.parent = parent;
        this.children = new ArrayList<>();
    }

    /**
     * Constructor
```

```
    * @param name name of the document
    */
    public DocumentReader(String name) {
        this(name, null);
    }
    /**
     * Getter for name
     * @return name of the document
     */
    public String getName() {
        return name;
    }

    /**
     * Getter for parent
     * @return parent of the document
     */
    public DocumentReader getParent() {
        return parent;
    }

    /**
     * Getter for children
     * @return children of the document
     */
    public ArrayList<DocumentReader> getChildren() {
        return children;
    }

    /**
     * Adds a child to the document
     * @param child child to be added
     */
    public void addChild(DocumentReader child) {
        children.add(child);
    }

    /**
     * Removes a child from the document
     * @param child child to be removed
     */
    public void removeChild(DocumentReader child) {
        children.remove(child);
    }
}
```

```
}
```

初期クラスができたので、追加のコメントを使用して追加の関数を生成できます。

入力

```
/**  
 * function to read document from local directory path  
 * and print first line of document  
 */
```

出力:

```
/**  
 * function to read document from local directory path  
 * and print first line of document  
 */  
public void readDocument() {  
    try {  
        BufferedReader reader = new BufferedReader(new FileReader(name));  
        String line = reader.readLine();  
        System.out.println(line);  
        reader.close();  
    } catch (IOException e) {  
        System.out.println("Error reading document: " + e.getMessage());  
    }  
}
```

ドキュメントコード

インラインドキュメントに依存することで、コードドキュメントの時間を短縮します。Java コメントブレイクを追加します/* */。

入力

```
import java.util.function.Function;  
  
public class S3Utils {  
    /*  
     * <place cursor here>  
     */  
}
```

```
public static Function<String, String> generateS3Function(String bucketName, String region) {
    return (String key) -> {
        String url = generateS3Url(bucketName, key, region);
        return url;
    };
}
```

インライン生成が発生しない場合は、ドキュメントを作成する場所にカーソルを置いた後に Amazon Q Developer のインライン提案を実行します。

出力:

```
import java.util.function.Function;

public class S3Utils {
    /*
     * Generate a function that returns the S3 URL for a given key
     * @param bucketName the name of the bucket
     * @param region the region of the bucket
     * @return the function that generates the S3 URL
     */
    public static Function<String, String> generateS3Function(String bucketName, String region) {
        return (String key) -> {
            String url = generateS3Url(bucketName, key, region);
            return url;
        };
    }
}
```

コード (クラス、関数、またはスクリプト要素) の簡単な説明には、ドキュメントを追加するコードセクションの前に戦略的に配置されたフォワード文字 (//) を使用します。

入力

```
//
public class S3Utils {
    /*
     * Generate a function that returns the S3 URL for a given key
     * @param bucketName the name of the bucket
```

```
* @param region the region of the bucket
* @return the function that generates the S3 URL
*/
public static Function<String, String> generateS3Function(String bucketName, String
region) {
    return (String key) -> {
        String url = generateS3Url(bucketName, key, region);
        return url;
    };
}
}
```

出力:

```
// This class is used to generate a function that returns the S3 URL for a given key
public class S3Utils {
    /*
    * Generate a function that returns the S3 URL for a given key
    * @param bucketName the name of the bucket
    * @param region the region of the bucket
    * @return the function that generates the S3 URL
    */
    public static Function<String, String> generateS3Function(String bucketName, String
region) {
        return (String key) -> {
            String url = generateS3Url(bucketName, key, region);
            return url;
        };
    }
}
```

アルゴリズムの生成

Amazon Q Developer は、特に AWS SDKs に関連する一般的なアルゴリズムを生成できます。スクリプトで、必要な関数を指定するプロンプトを追加します。

この例では、名前値へのユーザー ID キーのハッシュマップを生成し、バブルソートアルゴリズムを使用してソート関数を生成します。

入力

```
import java.util.HashMap;
```

```
public class SortList {

    // generate key value pair hash map for
    // user id and name
    public static HashMap<Integer, String> getUserIdNameMap() {
        HashMap<Integer, String> userIdNameMap = new HashMap<Integer, String>();
        userIdNameMap.put(1, "Amit");
        userIdNameMap.put(2, "Rahul");
        userIdNameMap.put(3, "Vijay");
        userIdNameMap.put(4, "Raj");
        userIdNameMap.put(5, "Kumar");
        return userIdNameMap;
    }

    /*
    * Create function to sort UserIdName map by name alphabetically using bubble sort
    */

}
```

出力:

```
/*
* Create function to sort UserIdName map by name alphabetically using bubble sort
*/
public static HashMap<Integer, String> sortUserIdNameMap(HashMap<Integer, String>
userIdNameMap) {
    int size = userIdNameMap.size();
    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size; j++) {
            if (userIdNameMap.get(i).compareTo(userIdNameMap.get(j)) > 0) {
                String temp = userIdNameMap.get(i);
                userIdNameMap.put(i, userIdNameMap.get(j));
                userIdNameMap.put(j, temp);
            }
        }
    }
    return userIdNameMap;
}
```

さらに、ユーザーは標準手順のプロンプトを作成できます。この例では、ユーザーはハッシュマップを Amazon DynamoDB テーブルに保存するためのプロンプトを作成します。スクリプトの先頭に、

ユーザーは Amazon Q Developer に追加コンテキストを提供するのに役立つ DynamoDB Java ライブラリを追加するプロンプトを追加します。

入力

```
// add libraries for Amazon DynamoDB
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;

/*
 * Store sorted UserIdName map into Amazon DynamoDB table named users
 */
```

出力:

```
// add libraries for Amazon DynamoDB
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;

/*
 * Store sorted UserIdName map into Amazon DynamoDB table named users
 */
public static void storeSortedUserIdNameMap(HashMap<Integer, String> userIdNameMap)
{
    // create Amazon DynamoDB client
    AmazonDynamoDB client = AmazonDynamoDBClientBuilder.defaultClient();

    // store sorted UserIdName map into Amazon DynamoDB table named users
    for (int userId : userIdNameMap.keySet()) {
        String name = userIdNameMap.get(userId);
        client.putItem("users", new HashMap<String, Object>() {{
            put("userId", userId);
            put("name", name);
        }});
    }
}
```

ユニットテストを生成する

インラインコメントを使用すると、Amazon Q Developer はアサーションやその他の JUnit テストを実行するためのユニットテストを生成できます。この例では、ユーザーはまず Amazon Q に JUnit が

テストスクリプトのコンテキストを設定するための適切なライブラリを生成するように依頼します。次に、ユーザーはパブリッククラスを作成し、その後に宣言子とコメント文字列を作成します。

入力

```
// import libraries for java unit tests
import org.junit.Test;
import static org.junit.Assert.*;

public class UnitTests {

    @Test
    /*
     * Create unit test for getUserIdNameMap
     */
}
```

出力:

```
@Test
/*
 * Create unit test for getUserIdNameMap
 */
public void testGetUserIdNameMap() {
    // create a new instance of the UserIdNameMap class
    UserIdNameMap userIdNameMap = new UserIdNameMap();

    // create a new instance of the User class
    User user = new User("John", "Doe");

    // add the user to the map
    userIdNameMap.addUser(user);

    // get the user from the map
    User retrievedUser = userIdNameMap.getUser("John");

    // assert that the retrieved user is the same as the user added to the map
    assertEquals(user, retrievedUser);
}
```

さらに、ユーザーと Amazon Q Developer の間のチャットでは、ユーザーのコード入力プロンプトに基づいてユニットテストを提案して生成します。詳細については、「[チャットの例](#)」を参照してください。

チャットの例

Amazon Q Developer チャットは、アプリケーションのコーディング、テスト、アップグレード、エラーの診断、セキュリティスキャンと修正の実行、AWS リソースの最適化など、開発者のさまざまなタスクを支援します。このセクションでは、Amazon Q とのチャットが、次のような一般的なプログラミングタスクで開発者をサポートする方法について説明します。

- [について質問する AWS のサービス](#)
- [コードの生成](#)
- [ユニットテストを生成する](#)
- [コードの説明](#)

について質問する AWS のサービス

ユーザーは、適切なサービスの検索やベストプラクティスの理解など AWS のサービス、について Amazon Q Developer に質問できます。

に関するチャット AWS のサービス

ユーザー：

コストを低く抑える必要がある場合、ワークロードの実行に使用するのに最適なコンテナサービスは何ですか？

Amazon Q:

提供された情報に基づいて、ワークロードを実行するための費用対効果の高いコンテナサービスを探しているようです。その場合は、Fargate で Amazon Elastic Container Service (Amazon ECS) を検討することをお勧めします。[1]

Amazon ECS with Fargate は、基盤となるインフラストラクチャを管理することなくコンテナを実行できるようにする、コンテナ用のサーバーレスコンピューティングエンジンです。Fargate では、サーバーをプロビジョニングまたは管理する必要はありません。Fargate がそれを処理します。これにより、アイドル状態のサーバー容量を支払うことなく、コンテナが実際に使用するコンピューティングリソースに対してのみ支払うため、コストを抑えることができます。[2]

Fargate で Amazon ECS を使用する主な利点は次のとおりです。

- サーバーやクラスターを管理する必要はありません。Fargate がインフラストラクチャのプロビジョニングと管理を代行します。
- Pay-as-you-go – コンテナが使用するコンピューティングリソースに対してのみ料金が発生し、アイドル容量に対して料金を支払う必要はありません。
- 自動スケーリング – Fargate は、需要に応じてコンテナを自動的にスケールアップおよびスケールダウンできます。
- ロードバランシング、ログ記録、モニタリング AWS のサービス など、他のとの緊密な統合。

提供されたリソースには、ECS サービスがコンテナのスケジュールと配置の点で柔軟性を提供し、コストをさらに最適化するのに役立つことも記載されています。

特定のワークロード要件に基づいてコストをより正確に把握するために、ウェブサイトでは AWS Fargate を使用した Amazon ECS の料金の詳細を確認することをお勧めします。もちろん、他に質問がございましたら、お気軽にお問い合わせください。

[Sources] (出典)

[1] どの AWS コンテナサービスが最適ですか？

aptgetops.tech > [aws-container-services-which-one-is-right-for-you](#)

[2] HTTP 以外のワーロードのJavaコンテナオプション

> 質問 > [43417016](#)

に関する Amazon Q チャットを終了する AWS のサービス

コードの生成

ユーザーは Amazon Q Developer にコードスニペットの記述に関するサポートを依頼できます。

コードの記述に関するチャット

ユーザー：

を使用してすべての S3 バケットを一覧表示する方法の例を挙げてください AWS SDK for Python (Boto)。

Amazon Q:

確かに、AWS SDK for Python (Boto)[1] を使用してすべての S3 バケットを一覧表示する方法の例を次に示します。

```
import boto3
# Create an S3 client
s3 = boto3.client('s3')

# List all the buckets
response = s3.list_buckets()

# Print the name of each bucket
for bucket in response['Buckets']:
    print(bucket['Name'])
```

コードの仕組みは次のとおりです。

1. 最初に boto3 ライブラリをインポートします。これは SDK for Python です。
2. を使用して S3 クライアントを作成します boto3.client('s3')。
3. S3 クライアントで list_buckets() メソッドを呼び出し、アカウント内のすべてのバケットのリストを含むレスポンスを返します。
4. 次に、レスポンス Buckets のリストを繰り返して、各バケットの名前を出力します。

これは基本的な例ですが、AWS SDK for Python (Boto) には、バケット内のオブジェクトの作成、削除、管理、バケットポリシーの設定など、Amazon S3 を操作するためのより高度な機能が多数用意されています。その他の例とドキュメントについては、AWS SDK for Python (Boto) ドキュメントを参照してください。[2]

を使用するときは AWS のサービス、コードにハードコーディングするのではなく、環境変数にアクセスキーを保存する AWS など、認証情報や機密情報を処理するためのベストプラクティスに従うことが重要です。

ソース: [1] [SDK for Python \(Boto3\) を使用した Amazon S3 の例 - AWS SDK コードの例 docs.aws.amazon.com > code-library > latest-1x > ... > python-s3.html](https://docs.aws.amazon.com/code-library/latest-1x/python-s3.html)

[2] [AWS SDK - AWS SDK コード例 docs.aws.amazon.com > code-library > latest-1x > ... > s3_example_s3_buckets_GettingStarted_sect](https://docs.aws.amazon.com/code-library/latest-1x/s3_example_s3_buckets_GettingStarted_sect) を使用して Amazon S3 バケットとオブジェクトの使用を開始します。

コードの記述に関する Amazon Q チャットを終了する

ユニットテストを生成する

ユーザーは、選択したコードをプロンプトとして Amazon Q Developer に送信できます。ここでは、ユニットテストの生成などの追加の手順が適用されます。詳細な手順についてはstep-by-step [「Amazon Q Developer User Guide」](#) の [「Send code to Amazon Q」](#) を参照してください。

ユニットテストに関するチャット

ユーザー :

次の関数の模擬データを使用してユニットテストを記述します。

```
// poll messages from Amazon SQS queue name
public List<String> pollMessages(String queueName) {
    List<String> messages = new ArrayList<>();
    // poll messages from queue
    ReceiveMessageResult = sqsClient.receiveMessage(new ReceiveMessageRequest().
    for (Message : receiveMessageResult.getMessages()) {
        messages.add(message.getBody());
    }
    return messages;
}
```

Amazon Q:

pollMessages 関数のモックデータを使用してユニットテストを記述するには、Mockito などのモックフレームワークを使用できます。例を示します。

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
import com.amazonaws.services.sqs.model.ReceiveMessageResult;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;

import java.util.Arrays;
import java.util.List;

import static org.junit.jupiter.api.Assertions.assertEquals;
```

```
@ExtendWith(MockitoExtension.class)
public class SQSUtilsTest {

    @Mock
    private AmazonSQS sqsClient;

    @InjectMocks
    private SQSUtils sqsUtils;

    @Test
    public void testPollMessages() {
        // Arrange
        String queueName = "test-queue";
        Message message1 = new Message().withBody("message1");
        Message message2 = new Message().withBody("message2");
        ReceiveMessageResult receiveMessageResult = new
ReceiveMessageResult().withMessages(message1, message2);
        Mockito.when(sqsClient.receiveMessage(new
ReceiveMessageRequest().withQueueUrl(queueName)))
                .thenReturn(receiveMessageResult);

        // Act
        List<String> messages = sqsUtils.pollMessages(queueName);

        // Assert
        assertEquals(Arrays.asList("message1", "message2"), messages);
    }
}
```

この例では、Mockito を使用して AmazonSQS クライアントをモックし、本文「message1」と「message2」を含む 2 つのモックメッセージを返すように receiveMessage メソッドに期待される動作を設定します。次に、pollMessages メソッドを呼び出し、返されたメッセージのリストが想定値と一致することをアサートします。

主なポイントは次のとおりです。

1. @ExtendWith(MockitoExtension.class) 注釈を使用して、Mockito と JUnit の統合を有効にします。
2. @Mock 注釈を使用して、AmazonSQS クライアントのモックインスタンスを作成します。
3. @InjectMocks 注釈を使用して、モック AmazonSQS クライアントを SQSUtils クラスに挿入します。

4. テスト方法では、Mockito.when().thenReturn() を使用して receiveMessage メソッドの予想される動作を設定します。
5. pollMessages メソッドを呼び出し、返されたメッセージのリストが期待値と一致することをアサートします。

このアプローチにより、Amazon SQS サービスと実際にやり取りすることなく pollMessages メソッドをテストできるため、テストを高速化して信頼性を高めることができます。

ユニットテストに関する Amazon Q チャットを終了する

コードの説明

コードレビュープロセスの場合、デベロッパーは Amazon Q Developer にコード選択を送信して、コードをよりよく理解できます。さらに、Amazon Q が提供する説明は、ドキュメントの目的を果たすこともできます。次の例では、Amazon Q が 関数を使用してハンドラー AWS Lambda 関数を説明し、docstring を提案します。

コードの説明に関するチャット

ユーザー :

コードの次の部分について説明します。

```
def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    s3_client = boto3.client('s3')
    table = dynamodb.Table('DYNAMODB_TABLE')
    # Get s3 object from the event
    print(event)
    print(context)
    # get object from event
    bucket_key = event['Records'][0]['s3']['object']['key']
    object_key = unquote_plus(bucket_key)
    fullText = detect_text(object_key, bucket)

    response = s3_client.head_object(Bucket=bucket, Key=object_key)
    print(response)
    print(fullText)
    metadata = response['ResponseMetadata']
    metadata = json.dumps(metadata)

    now = datetime.now()
```

```
current_time = now.strftime("%A %d %B, %Y %H:%M:%S")
print("Current Time =", current_time + "\n\n")
reason, outcome = detect_restaurant_closure(fullText, current_time)
table.put_item(
    Item={
        'id': object_key,
        'creationTime': now.strftime("%Y-%m-%d %H:%M:%S"),
        'reason': reason,
        'text': fullText,
        'metadata': metadata
    }
)
```

Amazon Q:

指定されたコードブロックは、で記述された AWS Lambda 関数のように見えます Python。このコードの主な側面について説明します。

1. 関数の名前は `lambda_handler` で、Lambda 関数のエントリーポイントです。
2. `boto3` ライブラリを使用して AWS DynamoDB および S3 サービスとやり取りします。
3. 関数は、受信イベントデータから S3 オブジェクトキーとバケット名を取得します。
4. 次に、`detect_text` 関数 (提供されたコードには表示されません) を呼び出して、S3 オブジェクトの完全なテキストコンテンツを抽出します。
5. また、この関数は `head_object` メソッドを使用して S3 オブジェクトに関するメタデータを取得します。
6. 次に、現在の時刻を抽出し、`detect_restaurant_closure` 関数 (非表示) を呼び出して、抽出されたテキストに基づいてレストラン閉鎖の理由と結果を決定します。
7. 最後に、関数は次の情報を DynamoDB テーブルに保存します。
 - ID としての S3 オブジェクトキー
 - 作成時刻
 - レストラン閉鎖の理由と結果
 - 抽出された S3 オブジェクトテキスト

全体として、この Lambda 関数は、S3 オブジェクトを処理し、テキストコンテンツを抽出し、レストランの閉鎖情報を分析し、結果を DynamoDB テーブルに保存する大規模なシステムの一部のように見えます。`detect_text` 関数と `detect_restaurant_closure` 関数の具体的な詳細は、このコードの機能を完全に理解するために必要です。

Amazon Q レスポンスの終了

ユーザー: この説明をこの lambda_handler 関数のドキュメント文字列にまとめます。

Amazon Q: lambda_handler 関数の推奨 docstring[1] は次のとおりです。

```
def lambda_handler(event, context):
    """
    AWS Lambda function that processes an S3 object, extracts text content, analyzes
    it, and stores the results.
    Args:
        event (dict): The event data that triggered the Lambda function, containing
        information about the S3 object.
        context (object): The context object provided by AWS Lambda, containing
        information about the runtime environment.

    Returns:
        None
    """

    # function implementation details...
```

この docstring は、関数の目的、予想される入力パラメータ、予想される戻り値 (またはこの場合はそれが無い) の概要を提供します。これにより、実装の詳細を掘り下げることなく、lambda_handler 関数の動作を明確に理解できます。

ソース: [1] Lambda 関数ハンドラーを C# - AWS Lambda <https://docs.aws.amazon.com/lambda/latest/dg/csharp-handler.html> - :~:text=Lambda 関数ハンドラーは、 、終了、またはタイムアウトです。

コードの説明に関する Amazon Q チャットの終了

Amazon Q Developer でのコード生成シナリオのトラブルシューティング

Amazon Q Developer を使用すると、コードの生成と解決が不正確になる一般的なシナリオとして、次のようなものがあります。

- [空のコード生成](#)
- [継続的なコメント](#)
- [インラインコード生成が正しくない](#)
- [チャットの結果が不十分](#)

空のコード生成

コードの開発中に、次の問題が発生することがあります。

- Amazon Q は提案を提供しません。
- IDE に「Amazon Q からの提案なし」というメッセージが表示されます。

まず、Amazon Q が正しく動作していない可能性があります。ただし、これらの問題の根本原因は通常、スクリプト内のコンテキストまたは IDE 内のオープンプロジェクトに関連付けられます。

Amazon Q Developer が自動的に提案を提供しない場合は、次のショートカットを使用して Amazon Q インライン提案を手動で実行できます。

- PC - Alt+C
- MacOS - Option+C

詳細については、「Amazon Q Developer [ユーザーガイド](#)」の「[ショートカットキーの使用](#)」を参照してください。

ほとんどのシナリオでは、Amazon Q が提案を生成します。Amazon Q が「Amazon Q からの提案なし」メッセージを返す場合は、この問題に関する次の一般的な解決策を確認してください。

- 不適切なコンテキスト — 該当するライブラリが存在することを確認します。スクリプトで定義されたクラスと関数が新しいコードに関連していることを確認してください。

- あいまいなプロンプト – プロンプトがわかりにくい、または不明確である場合、Amazon Q はインラインコード提案を行うとパフォーマンスが低下します。これらのシナリオでは、適切なレスポンスを引き出すために、コードとコメントをより具体的に記述してください。

行き詰まった場合は、可能な限りコーディングを続けます。通常、Amazon Q はユーザーの意図を認識し、提案をトリガーします。もう 1 つのオプションは、Amazon Q チャットでプロンプト指示を含むコードスニペットを送信することです。詳細については、[「Amazon Q Developer User Guide」](#)の「[Send code to Amazon Q](#)」を参照してください。

継続的なコメント

コメントを使用してプロンプトを記述し、Amazon Q から汎用または繰り返しのコメント行のみを受け取ることができます。

入力:

```
// Kafka test
```

結果:

```
// Kafka test
// Kafka test
// Kafka test
// Kafka test
```

通常、この問題はスクリプトのコンテキストが不足しているか、プロンプトの指示が不十分なことが原因です。言語を指定した場合、コメントには標準形式を使用することをお勧めします。この例では、ユーザーはコメントブロック `/* */` に使用するプロンプトを変更し、その変更により Amazon Q のパフォーマンスが向上しました。

入力:

```
/*
 * Create a unit test to push a message to a Kafka topic
 */
```

結果:

この結果、ユーザーはいくつかの提案を順番に受け入れました。

```
/*
 * Create a unit test to push a message to a Kafka topic
 */
import org.junit.Test;
import java.util.Properties;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import static org.junit.Assert.assertEquals;
// create function for a Kafka test
public class KafkaTest {
    @Test
    public void testKafka() {
        // create Kafka producer
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");
        KafkaProducer<String, String> producer = new KafkaProducer<>(props);
        // create Kafka topic
        String topic = "test";
        // send message to Kafka topic
        producer.send(new ProducerRecord<>(topic, "key", "value"));
        // close Kafka producer
        producer.close();
        // assert that message was sent successfully
        assertEquals("value", KafkaTest.consumer.poll().value());
    }
}
```

インラインコード生成が正しくない

Amazon Q Developer のインライン提案が役に立たないか、プロンプトの目標を満たしていない場合があります。場合によっては、インラインコード提案は、単純な Hello World メソッドに似た汎用関数を提供することがあります。

インラインコードの提案が役に立たない場合は、次のベストプラクティスを試してください。

- スクリプトのコンテキストを確認します。通常、コンテキストを指定するほど、Amazon Q から得られる結果が向上します。

- コンテキストが異なる場合やノイズが多い場合は、コードをモジュール化します。同じスクリプト内で同様の関数をグループ化します。目的別に異なるクラスと関数を区切ります。
- Amazon Q がより良いレコメンデーションを提供するのに役立つように、クラス名と関数名を戦術的に記述します。

この例では、開発者はファイルを Hadoop にアップロードするスクリプトを最初から作成していません。

入力:

```
class MyFunction{

    // create function to read data locally and upload to HDFS filesystem
```

結果:

```
class MyFunction{

    // create function to read local data file path and upload to HDFS filesystem
    public void readAndUploadToHDFS(String inputPath, String outputPath) throws
    IOException {
        // read data from local file
        BufferedReader in = new BufferedReader(new FileReader(inputPath));
    }
}
```

Amazon Q はレコメンデーションを提供しましたが、開発者の期待を満たしていませんでした。パフォーマンスを向上させるために、開発者はライブラリをインポートし、Hadoop ユーティリティを表すようにクラス名を変更します。

修正:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.FileReader;

// import applicable hadoop libraries for IO operations
import org.apache.hadoop.fs.Path;

class HadoopUtilities{
```

```
// create function to read local data file path and upload to HDFS filesystem
}
```

結果:

この結果、開発者はいくつかの提案を順番に受け入れました。

```
class HadoopUtilities{

    // create function to read local data file path and upload to HDFS filesystem
    public static void uploadToHDFS(String localFilePath, String hdfsFilePath) throws
    IOException {
        // create HDFS file path object
        Path hdfsPath = new Path(hdfsFilePath);
        // create local file path object
        Path localPath = new Path(localFilePath);
        // create hadoop file system object
        org.apache.hadoop.fs.FileSystem hdfs = org.apache.hadoop.fs.FileSystem.get(new
        org.apache.hadoop.conf.Configuration());
        // upload local file to HDFS
        hdfs.copyFromLocalFile(localPath, hdfsPath);
        // close HDFS file system
        hdfs.close();
        System.out.println("File uploaded successfully!");
    }
}
```

同じHadoopUtilitiesスクリプトで、開発者は Amazon Simple Queue Service (Amazon SQS) トピックを読み取るためのコメントプロンプトを追加します。

入力:

```
// read message from SQS
```

結果:

```
// read message from SQS
public static void readMessageFromSQS(String message) {
    System.out.println("Received message from SQS: " + message);
}
```

Amazon Q では、Java メソッドを使用して Amazon SQS から SQS メッセージを読み取る代わりに、文字列メソッドの印刷をお勧めします。ただし、この提案はデベロッパーの元の目標を満たしていません。

この場合、コードを再モジュール化することをお勧めします。スクリプトの焦点は Hadoop メソッドです。したがって、SQS オペレーション専用のユーティリティスクリプトを作成する方が良いかもしれません。

SQS ユーティリティ用に別のクラスを作成すると、HadoopUtilitiesスクリプトで使用できるインライン提案が改善されます。

入力:

```
// import function for SQS
import com.amazonaws.services.sqs.AmazonSQS;
import java.util.ArrayList;
import java.util.List;

class SQSUtils {

    // instantiate SQS client
    private AmazonSQS sqsClient;

    // load SQS object
    public SQSUtils(AmazonSQS sqsClient) {
        this.sqsClient = sqsClient;
    }

    // poll message from Amazon SQS given queue name
}
```

結果:

```
// poll message from Amazon SQS given queue name
public List<String> pollMessages(String queueName) {
    List<String> messages = new ArrayList<>();
    // poll messages from queue
    receiveMessageResult = sqsClient.receiveMessage(new
ReceiveMessageRequest(queueName));
    for (Message message : receiveMessageResult.getMessages()) {
        messages.add(message.getBody());
    }
    return messages;
}
```

インライン提案がまだ期待される結果を提供していない場合は、チャットセッションで Amazon Q Developer にコードの生成を依頼してください。

コードの生成に関するチャット

ユーザー:

Amazon SQS キューからメッセージを読み取るための Java 関数を生成する

Amazon Q:

Amazon SQS キューからメッセージを読み取る Java 関数の例を次に示します。

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
import com.amazonaws.services.sqs.model.ReceiveMessageResult;
public class SQSMessageReader {
    private AmazonSQS sqsClient;
    private String queueUrl;

    public SQSMessageReader(AmazonSQS sqsClient, String queueUrl) {
        this.sqsClient = sqsClient;
        this.queueUrl = queueUrl;
    }

    public String readMessage() {
        ReceiveMessageRequest receiveMessageRequest = new
ReceiveMessageRequest(queueUrl)
            .withMaxNumberOfMessages(1)
            .withWaitTimeSeconds(10);

        ReceiveMessageResult receiveMessageResult =
sqsClient.receiveMessage(receiveMessageRequest);

        if (receiveMessageResult.getMessages().size() > 0) {
            Message message = receiveMessageResult.getMessages().get(0);
            String messageBody = message.getBody();
            String receiptHandle = message.getReceiptHandle();

            // Process the message
            System.out.println("Received message: " + messageBody);

            // Delete the message from the queue
```

```
        sqsClient.deleteMessage(queueUrl, receiptHandle);

        return messageBody;
    } else {
        return null;
    }
}
```

このコードは、AmazonSQS クライアントとキュー URL をコンストラクタパラメータとして使用する SQSMessageReader クラスを作成します。readMessage() メソッドは、ReceiveMessageRequest を使用してキューから最大 1 つのメッセージを取得し、最大待機時間は 10 秒です。

メッセージを受信すると、メッセージ本文が出力され、DeleteMessageRequest を使用してメッセージがキューから削除されます。その後、メッセージ本文が返されます。メッセージを受信しない場合、null が返されます。

コードの生成に関する Amazon Q チャットを終了する

チャットの結果が不十分

コードの開発中に、開発者は Amazon Q に一般的な関数の生成、レコメンデーションの提供、またはコードの説明を依頼できます。プロンプト (質問やコードスニペットなど) がある場合、Amazon Q は一般的なフィードバックや期待を満たさない結果を提供することがあります。これらのシナリオでは、以下を試してください。

- Amazon Q から得られるコード生成結果を改善するために、さまざまなプロンプトを試してください。
- Amazon Q がプロンプトや質問への応答で頻繁に提供する URLs に従います。
- Amazon Q の将来のパフォーマンス向上に役立つフィードバックを提供します。

たとえば、開発者は検索拡張生成 (RAG) ソリューションを作成しており、文間のセマンティック関係を考慮してドキュメントをチャンクする必要があります。開発者は次のように Amazon Q とチャットします。

RAG ソリューションについてチャットする

ユーザー:

RAG ソリューションの場合、セマンティックチャンキングのコードを生成する

Amazon Q:

LangChain ライブラリ Python を使用して、検索拡張生成 (RAG) ソリューションのセマンティックチャンキングを実装する方法の例を次に示します。

```
from langchain.text_splitter import CharacterTextSplitter, TokenTextSplitter
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import Chroma

# Initialize the text splitter
text_splitter = TokenTextSplitter(
    chunk_size=500,
    chunk_overlap=50,
    length_function=len
)
# Initialize the embeddings model
embeddings = HuggingFaceEmbeddings()

# Create the vector store
vectorstore = Chroma(
    embedding_function=embeddings.embed_documents,
    persist_directory="./chroma-data"
)

# Chunk the text using semantic similarity
def semantic_chunk(text):
    chunks = text_splitter.split_text(text)
    embeddings = embeddings.embed_documents(chunks)
    vectorstore.add_texts(chunks, embeddings)
    return chunks
```

この例では、LangChain `TokenTextSplitter` のを使用して、入力テキストを 500 文字のチャンクに分割し、重複する 50 文字にします。HuggingFaceEmbeddings モデルは各チャンクの埋め込みを生成するために使用され、Chroma ベクトルストアはチャンクとその埋め込みを保存するために使用されます。

`semantic_chunk` 関数は入力テキストを受け取り、`text_splitter` を使用してチャンクに分割し、埋め込みモデルを使用して各チャンクの埋め込みを生成し、チャンクとその埋め込みを `vectorstore` に追加します。

このアプローチにより、入力テキストを意味的に意味のあるチャンクに分割できます。これは、RAG ソリューションで質問への回答や要約などのさまざまなタスクに役立ちます。

これは基本的な例であり、特定のユースケースとデータに基づいてパラメータ (chunk_size、chunk_overlap など) またはテキストスプリッターと埋め込みモデルの選択を調整する必要がある場合があります。

RAG ソリューションに関する Amazon Qチャットの終了

ただし、このアプローチでは、スライディングウィンドウ手法を使用してドキュメントをチャンクする基本的な方法のみを提供します。開発者は、埋め込みの測定に依存する高度な方法を求めています。これらのシナリオでは、開発者は目的に応じてプロンプトに詳細を追加する必要があります。開発者が LangChain で既知の AI21 テキストスプリッター関数を参照すると、次のチャットに示すようにレスポンスが改善されます。

RAG ソリューションについてチャットする (プロンプトを改善)

ユーザー:

上記のプロンプトは、チャンクあたりのトークンサイズに関する分割にのみ関連しています。AI21 セマンティックテキストスプリッターを使用します。AI21 セマンティックテキストを言語チェーンで分割して関数を生成します。

Amazon Q:

さて、AI21 Semantic Text Splitter with LangChain を使用してセマンティックチャンキングを実装しましょう。以下に例を示します: [1]

```
from langchain.text_splitter import AI21SemanticTextSplitter
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import Chroma

# Initialize the text splitter
text_splitter = AI21SemanticTextSplitter()

# Initialize the embeddings model
embeddings = HuggingFaceEmbeddings()

# Create the vector store
vectorstore = Chroma(
    embedding_function=embeddings.embed_documents,
    persist_directory="./chroma-data"
)

# Chunk the text using semantic similarity
```

```
def semantic_chunk(text):
    chunks = text_splitter.split_text(text)
    embeddings = embeddings.embed_documents(chunks)
    vectorstore.add_texts(chunks, embeddings)
    return chunks
```

この例では、LangChain の `AI21SemanticTextSplitter` を使用して、入力テキストを意味的に意味のあるチャンクに分割しています。HuggingFaceEmbeddings モデルはチャンクごとに埋め込みを作成し、Chromaベクトルストアはチャンクとその埋め込みを保存するために使用されます。

`semantic_chunk` 関数は入力テキストを受け取り、`text_splitter` を使用してチャンクに分割し、埋め込みモデルを使用してチャンクごとに埋め込みを作成し、チャンクとその埋め込みを `vectorstore` に追加します。

このアプローチにより、入力テキストを意味的に意味のあるチャンクに分割できます。これは、RAG ソリューションで質問への回答や要約などのさまざまなタスクに役立ちます。

特定のユースケースとデータに基づいて、パラメータまたは埋め込みモデルの選択を調整する必要があります。

ソース [1] LangChain でテキストスプリッターを使用する方法について説明します

[soosweb3.hashnode.dev > the-ultimate-langchain-series-text-splitters](https://soosweb3.hashnode.dev/the-ultimate-langchain-series-text-splitters)

RAG ソリューションに関する Amazon Q チャットの終了 (プロンプトを改善)

チャットの結果に基づいて、開発者は関数を使用して情報のソースに移動できます。

一般的に、高度なトピックや質問では、Amazon Q Developer とチャットするときにプロンプト内でより具体的なコンテキストが必要です。チャットの結果が正確ではないと思われる場合は、サムズダウンアイコンを使用して Amazon Q レスポンスに関するフィードバックを提供します。Amazon Q Developer は、フィードバックを継続的に使用して将来のリリースを改善します。肯定的な結果をもたらしたインタラクションの場合は、サムズアップアイコンを使用してフィードバックを提供すると便利です。

Amazon Q Developer FAQs

このセクションでは、コード開発での Amazon Q Developer の使用に関するよくある質問への回答を提供します。

Amazon Q Developer とは？

Amazon Q Developer は、インテリジェントなコード生成とレコメンデーションを提供することでコード開発タスクを加速するように設計された強力な生成 AI を活用したサービスです。2024 年 4 月 30 日、Amazon CodeWhisperer は Amazon Q Developer の一部になりました。

Amazon Q Developer にアクセスするにはどうすればよいですか？

Amazon Q Developer は、IntelliJ や PyCharm などの AWS Toolkits for Visual Studio Code および JetBrains IDEs の一部として使用できます。開始するには、[AWS Toolkit 最新バージョンをインストールします](#)。

Amazon Q Developer はどのプログラミング言語をサポートしていますか？

Visual Studio Code と IDE JetBrains の場合、Amazon Q Developer は Python、Java、JavaScript、TypeScript、C#、GoRust、PHP、Ruby、Kotlin、C、C++、シェルスクリプト、SQL、Scala をサポートしています。IDEs このガイドでは、Java 例として Python とに焦点を当てていますが、この概念はサポートされているすべてのプログラミング言語に適用されます。

より良いコード生成のために Amazon Q Developer にコンテキストを提供するにはどうすればよいですか？

既存のコードから始める、関連するライブラリをインポートする、クラスと関数を作成する、またはコードスケルトンを確立する。自然言語プロンプトには標準のコメントブロックを使用します。スクリプトを特定の目標に集中させ、個別の機能を関連するコンテキストを持つ個別のスクリプトにモジュール化します。詳細については、[「Amazon Q Developer でのコーディングのベストプラクティス」](#)を参照してください。

Amazon Q Developer でのインラインコード生成が正確でない場合はどうすればよいですか？

スクリプトのコンテキストを確認し、ライブラリが存在することを確認し、クラスと関数が新しいコードに関連していることを確認します。コードをモジュール化し、目的別に異なるクラスと関数を分離します。明確で具体的なプロンプトやコメントを記述します。コードの精度がまだ不明で、続行できない場合は、Amazon Q とのチャットを開始し、コードスニペットを指示とともに送信します。詳細については、[「Amazon Q Developer でのコード生成シナリオのトラブルシューティング」](#)を参照してください。

Amazon Q Developer チャット機能を使用してコードの生成とトラブルシューティングを行うにはどうすればよいですか？

Amazon Q とチャットして、一般的な関数の生成、レコメンデーションのリクエスト、コードの説明を行います。最初のレスポンスで満足できない場合は、さまざまなプロンプトを試して、提供されたURLsに従ってください。また、Amazon Q にフィードバックを提供して、将来のチャットパフォーマンスを向上させます。サムズアップアイコンとサムズダウンアイコンを使用してフィードバックを提供します。詳細については、[「チャットの例」](#)を参照してください。

Amazon Q Developer を使用する際のベストプラクティスにはどのようなものがありますか？

プロンプトに関連するコンテキスト、実験、反復を提供し、プロンプトを受け入れる前にコード提案を確認し、カスタマイズ機能を使用し、データプライバシーとコンテンツの使用ポリシーを理解します。詳細については、[「Amazon Q Developer でのコード生成のベストプラクティス」](#)および[「Amazon Q Developer でのコードレコメンデーションのベストプラクティス」](#)を参照してください。

Amazon Q Developer をカスタマイズして、自分のコードに基づいてレコメンデーションを生成できますか？

はい。Amazon Q Developer の高度な機能であるカスタマイズを使用します。カスタマイズにより、企業は独自のコードリポジトリを提供して、Amazon Q Developer がインラインコード提案を推奨できるようにします。詳細については、[「Amazon Q Developer and Resources の高度な機能」](#)を参照してください。 ???

Amazon Q Developer を使用する次のステップ

この包括的なガイドから得られた知識があれば、コーディングワークフローで Amazon Q Developer を効果的に使用できます。任意の IDE ([Visual Studio Code](#) または [JetBrains](#)) AWS Toolkit にをインストールし、生成 AI を活用したコード生成と Amazon Q Developer からのレコメンデーションの探索を開始します。

Amazon Q Developer の可能性を最大限に引き出す最も効果的な方法は、独自のコードを使用した実践的な経験です。Amazon Q を開発ライフサイクルに統合する際、ベストプラクティス、トラブルシューティング、実際の例については、このガイドを参照してください。

さらに、[リソース](#)で参照されている AWS ブログと開発者ガイドを確認して、最新情報を入手してください。これらのリソースは、Amazon Q Developer の使用を最適化するのに役立つ最新の更新、ベストプラクティス、インサイトを提供します。

お客様からのフィードバックは、このガイドを改善し、開発者にとって貴重なリソースであり続けるために役立ちます。今後のバージョンに関する経験、課題、提案を共有します。入力、ニーズに合わせた追加の例、トラブルシューティングシナリオ、インサイトでガイドを強化するのに役立ちます。

リソース

AWS ブログ

- [Amazon Q でソフトウェア開発ライフサイクルを加速](#)
- [Amazon Q Developer Agent を使用したソフトウェア開発の再考](#)
- [Amazon Q の 5 つのトラブルシューティング例](#)
- [プライベートコードベースIDEを使用して Amazon Q Developer をカスタマイズする](#)
- [コード変換用の Amazon Q Developer エージェントで Java アップグレードを高速化する 3 つの方法](#)
- [Amazon Q Developer を活用してコードのデバッグとメンテナンスを効率化する](#)
- [Amazon Q Developer を使用したアプリケーションのテスト](#)

AWS ドキュメント

- [Amazon Q Developer ユーザーガイド](#)
- [Amazon Q Developer コードのカスタマイズ](#)
- [Amazon Q Developer コード変換](#)

AWS ワークショップ

- [Amazon Q デベロッパーイマージョンデー](#)
- [Amazon Q Developer Workshop - Q-Words アプリの構築](#)
- [Amazon Q Developer Workshop - 効果的なプロンプトの作成](#)

寄稿者

以下の個人がこのガイドに貢献しました。

- JoeKing、シニアデータサイエンティスト、AWS
- Prateek Gupta、チームリード – シニア CAA、AWS
- Manohar Reddy Arranagu、DevOps アーキテクト、AWS
- Soumik Roy、クラウドアプリケーションアーキテクト、AWS
- Sanket Shinde、コンサルタント、AWS

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。今後の更新について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 |
|----------------------|----|-----------------|
| 初版発行 | — | 2024 年 8 月 16 日 |

AWS 規範ガイドの用語集

以下は、AWS 規範ガイドが提供する戦略、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 AWS するための、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の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ウラーなど、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н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н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н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 AWS のサービス 受信する前に、ローカルでデータを暗号化します。

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。信頼できる ID のみが、期待されるネットワークから信頼できる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。この戦略を採用するときは AWS、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て、リソースの安全性を確保します。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「[Disaster Recovery of Workloads on AWS: Recovery in the Cloud](#)」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、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で、障害の影響を制限し、ワークロードの耐障害性を向上させるのに役立ちます。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

laC

「[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パスワードやユーザー認証情報などの AWS Secrets Manager 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

送信先にあるデータの、それ AWS のサービスを受け取る による暗号化。

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、お客様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「[を他の AWS のサービス AWS Organizations で使用する 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。詳細については、[深層学習システムにおける不確実性の定量化ガイド](#)を参照してください。

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。