での六角形アーキテクチャの構築 AWS

AWS 規範ガイダンス

AWS 規範ガイダンス: での六角形アーキテクチャの構築 AWS

Table of Contents

での六角形アーキテクチャの構築 AWS

Furkan Oruc、Dominik Goby、Darius Kunce、Michal Ploski、Amazon Web Services (AWS)

2022 年 6 月 (ドキュメント履歴)

このガイドでは、ソフトウェアアーキテクチャを開発するためのメンタルモデルとパターンのコレクションについて説明します。これらのアーキテクチャは、製品の導入が進むにつれて、組織全体で簡単に維持、拡張、スケーリングできます。Amazon Web Services (AWS) などのクラウドハイパースケーラーは、中小企業や大企業が新しいソフトウェア製品を革新および作成するための構成要素を提供します。これらの新しいサービスと機能の導入の急速なペースにより、ビジネスステークホルダーは、開発チームが新しい最小実行可能製品 (MVPs) のプロトタイプを迅速に作成することを期待し、新しいアイデアをできるだけ早くテストして検証できます。多くの場合、これらの MVPsは採用され、エンタープライズソフトウェアエコシステムの一部になります。これらの MVPs、チームは SOLID の原則や単体テストなどのソフトウェア開発ルールやベストプラクティスを中止することがあります。このアプローチは開発を高速化し、市場投入までの時間を短縮することを前提としています。ただし、基盤モデルとソフトウェアアーキテクチャのフレームワークをすべてのレベルで作成できない場合、製品の新機能を開発することは困難または不可能です。確実性の欠如や要件の変化も、開発プロセス中にチームを遅らせる可能性があります。

このガイドでは、低レベルの六角形アーキテクチャから高レベルのアーキテクチャと組織の分解まで、ドメイン駆動型設計 (DDD) を使用してこれらの課題に対処する、提案されたソフトウェアアーキテクチャについて説明します。DDD は、ビジネスの複雑さを管理し、新機能の開発に応じてエンジニアリングチームを拡張するのに役立ちます。ユビキタス言語を使用することで、ビジネス関係者と技術関係者をドメインと呼ばれるビジネス上の問題に合わせます。六角形アーキテクチャは、境界コンテキストと呼ばれる非常に特定のドメインにおけるこのアプローチの技術的なイネーブラーです。境界コンテキストは、ビジネス問題の非常にまとまりがあり、疎結合されたサブエリアです。複雑さに関係なく、すべてのエンタープライズソフトウェアプロジェクトに六角形アーキテクチャを採用することをお勧めします。

六角形アーキテクチャは、エンジニアリングチームが最初にビジネス上の問題を解決することを奨励しますが、従来のレイヤードアーキテクチャはエンジニアリングの焦点をドメインから遠ざけて、最初に技術的な問題を解決します。さらに、ソフトウェアが六角形アーキテクチャに従う場合、テスト駆動型の開発アプローチを採用する方が簡単です。これにより、デベロッパーがビジネス要件をテストするために必要なフィードバックループが短縮されます。最後に、コマンドとコマンドハンドラーを使用することは、SOLID から単一の責任とオープンクローズの原則を適用する方法です。これらの原則に従うことで、プロジェクトに取り組む開発者やアーキテクトが簡単にナビゲートして理解できるコードベースが生成され、既存の機能に重大な変更を加えるリスクが軽減されます。

このガイドは、ソフトウェア開発プロジェクトに六角形アーキテクチャと DDD を採用することの利点を理解したいソフトウェアアーキテクトやデベロッパーを対象としています。これには、六角形アーキテクチャをサポートする 上のアプリケーションのインフラストラクチャ AWS を設計する例が含まれています。実装例については、AWS 規範ガイダンスウェブサイトの「 を使用して六角形アーキテクチャで Python プロジェクト AWS Lambdaを構築する」を参照してください。

概要

ドメイン駆動型設計 (DDD)

ドメイン駆動型設計 (DDD) では、ドメインはソフトウェアシステムの中核です。ドメインモデルは、他のモジュールを開発する前に最初に定義され、他の低レベルモジュールに依存しません。代わりに、データベース、プレゼンテーションレイヤー、外部 APIsなどのモジュールはすべてドメインに依存します。

DDD では、アーキテクトは技術的な分解ではなくビジネスロジックベースの分解を使用して、ソリューションを境界のあるコンテキストに分解します。このアプローチの利点については、ターゲットを絞ったビジネス成果「」セクションで説明します。

チームが六角形アーキテクチャを使用する場合、DDD は実装が容易です。六角形アーキテクチャでは、アプリケーションコアがアプリケーションの中心です。ポートとアダプターを介して他のモジュールから分離され、他のモジュールに依存しません。これは DDD と完全に一致します。ここで、ドメインはビジネス上の問題を解決するアプリケーションの中核です。このガイドでは、六角形アーキテクチャの中核を境界コンテキストのドメインモデルとしてモデル化するアプローチを提案します。次のセクションでは、六角形アーキテクチャについて詳しく説明します。

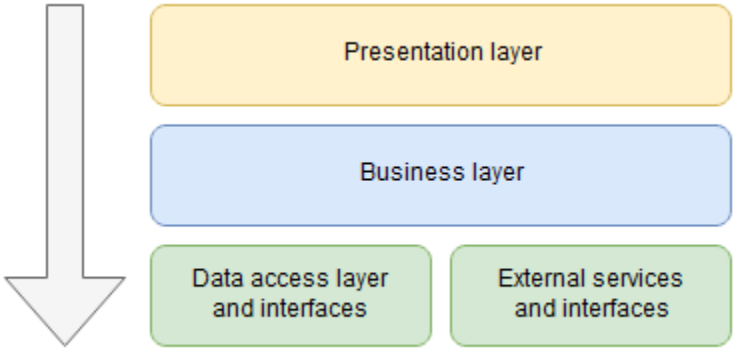
このガイドでは、非常に広範なトピックである DDD のすべての側面を網羅しているわけではありません。ドメイン言語ウェブサイトにリストされている DDD リソースを確認して、理解を深めることができます。

六角形アーキテクチャ

六角形アーキテクチャは、ポートとアダプター、またはオニオンアーキテクチャとも呼ばれ、ソフトウェアプロジェクトの依存関係の反転を管理する原則です。六角形アーキテクチャは、ソフトウェアの開発時にコアドメインのビジネスロジックに重点を置き、外部統合ポイントをセカンダリとして扱います。六角形アーキテクチャは、ソフトウェアエンジニアがテスト駆動型開発 (TDD) などのベストプラクティスを採用するのに役立ちます。これにより、アーキテクチャの進化が促進され、複雑なドメインを長期的に管理できるようになります。

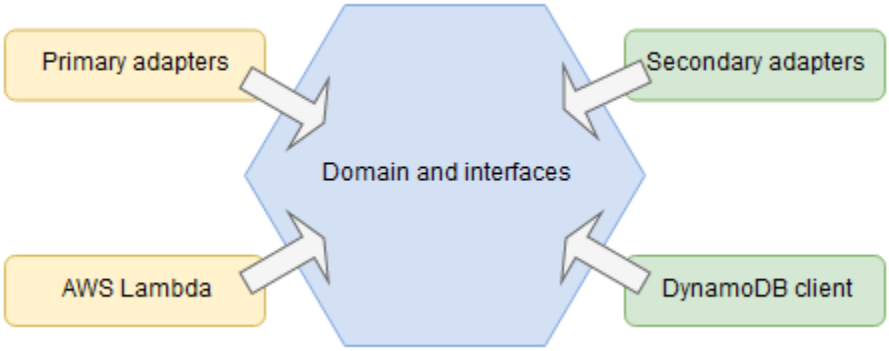
六角形アーキテクチャと従来のレイヤードアーキテクチャを比較してみましょう。これは、構造化ソフトウェアプロジェクトのモデリングで最も人気のある選択肢です。2 つのアプローチには微妙ですが強力な違いがあります。

階層型アーキテクチャでは、ソフトウェアプロジェクトは階層構造になっており、ビジネスロジックやプレゼンテーションロジックなど、幅広い懸念事項を表しています。このアーキテクチャでは、上位レイヤーが下位レイヤーに依存する依存関係階層を使用しますが、逆はできません。次の図では、プレゼンテーションレイヤーがユーザーインタラクションを担当するため、ユーザーインターフェイス、APIs、コマンドラインインターフェイス、および同様のコンポーネントが含まれます。プレゼンテーションレイヤーは、ドメインロジックを実装するビジネスレイヤーに依存しています。ビジネスレイヤーには、データアクセスレイヤーと複数の外部サービスへの依存関係があります。



この設定の主な欠点は、依存関係構造です。例えば、データベースにデータを保存するモデルが変更されると、データアクセスインターフェイスに影響します。データモデルを変更すると、データアクセスインターフェイスに依存するビジネスレイヤーにも影響します。その結果、ソフトウェアエンジニアはドメインロジックに影響を与えずにインフラストラクチャを変更することはできません。これにより、リグレッションバグの可能性が高まります。

次の図に示すように、六角形アーキテクチャでは、依存関係を別の方法で定義します。すべてのインターフェイスを定義するドメインビジネスロジックを中心に意思決定を集中的に行います。外部コンポーネントは、ポートと呼ばれるインターフェイスを介してビジネスロジックとやり取りします。ポートは、ドメインと外部世界の相互作用を定義する抽象化です。各インフラストラクチャコンポーネントはこれらのポートを実装する必要があるため、これらのコンポーネントの変更がコアドメインロジックに影響を与えなくなります。



周囲のコンポーネントはアダプターと呼ばれます。アダプターは、外部ワールドと内部ワールド間のプロキシであり、ドメインで定義されたポートを実装します。アダプターは、プライマリとセカンダリの 2 つのグループに分類できます。プライマリアダプターは、ソフトウェアコンポーネントへのエントリポイントです。これにより、外部のアクター、ユーザー、およびサービスがコアロジックとやり取りできるようになります。 AWS Lambda はプライマリアダプターの良い例です。Lambda 関数をエントリポイントとして呼び出すことができる複数の AWS サービスと統合されます。セカンダリアダプターは、外部ワールドとの通信を処理する外部サービスライブラリラッパーです。セカンダリアダプターの例として、データアクセス用の Amazon DynamoDB クライアントがあります。

ターゲットを絞ったビジネス成果

このガイドで説明する六角形アーキテクチャは、次の目的を達成するのに役立ちます。

- 開発サイクルを改善することで市場投入までの時間を短縮する
- ソフトウェア品質の向上
- より簡単に変更できる

これらのプロセスについては、以下のセクションで詳しく説明します。

開発サイクルの改善

クラウド用のソフトウェアを開発すると、ランタイム環境を開発マシンでローカルにレプリケートすることが非常に難しいため、ソフトウェアエンジニアにとって新しい課題が生じます。ソフトウェアを検証する簡単な方法は、クラウドにデプロイしてテストすることです。ただし、このアプローチでは、特にソフトウェアアーキテクチャに複数のサーバーレスデプロイが含まれている場合、フィードバックサイクルが長くなります。このフィードバックサイクルを改善すると、機能の開発時間が短縮され、市場投入までの時間が大幅に短縮されます。

クラウドでのテスト

クラウドで直接テストすることは、Amazon API Gateway のゲートウェイ、 AWS Lambda 関数、Amazon DynamoDB テーブル、 AWS Identity and Access Management (IAM) アクセス許可などのアーキテクチャコンポーネントが正しく設定されていることを確認する唯一の方法です。また、コンポーネント統合をテストする唯一の信頼できる方法である可能性もあります。一部の AWS サービス (DynamoDB など) はローカルにデプロイできますが、ほとんどのサービスはローカルセットアップではレプリケートできません。同時に、テスト目的でサービスを模 AWS 倣する Moto や LocalStack などのサードパーティーツールには、実際のサービス API 契約が正確に反映されていないか、機能の数が制限されている可能性があります。

ただし、エンタープライズソフトウェアの最も複雑な部分は、クラウドアーキテクチャではなくビジネスロジックにあります。アーキテクチャはドメインよりも頻繁に変更されないため、新しいビジネス要件を満たす必要があります。したがって、クラウドでビジネスロジックをテストすることは、コードに変更を加え、デプロイを開始し、環境の準備が整うのを待って、変更を検証する集中的なプロセスになります。デプロイに 5 分しかかからない場合、ビジネスロジックで 10 回の変更とテストには 1 時間以上かかります。ビジネスロジックがより複雑な場合、テストにはデプロイが完了するまで数日かかることがあります。チームに複数の機能とエンジニアがいる場合、延長期間はすぐにビジネスにとって目立つものになります。

ローカルでのテスト

六角形アーキテクチャは、デベロッパーがインフラストラクチャの技術ではなくドメインに集中するのに役立ちます。このアプローチでは、ローカルテスト (選択した開発フレームワークのユニットテストツール) を使用して、ドメインロジック要件に対応します。技術的な統合の問題の解決に時間を費やす必要も、ビジネスロジックをテストするためにソフトウェアをクラウドにデプロイする必要もありません。ユニットテストをローカルで実行し、フィードバックループを数分から数秒に短縮でき

ます。デプロイに 5 分かかり、ユニットテストが 5 秒で完了する場合、ミスの検出にかかる時間は大幅に短縮されます。このガイドの後半最初にビジネスロジックをテストするのセクションでは、このアプローチについて詳しく説明します。

開発の並列化

六角形アーキテクチャアプローチにより、開発チームは開発作業を並列化できます。開発者は、サービスのさまざまなコンポーネントを個別に設計および実装できます。この並列化は、各コンポーネントと各コンポーネント間の定義済みインターフェイスを分離することで可能です。

製品市場投入までの時間

ローカルユニットテストは、開発フィードバックサイクルを改善し、特に前述のように複雑なビジネスロジックが含まれている場合、新しい製品や機能の市場投入までの時間を短縮します。さらに、ユニットテストによってコードカバレッジが増加すると、コードベースを更新またはリファクタリングするときにリグレッションバグが発生するリスクが大幅に軽減されます。ユニットテストカバレッジでは、コードベースを継続的にリファクタリングして整理しやすくすることもできるため、新しいエンジニアのオンボーディングプロセスが高速化されます。これは、設計による品質「」セクションで詳しく説明します。最後に、ビジネスロジックが適切に分離され、テストされている場合、開発者は変化する機能要件と非機能要件に迅速に適応できます。これは、変更への適応「」セクションで詳しく説明します。

設計による品質

六角形アーキテクチャを採用することで、プロジェクトの最初からコードベースの品質を高めることができます。開発プロセスを遅らせることなく、期待される品質要件を最初から満たすのに役立つプロセスを構築することが重要です。

ローカライズされた変更と読みやすさの向上

六角形アーキテクチャアプローチを使用すると、デベロッパーは他のクラスやコンポーネントに影響を与えることなく、1 つのクラスやコンポーネントのコードを変更できます。この設計により、開発されたコンポーネントの結合が促進されます。ドメインをアダプターから切り離し、よく知られているインターフェイスを使用することで、コードの読みやすさを高めることができます。問題やコーナーケースを識別しやすくなります。

このアプローチは、開発中のコードレビューを容易にし、未検出の変更や技術的負債の導入を制限します。

最初にビジネスロジックをテストする

ローカルテストは、end-to-end、統合、ユニットテストをプロジェクトに導入することで実現できます。End-to-endのテストは、受信リクエストのライフサイクル全体を対象としています。通常、アプリケーションエントリポイントを呼び出し、ビジネス要件を満たしているかどうかをテストします。各ソフトウェアプロジェクトには、既知の入力を使用し、期待される出力を生成するテストシナリオが少なくとも 1 つ必要です。ただし、各テストはエントリポイント (REST API やキューなど) を介してリクエストを送信するように設定する必要があるため、コーナーケースシナリオを追加すると複雑になる可能性があります。ビジネスアクションに必要なすべての統合ポイントを通過し、結果をアサートします。テストシナリオの環境を設定し、結果をアサートするには、デベロッパーに多くの時間がかかる場合があります。

六角形アーキテクチャでは、ビジネスロジックを個別にテストし、統合テストを使用してセカンダリアダプターをテストします。ビジネスロジックテストでは、模擬アダプターまたはフェイクアダプターを使用できます。また、ビジネスユースケースのテストとドメインモデルのユニットテストを組み合わせて、結合率が低く高いカバレッジを維持することもできます。統合テストではビジネスロジックを検証しないでください。代わりに、セカンダリアダプターが外部サービスを正しく呼び出すことを確認する必要があります。

理想的には、テスト駆動型開発 (TDD) を使用して、開発の開始時に適切なテストでドメインエンティティまたはビジネスユースケースの定義を開始できます。最初にテストを記述すると、ドメイ

ンに必要なインターフェイスのモック実装を作成するのに役立ちます。テストが成功し、ドメインロジックルールが満たされたら、実際のアダプターを実装し、ソフトウェアをテスト環境にデプロイできます。この時点では、ドメインロジックの実装が理想的ではない可能性があります。その後、設計パターンを導入するか、コードを一般的に再配置することで、既存のアーキテクチャのリファクタリングに取り組み、それを進化させることができます。このアプローチを使用することで、リグレッションバグの発生を回避し、プロジェクトの拡大に合わせてアーキテクチャを改善できます。このアプローチを継続的な統合プロセスで実行する自動テストと組み合わせることで、本番稼働前に潜在的なバグの数を減らすことができます。

サーバーレスデプロイを使用する場合、手動統合とend-to-endテストのために、 AWS アカウントのアプリケーションのインスタンスをすばやくプロビジョニングできます。これらの実装手順の後、リポジトリにプッシュされる新しい変更をすべて実行してテストを自動化することをお勧めします。

保守性

保守性とは、アプリケーションを運用およびモニタリングして、すべての要件を満たし、システム障害の可能性を最小限に抑えることです。システムを運用可能にするには、将来のトラフィックや運用要件に適応させる必要があります。また、クライアントへの影響を最小限に抑えながら、またはまったく影響を与えずに、利用可能かつ簡単にデプロイできることを確認する必要があります。

システムの現在の状態と履歴状態を理解するには、それを観察可能にする必要があります。これを行うには、システムが期待どおりに動作し、バグを追跡するためにオペレーターが使用できる特定のメトリクス、ログ、トレースを指定します。これらのメカニズムにより、オペレーターはマシンにログインしてコードを読み取ることなく、根本原因の分析を実行することもできます。

六角形アーキテクチャは、ウェブアプリケーションの保守性を高め、コード全体の作業を減らすことを目的としています。モジュールを分離し、変更をローカライズし、アプリケーションのビジネスロジックをアダプター実装から切り離すことで、オペレーターがシステムをより深く理解し、プライマリアダプターまたはセカンダリアダプターに加えられた特定の変更の範囲を理解するのに役立つメトリクスとログを生成できます。

変更への適応

ソフトウェアシステムは複雑になる傾向があります。その理由の 1 つは、ビジネス要件を頻繁に変更し、それに応じてソフトウェアアーキテクチャを適応させる時間が少ないことです。もう 1 つの理由は、プロジェクトの開始時にソフトウェアアーキテクチャを設定して頻繁な変更に適応するための投資が不十分であることです。理由に関係なく、ソフトウェアシステムは、変更を行うことがほとんど不可能なほど複雑になる可能性があります。したがって、メンテナンス可能なソフトウェアアーキテクチャをプロジェクトの最初から構築することが重要です。優れたソフトウェアアーキテクチャは、変更に簡単に適応できます。

このセクションでは、非機能要件やビジネス要件に簡単に適応する六角形アーキテクチャを使用して、保守可能なアプリケーションを設計する方法について説明します。

ポートとアダプターを使用した新しい非機能要件への適応

アプリケーションの中核となるドメインモデルは、ビジネス要件を満たすために外部から必要なアクションを定義します。これらのアクションは、ポートと呼ばれる抽象化によって定義されます。これらのポートは個別のアダプターによって実装されます。各アダプターは、別のシステムとのやり取りを担当します。たとえば、データベースリポジトリ用のアダプターが 1 つあり、サードパーティー API とやり取りするためのアダプターがもう 1 つあるとします。ドメインはアダプターの実装を認識していないため、あるアダプターを別のアダプターに置き換えるのは簡単です。たとえば、アプリケーションは SQL データベースから NoSQL データベースに切り替えることができます。この場合、ドメインモデルで定義されたポートを実装するために、新しいアダプターを開発する必要があります。ドメインにはデータベースリポジトリへの依存関係がなく、抽象化を使用してやり取りするため、ドメインモデルで何も変更する必要はありません。したがって、六角形アーキテクチャは非機能的な要件に簡単に適応します。

コマンドとコマンドハンドラーを使用した新しいビジネス要件への適応

クラシックレイヤードアーキテクチャでは、ドメインは永続性レイヤーに依存します。ドメインを変更する場合は、永続化レイヤーも変更する必要があります。これに対して、六角形アーキテクチャでは、ドメインはソフトウェアの他のモジュールに依存しません。ドメインはアプリケーションのコアであり、他のすべてのモジュール (ポートとアダプター) はドメインモデルによって異なります。ドメインは、依存関係の逆転の原則を使用して、ポートを介して外部と通信します。依存関係の反転の利点は、コードの他の部分を壊すことを恐れることなく、ドメインモデルを自由に変更できることで

す。ドメインモデルは解決しようとしているビジネス上の問題を反映しているため、変化するビジネス要件に適応するようにドメインモデルを更新することは問題ではありません。

ソフトウェアを開発する場合、懸念の分離は従うべき重要な原則です。この分離を実現するには、わずかに変更されたコマンドパターンを使用できます。これは、オペレーションを完了するために必要なすべての情報がコマンドオブジェクトにカプセル化される動作設計パターンです。これらのオペレーションはコマンドハンドラーによって処理されます。コマンドハンドラーは、コマンドを受信し、ドメインの状態を変更して、呼び出し元にレスポンスを返すメソッドです。同期 APIs や非同期キューなど、さまざまなクライアントを使用してコマンドを実行できます。ドメインに対するオペレーションごとに、コマンドとコマンドハンドラーを使用することをお勧めします。このアプローチに従うことで、既存のビジネスロジックを変更することなく、新しいコマンドとコマンドハンドラーを導入することで、新機能を追加できます。したがって、コマンドパターンを使用すると、新しいビジネス要件に適応しやすくなります。

サービスファサードまたは CQRS パターンを使用したコンポーネントのデカップリング

六角形アーキテクチャでは、プライマリアダプターがクライアントからの受信読み取りおよび書き込みリクエストをドメインに疎結合します。この疎結合を実現するには、サービスファサードパターンを使用するか、コマンドクエリ責任分離 (CQRS) パターンを使用するという 2 つの方法があります。

サービスファサードパターンは、プレゼンテーションレイヤーやマイクロサービスなどのクライアントに対応する前面インターフェイスを提供します。サービスファサードは、クライアントに複数の読み取りおよび書き込みオペレーションを提供します。受信リクエストをドメインに転送し、ドメインから受信したレスポンスをクライアントにマッピングします。サービスファサードの使用は、複数のオペレーションで単一の責任を持つマイクロサービスにとって簡単です。ただし、サービスファサードを使用する場合、単一の責任とオープンクローズの原則に従うことは困難です。単一責任の原則では、各モジュールがソフトウェアの 1 つの機能のみに対して責任を負う必要があると規定されています。オープンクローズの原則では、拡張のためにコードを開き、変更のためにクローズする必要があります。サービスファサードが拡張されると、すべてのオペレーションが 1 つのインターフェイスに収集され、より多くの依存関係がそのインターフェイスにカプセル化され、より多くのデベロッパーが同じファサードの変更を開始します。したがって、開発中にサービスがあまり拡張されないことが明らかな場合にのみ、サービスファサードを使用することをお勧めします。

六角形アーキテクチャでプライマリアダプターを実装するもう 1 つの方法は、クエリとコマンドを使用して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する CQRS パターンを使用する

ことです。前述のように、コマンドはドメインの状態を変更するために必要なすべての情報を含むオブジェクトです。コマンドは、コマンドハンドラーメソッドによって実行されます。一方、クエリはシステムの状態を変更しません。唯一の目的は、データをクライアントに返すことです。CQRS パターンでは、コマンドとクエリは別々のモジュールに実装されます。これは、非同期的に処理されるイベントとしてコマンドを実装でき、API を使用してクエリを同期的に実行できるため、イベント駆動型アーキテクチャに従うプロジェクトに特に便利です。クエリには、最適化された別のデータベースを使用することもできます。CQRS パターンの欠点は、サービスファサードよりも実装に時間がかかることです。長期的にスケーリングして維持する予定のプロジェクトには、CQRS パターンを使用することをお勧めします。コマンドとクエリは、単一責任の原則を適用し、特に大規模なプロジェクトで疎結合ソフトウェアを開発するための効果的なメカニズムを提供します。

CQRS には長期的には大きな利点がありますが、初期投資が必要です。このため、CQRS パターンを使用する前にプロジェクトを慎重に評価することをお勧めします。ただし、読み取り/書き込みオペレーションを分離することなく、最初からコマンドとコマンドハンドラーを使用してアプリケーションを構造化できます。これにより、後でそのアプローチを採用することを決定した場合、CQRS のプロジェクトを簡単にリファクタリングできます。

組織のスケーリング

六角形アーキテクチャ、ドメイン駆動型設計、および (オプションで) CQRS の組み合わせにより、組織は製品を迅速にスケーリングできます。Conway の法則によると、ソフトウェアアーキテクチャは企業のコミュニケーション構造を反映するように進化する傾向があります。この観測には歴史的に否定的な意味があります。これは、大規模な組織がデータベースやエンタープライズサービスバスなどの技術的な専門知識に基づいてチームを編成することがよくあるためです。このアプローチの問題は、製品と機能の開発には常にセキュリティやスケーラビリティなどのクロスカットの懸念が伴い、チーム間の継続的なコミュニケーションが必要であることです。技術的な機能に基づいてチームを構築すると、組織に不要なサイロが発生し、コミュニケーションが悪くなり、所有権が不足し、全体像が見えなくなります。最終的に、これらの組織の問題はソフトウェアアーキテクチャに反映されます。

一方、逆方向コンウェイマヌーバーは、ソフトウェアアーキテクチャを促進するドメインに基づいて組織構造を定義します。例えば、部門横断的なチームには、DDD とイベントストーミングを使用して識別される特定の一連の境界コンテキストに対する責任が与えられます。 https://www.eventstorming.com/これらの境界コンテキストは、製品の非常に具体的な機能を反映している可能性があります。たとえば、アカウントチームが支払いコンテキストを担当する場合があります。各新機能は、緊密に連携し、疎結合された責任を持つ新しいチームに割り当てられるため、その機能の提供のみに集中し、市場投入までの時間を短縮できます。チームは機能の複雑さに応じてスケーリングできるため、複雑な機能をより多くのエンジニアに割り当てることができます。

ベストプラクティス

ビジネスドメインのモデル化

ビジネスドメインからソフトウェア設計に戻り、記述しているソフトウェアがビジネスニーズに合っていることを確認します。

イベントストーミングなどのドメイン駆動型設計 (DDD) 手法を使用して、ビジネスドメインをモデル化します。イベントストーミングには柔軟なワークショップ形式があります。ワークショップでは、ドメインとソフトウェアの専門家がビジネスドメインの複雑さを共同で調査します。ソフトウェアの専門家は、ワークショップの成果物を使用して、ソフトウェアコンポーネントの設計と開発プロセスを開始します。

最初からテストを記述して実行する

テスト駆動型開発 (TDD) を使用して、開発中のソフトウェアが正しいことを確認します。TDD はユニットテストレベルで最適に機能します。開発者は、最初にテストを記述してソフトウェアコンポーネントを設計し、そのコンポーネントを呼び出します。そのコンポーネントには最初に実装がないため、テストは失敗します。次のステップとして、開発者はモックオブジェクトを含むテストフィクスチャを使用してコンポーネントの機能を実装し、外部依存関係またはポートの動作をシミュレートします。テストが成功すると、開発者は実際のアダプターを実装することで続行できます。このアプローチにより、開発者はユーザーがコンポーネントをどのように使用するかを理解できるため、ソフトウェアの品質が向上し、コードがより読みやすくなります。六角形アーキテクチャは、アプリケーションコアを分離することで TDD 手法をサポートします。開発者は、ドメインコアの動作に焦点を当てたユニットテストを記述します。テストを実行するために複雑なアダプターを記述する必要はありません。代わりに、シンプルな模擬オブジェクトやフィクスチャを使用できます。

動作駆動型開発 (BDD) を使用して、機能レベルでend-to-endの承諾を確保します。BDD では、開発者は機能のシナリオを定義し、ビジネスステークホルダーと検証します。BDD テストでは、これを実現するためにできるだけ多くの自然言語を使用します。六角アーキテクチャは、プライマリアダプターとセカンダリアダプターの概念で BDD 方法論をサポートします。開発者は、外部サービスを呼び出すことなくローカルで実行できるプライマリアダプターとセカンダリアダプターを作成できます。ローカルプライマリアダプターを使用してアプリケーションを実行するように BDD テストスイートを設定します。

継続的インテグレーションパイプラインで各テストを自動的に実行して、システムの品質を常に評価します。

ドメインの動作を定義する

ドメインをエンティティ、値オブジェクト、集計に分解し (ドメイン駆動型設計の実装について読む)、その動作を定義します。プロジェクトの開始時に記述されたテストが成功するように、ドメインの動作を実装します。ドメインオブジェクトの動作を呼び出すコマンドを定義します。動作の完了後にドメインオブジェクトが出力するイベントを定義します。

アダプターがドメインとやり取りするために使用できるインターフェイスを定義します。

テストとデプロイを自動化する

最初の概念実証の後、DevOps プラクティスの実装に時間を費やすことをお勧めします。たとえば、継続的インテグレーションおよび継続的デリバリー (CI/CD) パイプラインや動的テスト環境は、コードの品質を維持し、デプロイ中のエラーを回避するのに役立ちます。

- CI プロセス内でユニットテストを実行し、マージする前にコードをテストします。
- CD プロセスを構築して、アプリケーションを静的な開発/テスト環境または自動統合とend-to-endのテストをサポートする動的に作成された環境にデプロイします。
- 専用環境のデプロイプロセスを自動化します。

マイクロサービスと CQRS を使用して製品をスケールする

製品が成功したら、ソフトウェアプロジェクトをマイクロサービスに分解して製品をスケールします。六角形アーキテクチャが提供する移植性を活用してパフォーマンスを向上させます。クエリサービスとコマンドハンドラーを別々の同期 API と非同期 APIs。コマンドクエリ責任分離 (CQRS) パターンとイベント駆動型アーキテクチャを採用することを検討してください。

多くの新機能リクエストが発生した場合は、DDD パターンに基づいて組織をスケーリングすることを検討してください。組織のスケーリング「」セクションで前述したように、1 つ以上の機能を境界コンテキストとして所有するようにチームを構成します。これらのチームは、六角形アーキテクチャを使用してビジネスロジックを実装できます。

六角形アーキテクチャの概念にマッピングするプロジェクト構造を設計する

Infrastructure as Code (IaC) は、クラウド開発で広く採用されているプラクティスです。これにより、インフラストラクチャリソース (ネットワーク、ロードバランサー、仮想マシン、ゲートウェイなど) をソースコードとして定義して維持できます。これにより、バージョン管理システムを使用して、アーキテクチャに対するすべての変更を追跡できます。さらに、テスト目的でインフラストラクチャを簡単に作成して移動できます。クラウドアプリケーションを開発するときは、アプリケーションコードとインフラストラクチャコードを同じリポジトリに保持することをお勧めします。このアプローチにより、アプリケーションのインフラストラクチャを簡単に維持できます。

アプリケーションは、六角形アーキテクチャの概念 (プライマリアダプター) 、 `entrypoints` (domainドメインとインターフェイス) 、 `adapters` (セカンダリアダプター) の 3 つのフォルダまたはプロジェクトに分割することをお勧めします。

次のプロジェクト構造は、 API を設計する際のこのアプローチの例を示しています AWS。プロジェクトは、前述のように、アプリケーションコード (app) とインフラストラクチャコード (infra) を同じリポジトリに保持します。

```
app/  # application code
|--- adapters/  # implementation of the ports defined in the domain
     |--- tests/  # adapter unit tests
|--- entrypoints/  # primary adapters, entry points
     |--- api/  # api entry point
          |--- model/  # api model
          |--- tests/  # end to end api tests
|--- domain/  # domain to implement business logic using hexagonal architecture
     |--- command_handlers/  # handlers used to run commands on the domain
     |--- commands/  # commands on the domain
     |--- events/  # events emitted by the domain
     |--- exceptions/  # exceptions defined on the domain
     |--- model/  # domain model
     |--- ports/  # abstractions used for external communication
     |--- tests/  # domain tests
infra/  # infrastructure code
```

前述のように、ドメインはアプリケーションのコアであり、他のモジュールに依存しません。domain フォルダには、次のサブフォルダを含めることをお勧めします。

- command handlers には、ドメインでコマンドを実行するメソッドまたはクラスが含まれています。
- commands には、ドメインで オペレーションを実行するために必要な情報を定義するコマンドオブジェクトが含まれています。
- events には、ドメインを介して出力され、他のマイクロサービスにルーティングされるイベントが含まれます。
- exceptions には、ドメイン内で定義された既知のエラーが含まれています。
- model には、ドメインエンティティ、値オブジェクト、ドメインサービスが含まれます。
- ports には、ドメインがデータベース、APIs、またはその他の外部コンポーネントと通信するための抽象化が含まれています。
- tests には、ドメインで実行されるテスト方法 (ビジネスロジックテストなど) が含まれています。

プライマリアダプターは、 entrypointsフォルダで表されるアプリケーションへのエントリポイントです。この例では、 apiフォルダをプライマリアダプターとして使用します。このフォルダにはmodel、プライマリアダプターがクライアントと通信するために必要なインターフェイスを定義する API が含まれています。tests フォルダには、 API end-to-endのテストが含まれています。これらは、アプリケーションのコンポーネントが統合され、連携して動作することを検証する浅いテストです。

セカンダリアダプターは、 adaptersフォルダで表されるとおり、ドメインポートに必要な外部統合を実装します。データベースリポジトリは、セカンダリアダプターの優れた例です。データベースシステムが変更されたら、ドメインで定義された実装を使用して新しいアダプターを記述できます。ドメインやビジネスロジックを変更する必要はありません。tests サブフォルダには、各アダプターの外部統合テストが含まれています。

でのインフラストラクチャの例 AWS

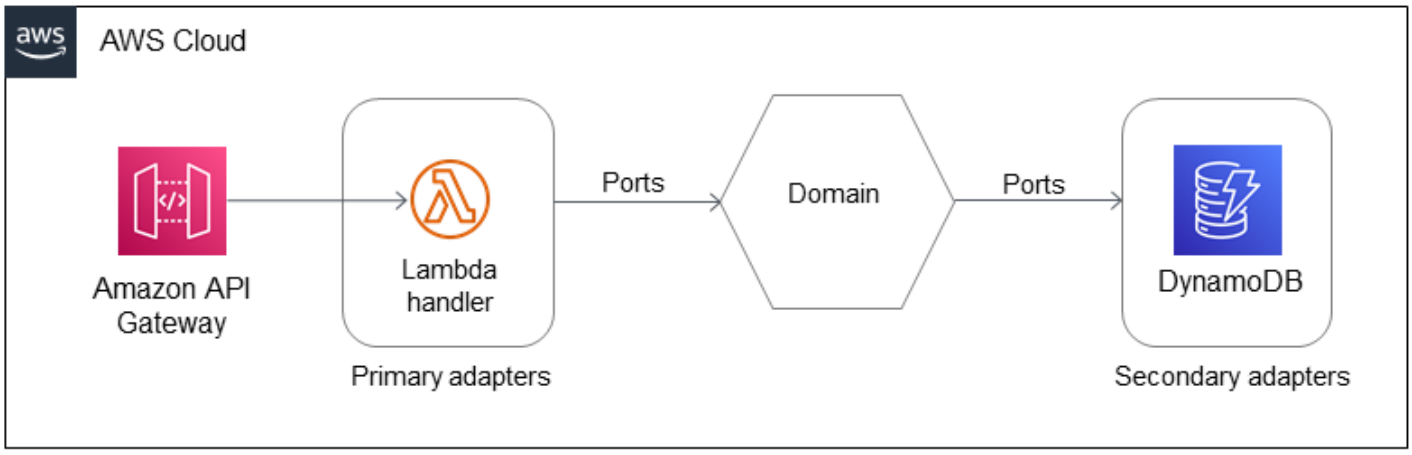
このセクションでは、六角形アーキテクチャの実装に使用できる の AWS アプリケーションのインフラストラクチャを設計する例を示します。最小実行可能な製品 (MVP) を構築するためのシンプルなアーキテクチャから始めることをお勧めします。ほとんどのマイクロサービスでは、クライアントリクエストを処理する単一のエントリポイント、コードを実行するコンピューティングレイヤー、データを保存するための永続化レイヤーが必要です。以下の AWS サービスは、六角形アーキテクチャのクライアント、プライマリアダプター、セカンダリアダプターとして使用するのに最適です。

- クライアント: Amazon API Gateway、Amazon Simple Queue Service (Amazon SQS)、Elastic Load Balancing、Amazon EventBridge
- プライマリアダプター: AWS Lambda、Amazon Elastic Container Service (Amazon ECS)、Amazon Elastic Kubernetes Service (Amazon EKS)、Amazon Elastic Compute Cloud (Amazon EC2)
- セカンダリアダプター: Amazon DynamoDB、Amazon Relational Database Service (Amazon RDS)、Amazon Aurora、API Gateway、Amazon SQS、Elastic Load Balancing、EventBridge、Amazon Simple Notification Service (Amazon SNS)

以下のセクションでは、六角形アーキテクチャのコンテキストにおけるこれらのサービスについて詳しく説明します。

シンプルに開始する

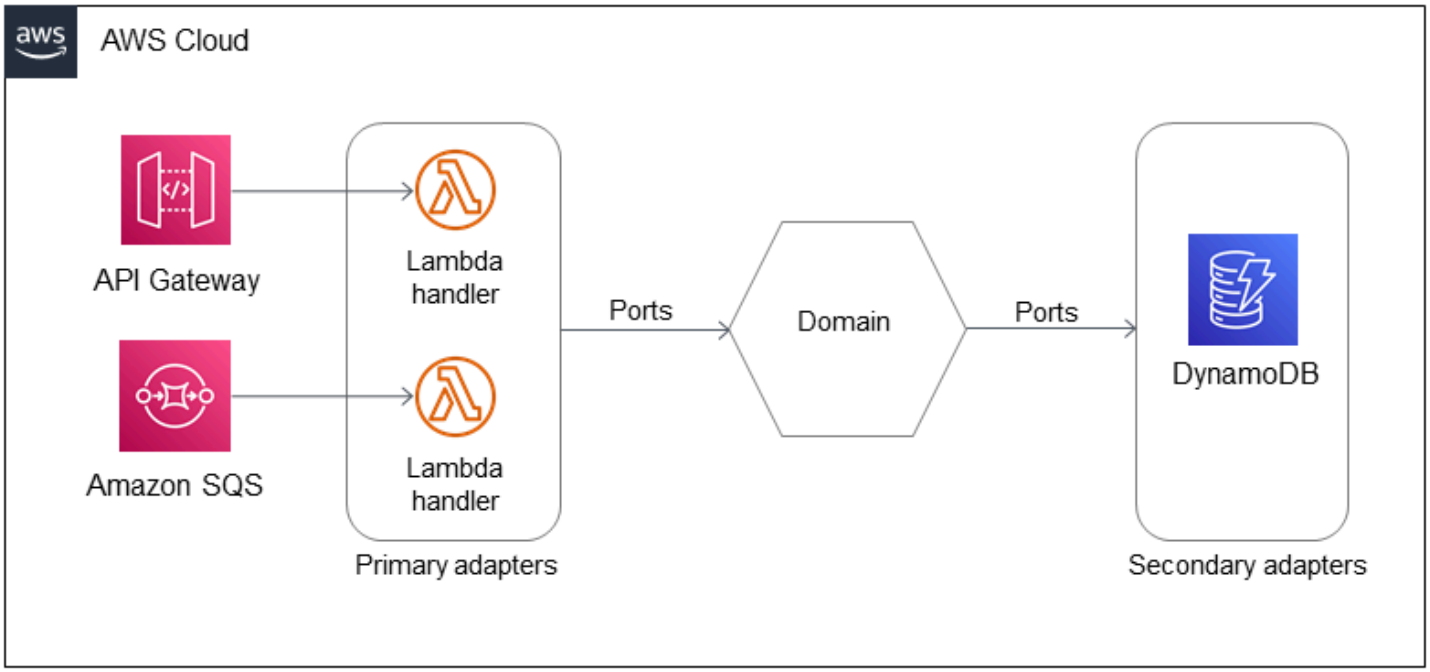
六角形アーキテクチャを使用してアプリケーションを設計するときは、簡単に開始することをお勧めします。この例では、API Gateway をクライアント (REST API) として使用し、Lambda をプライマリアダプター (コンピューティング) として使用し、DynamoDB をセカンダリアダプター (永続性) として使用します。ゲートウェイクライアントはエントリポイントを呼び出します。この場合は Lambda ハンドラーです。



このアーキテクチャは完全にサーバーレスであり、アーキテクトに良い出発点を与えます。ドメインでコマンドパターンを使用することをお勧めします。これは、コードのメンテナンスが容易になり、新しいビジネス要件や非機能要件に適応するためです。このアーキテクチャは、いくつかのオペレーションでシンプルなマイクロサービスを構築するのに十分です。

CQRS パターンを適用する

ドメインのオペレーション数がスケールする場合は、CQRS パターンに切り替えることをお勧めします。次の例を使用して、 で AWS CQRS パターンを完全にサーバーレスアーキテクチャとして適用できます。

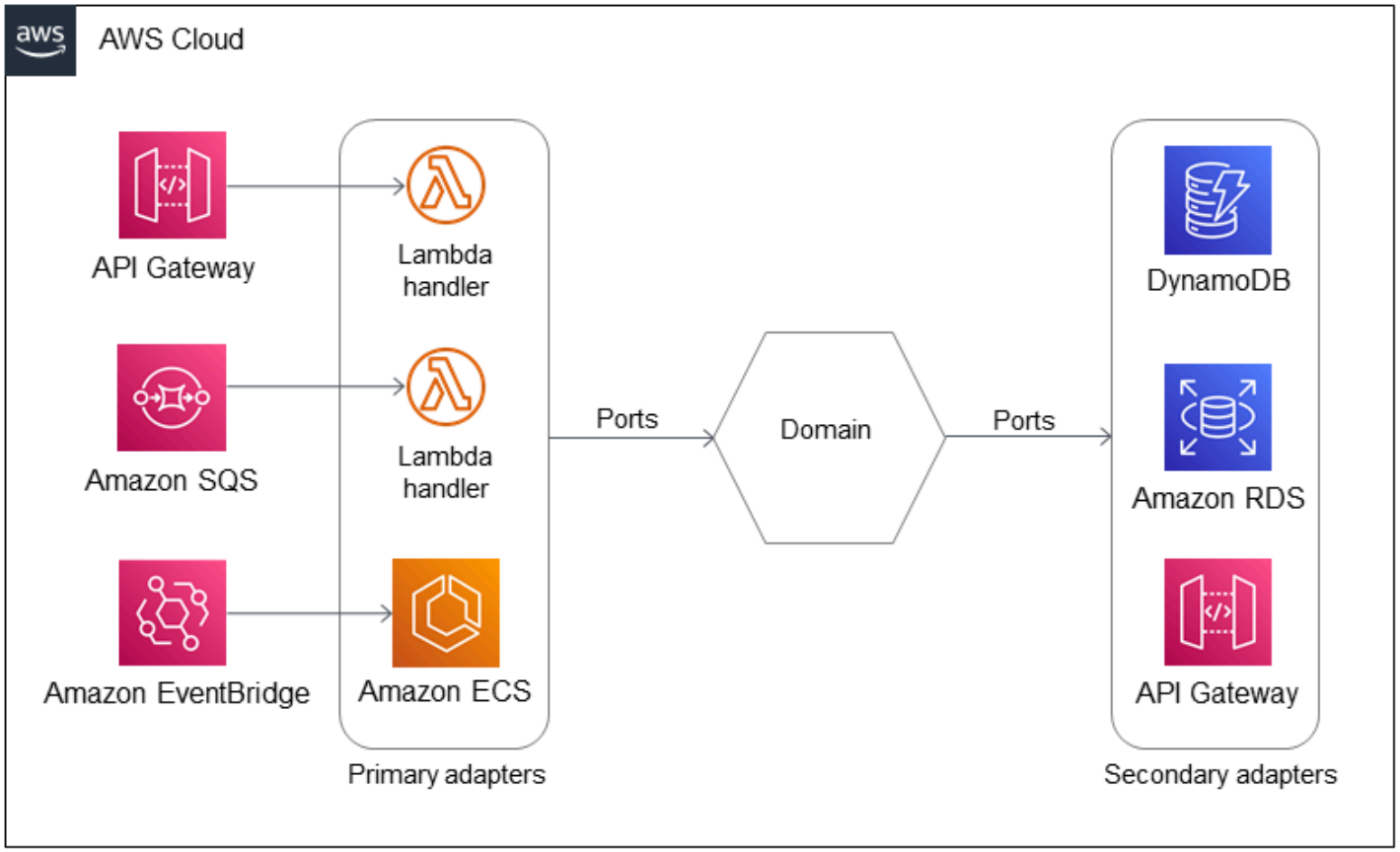


この例では、2 つの Lambda ハンドラーを使用します。1 つはクエリ用、もう 1 つはコマンド用です。クエリは、API ゲートウェイをクライアントとして使用して同期的に実行されます。コマンドは、Amazon SQS をクライアントとして使用して非同期的に実行されます。

このアーキテクチャには、複数のクライアント (API Gateway と Amazon SQS) と、対応するエントリポイント (Lambda ハンドラー) によって呼び出される複数のプライマリアダプター (Lambda) が含まれます。すべてのコンポーネントは同じ境界コンテキストに属するため、同じドメイン内にあります。

コンテナ、リレーショナルデータベース、外部 API を追加してアーキテクチャを進化させる

コンテナは、長時間実行されるタスクに適したオプションです。事前定義されたデータスキーマがあり、SQL 言語の能力を活用する場合は、リレーショナルデータベースを使用することもできます。さらに、ドメインは外部 APIsと通信する必要があります。次の図に示すように、これらの要件をサポートするようにアーキテクチャの例を進化させることができます。

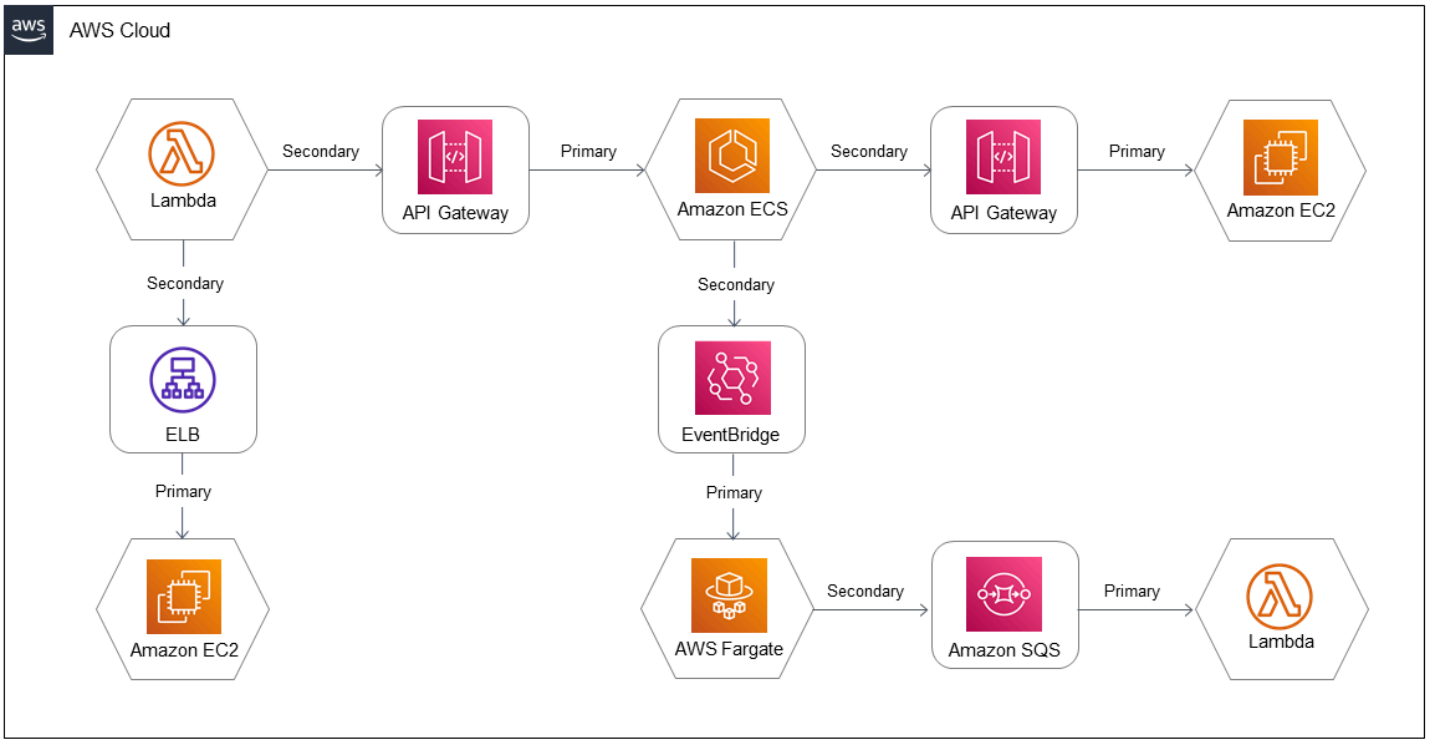


この例では、ドメインで長時間実行されるタスクを起動するためのプライマリアダプターとして Amazon ECS を使用します。Amazon EventBridge (クライアント) は、特定のイベントが発生したときに Amazon ECS タスク (エントリポイント) を開始します。このアーキテクチャには、リレーショナルデータを保存するための別のセカンダリアダプターとして Amazon RDS が含まれています。また、外部 API コールを呼び出すためのセカンダリアダプターとして別の API ゲートウェイも追加されます。その結果、アーキテクチャは、1 つのビジネスドメイン内のさまざまな基盤となるコンピューティングレイヤーに依存する複数のプライマリアダプターとセカンダリアダプターを使用します。

ドメインは常に、ポートと呼ばれる抽象化を通じて、すべてのプライマリアダプターとセカンダリアダプターと疎結合されます。ドメインは、ポートを使用して外部から必要なものを定義します。ポートを実装するのはアダプターの責任であるため、あるアダプターから別のアダプターに切り替えてもドメインには影響しません。たとえば、ドメインに影響を与えずに新しいアダプターを記述することで、Amazon DynamoDB から Amazon RDS に切り替えることができます。

さらにドメインを追加する (ズームアウト)

六角形アーキテクチャは、マイクロサービスアーキテクチャの原則とよく一致しています。ここに示したアーキテクチャの例には、単一のドメイン (または境界コンテキスト) が含まれていました。アプリケーションには通常、プライマリアダプターとセカンダリアダプターを介して通信する必要がある複数のドメインが含まれます。各ドメインはマイクロサービスを表し、他のドメインと疎結合されています。



このアーキテクチャでは、各ドメインは異なるコンピューティング環境のセット (複数可) を使用します。(各ドメインには、前の例のように複数のコンピューティング環境がある場合もあります) 。各ドメインは、ポートを介して他のドメインと通信するために必要なインターフェイスを定義します。ポートは、プライマリアダプターとセカンダリアダプターを使用して実装されます。これにより、アダプターに変更があった場合、ドメインは影響を受けません。さらに、ドメインは互いに分離されます。

前の図に示すアーキテクチャ例では、Lambda、Amazon EC2、Amazon ECS、 AWS Fargate がプライマリアダプターとして使用されます。API Gateway、Elastic Load Balancing、EventBridge、Amazon SQS はセカンダリアダプターとして使用されます。

よくある質問

六角形アーキテクチャを使用する理由

六角形アーキテクチャは、開発者の焦点をドメインロジックに移行し、テストの自動化を簡素化し、コードの品質と適応性を向上させます。これらの改善により、市場投入までの時間が短縮され、技術的および組織的なスケーリングが容易になります。

ドメイン駆動型設計を使用する理由

ドメイン駆動型設計 (DDD) では、ビジネスステークホルダーとエンジニアの間で共通言語を使用することで、ソフトウェアコンポーネントとコンストラクトを構築できます。DDD はソフトウェアの複雑さの管理に役立ち、ソフトウェア製品を長期的に維持するための効果的な戦略です。

六角形アーキテクチャなしでテスト駆動型開発を実践できますか?

はい。テスト駆動型開発 (TDD) は、特定のソフトウェア設計パターンに限定されません。ただし、六角形アーキテクチャにより、TDD の練習が容易になります。

六角形アーキテクチャとドメイン駆動型設計なしで製品をスケールできますか?

はい。技術的および組織的な製品のスケーリングは、ほとんどの設計パターンで実現できます。ただし、六角形アーキテクチャと DDD により、拡張が容易になり、長期的には大規模なプロジェクトに対してより効果的です。

六角形アーキテクチャを実装するには、どのテクノロジーを使用する必要がありますか?

六角アーキテクチャは、特定のテクノロジースタックに限定されません。依存関係の反転と単体テストをサポートするテクノロジーを選択することをお勧めします。

最小限の実行可能な製品を開発しています。ソフトウェアアーキテクチャについて考えることに時間を費やすことは理にかなっていますか?

はい。MVPs には使い慣れた設計パターンを使用することをお勧めします。エンジニアが慣れるまで、六角形アーキテクチャを試してみることをお勧めします。新しいプロジェクトに六角形アーキテクチャを確立しても、アーキテクチャなしで開始するよりも大幅に大きな時間投資は必要ありません。

最小限の実行可能な製品を開発しており、テストを作成する時間がない。

MVP にビジネスロジックが含まれている場合は、自動テストを作成することを強くお勧めします。これにより、フィードバックループが短縮され、時間が節約されます。

六角形アーキテクチャでは、どのような追加の設計パターンを使用できますか?

CQRS パターンを使用して、システム全体のスケーリングをサポートします。リポジトリパターンを使用して、ドメインモデルを保存および復元します。作業パターンの単位を使用して、トランザクションプロセスステップを管理します。継承に対してコンポジションを使用して、ドメインの集計、エンティティ、値オブジェクトをモデル化します。複雑なオブジェクト階層を構築しないでください。

次のステップ

- リソース「」セクションで収集されたリンクを読んで、ドメイン駆動型設計の概念をさらに理解してください。
- 新しいプロジェクトを実装する場合は、このガイドに記載されているプロジェクト構造テンプレートを使用して、いくつかの機能を実装します。
- 既存のプロジェクトを実装している場合は、読み取り専用オペレーションと書き込み専用オペレーションに分割できるコードを特定します。読み取り専用コードをクエリサービスに抽象化し、書き込み専用コードをコマンドハンドラーに配置します。
- 基本プロジェクト構造が整ったら、ユニットテストを記述し、テスト自動化との継続的な統合 (CI) を確立し、テスト駆動型開発 (TDD) のプラクティスに従います。

リソース

リファレンス

- (AWS 規範ガイダンスパターン) を使用して六角形アーキテクチャで Python プロジェクトを構築する AWS Lambda
- Agile Teams (Scaled Agile Framework ウェブサイト)
- Harry Percival と Bob Gregory (O'Reilly Media、2020 年 3 月 31 日) による Python を使用したアーキテクチャパターン、特に以下の章。
  - コマンドとコマンドハンドラー
  - コマンドクエリ責任分離 (CQRS)
  - リポジトリパターン
- イベントストーミング: Alberto Brandolini (Event Storming ウェブサイト) によるサイロの境界を超えたコラボレーションへの最もスマートなアプローチ
- Sam Newman による Conway の法則の説明 (Thoughtworks ウェブサイト、2014 年 6 月 30 日)
- James Beswick による進化アーキテクチャの開発 AWS Lambda (AWS コンピューティングブログ、2021 年 7 月 8 日)
- Domain Language: Tackling Complexity in the Heart of Software (ドメイン言語ウェブサイト)
- Facade, from Dive Into Design Patterns by Alexander Shvets (ebook, December 5, 2018)
- GivenWhenThen、Martin Fowler (2013 年 8 月 21 日)
- Vaughn Vernon によるドメイン駆動設計の実装 (Addison-Wesley Professional、2013 年 2 月)
- 逆方向コンウェイマヌーバー (Thoughtworks ウェブサイト、2014 年 7 月 8 日)
- 月のパターン: Red Green Refactor (DZone ウェブサイト、2017 年 6 月 2 日)
- SOLID Design Principles Explained: Dependency Inversion Principle with Code Examples (Stackify website, May 7, 2018)
- SOLID 原則: Simon Hoiberg による説明と例 (ITNEXT ウェブサイト、2019 年 1 月 1 日)
- 「The Art of Agile Development: Test-Driven Development」、James Shore と Shane Warden (O'Reilly Media、2010 年 3 月 25 日)
- Yigit Kemal Erinc によるプレーン英語で説明されるオブジェクト指向プログラミングの SOLID 原則 (freeCodeCamp オブジェクト指向プログラミング投稿、2020 年 8 月 20 日)
- イベント駆動型アーキテクチャとは (AWS ウェブサイト)

AWS サービス

- Amazon API Gateway
- Amazon Aurora
- Amazon DynamoDB
- Amazon Elastic Compute Cloud (Amazon EC2)
- Amazon Elastic Container Service (Amazon ECS)
- Amazon Elastic Kubernetes Service (Amazon EKS)
- エラスティックロードバランシング
- Amazon EventBridge
- AWS Fargate
- AWS Lambda
- Amazon Relational Database Service (Amazon RDS)
- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)

その他のツール

- ムート
- LocalStack

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
初版発行	—	2022 年 6 月 15 日

AWS 規範ガイダンスの用語集

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

数字

7 Rs

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

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

A

ABAC

「属性ベースのアクセス制御」をご覧ください。

抽象化されたサービス

「マネージドユーザー」をご覧ください。

ACID

「原子性、一貫性、分離性、耐久性 (ACID)」をご覧ください。

アクティブ/アクティブ移行

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

アクティブ/パッシブ移行

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

集計関数

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

AI

「人工知能」をご覧ください。

AIOps

「AI オペレーション」をご覧ください。

匿名化

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

アンチパターン

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

アプリケーション制御

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

アプリケーションポートフォリオ

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

人工知能 (AI)

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

AI オペレーション (AIOps)

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

非対称暗号化

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

原子性、一貫性、分離性、耐久性 (ACID)

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

属性ベースのアクセス制御 (ABAC)

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

信頼できるデータソース

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

アベイラビリティーゾーン (AZ)

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

AWS クラウド導入フレームワーク (AWS CAF)

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

AWS ワークロード認定フレームワーク (AWS WQF)

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

B

不正なボット

個人や組織に混乱や損害を与えることを目的としたボット。

BCP

「ビジネス継続性計画 (BCP)」をご覧ください。

動作グラフ

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

ビッグエンディアンシステム

最上位バイトを最初に格納するシステム。「エンディアン性」もご覧ください。

二項分類

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

ブルームフィルター

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

ブルー/グリーンデプロイ

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

ボット

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

ボットネット

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

ブランチ

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

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

ブレークグラスアクセス

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

ブラウンフィールド戦略

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

バッファキャッシュ

アクセス頻度が最も高いデータが保存されるメモリ領域。

ビジネス能力

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

ビジネス継続性計画 (BCP)

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

C

CAF

「AWS クラウド導入フレームワーク」を参照してください

カナリアデプロイ

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

CCoE

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

CDC

「変更データキャプチャ」を参照してください。

変更データキャプチャ (CDC)

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

カオスエンジニアリング

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

CI/CD

「継続的インテグレーションと継続的デリバリー」を参照してください。

分類

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

クライアント側の暗号化

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

Cloud Center of Excellence (CCoE)

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

クラウドコンピューティング

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

クラウド運用モデル

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

導入のクラウドステージ

組織が、 AWS クラウドへの移行時に通常実行する 4 つの段階。

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

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

CMDB

「構成管理データベース (CMDB)」を参照してください。

コードリポジトリ

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

コールドキャッシュ

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

コールドデータ

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

コンピュータビジョン (CV)

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

設定ドリフト

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

構成管理データベース (CMDB)

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

コンフォーマンスパック

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

継続的インテグレーションと継続的デリバリー (CI/CD)

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

CV

「コンピュータビジョン」を参照してください。

D

保管中のデータ

ストレージ内にあるデータなど、常に自社のネットワーク内にあるデータ。

データ分類

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

データドリフト

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

転送中のデータ

ネットワーク内 (ネットワークリソース間など) を活発に移動するデータ。

データメッシュ

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

データ最小化

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

データ境界

AWS 環境内の一連の予防ガードレール。信頼できる 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、リソースの保護に役立つように、 AWS Organizations 構造の異なるレイヤーに複数のコントロールを追加します。たとえば、多層防御アプローチでは、多要素認証、ネットワークセグメンテーション、暗号化を組み合わせることができます。

委任管理者

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

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

トラブルシューティング

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

開発環境

「環境」を参照してください。

検出管理

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

開発バリューストリームマッピング (DVSM)

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

デジタルツイン

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

ディメンションテーブル

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

ディザスタ

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

ディザスタリカバリ (DR)

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

DML

「データベース操作言語」を参照してください。

ドメイン駆動型設計

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

DR

「ディザスタリカバリ」を参照してください。

ドリフト検出

ベースライン設定からの偏差を追跡します。たとえば、 AWS CloudFormation を使用してシステムリソースのドリフトを検出したり、 を使用して AWS Control Tower 、ガバナンス要件のコンプライアンスに影響を与える可能性のあるランディングゾーンの変更を検出したりできます。

DVSM

「開発バリューストリームマッピング」を参照してください。

E

EDA

「探索的データ分析」を参照してください。

EDI

「電子データ交換」を参照してください。

エッジコンピューティング

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

電子データ交換 (EDI)

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

暗号化

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

暗号化キー

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

エンディアン

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

エンドポイント

「サービスエンドポイント」を参照してください。

エンドポイントサービス

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

エンタープライズリソースプランニング (ERP)

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

エンベロープ暗号化

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

環境

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

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

エピック

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

ERP

「エンタープライズリソース計画」を参照してください。

探索的データ分析 (EDA)

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

F

ファクトテーブル

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

フェイルファスト

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

障害分離境界

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

機能ブランチ

「ブランチ」を参照してください。

特徴量

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

特徴量重要度

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

機能変換

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

数ショットプロンプト

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

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

FGAC

「きめ細かなアクセス制御」を参照してください。

きめ細かなアクセス制御 (FGAC)

複数の条件を使用してアクセス要求を許可または拒否すること。

フラッシュカット移行

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

FM

「基盤モデル」を参照してください。

基盤モデル (FM)

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

G

生成 AI

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

ジオブロッキング

「地理的制限」を参照してください。

地理的制限 (ジオブロッキング)

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

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

Gitflow ワークフロー

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

ゴールデンイメージ

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

グリーンフィールド戦略

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

ガードレール

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

H

HA

「高可用性」を参照してください。

異種混在データベースの移行

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

高可用性 (HA)

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

ヒストリアンのモダナイゼーション

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

ホールドアウトデータ

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

同種データベースの移行

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

ホットデータ

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

ホットフィックス

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

ハイパーケア期間

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

I

IaC

「Infrastructure as Code」を参照してください。

ID ベースのポリシー

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

アイドル状態のアプリケーション

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

IIoT

「インダストリアル IoT」を参照してください。

イミュータブルインフラストラクチャ

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

インバウンド (受信) VPC

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

増分移行

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

インダストリー 4.0

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

インフラストラクチャ

アプリケーションの環境に含まれるすべてのリソースとアセット。

Infrastructure as Code (IaC)

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

インダストリアル IoT (IIoT)

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

インスペクション VPC

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

IoT

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

解釈可能性

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

IoT

「IoT」を参照してください。

IT 情報ライブラリ (ITIL)

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

IT サービス管理 (ITSM)

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

ITIL

「IT 情報ライブラリ」を参照してください。

ITSM

「IT サービス管理」を参照してください。

L

ラベルベースアクセス制御 (LBAC)

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

ランディングゾーン

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

大規模言語モデル (LLM)

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

大規模な移行

300 台以上のサーバの移行。

LBAC

「ラベルベースアクセス制御」を参照してください。

最小特権

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

リフトアンドシフト

「7 Rs」を参照してください。

リトルエンディアンシステム

最下位バイトを最初に格納するシステム。「エンディアン性」もご覧ください。

LLM

「大規模言語モデル」を参照してください。

下位環境

「環境」を参照してください。

M

機械学習 (ML)

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

メインブランチ

「ブランチ」を参照してください。

マルウェア

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

マネージドサービス

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

製造実行システム (MES)

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

MAP

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

メカニズム

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

メンバーアカウント

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

MES

「製造実行システム」を参照してください。

Message Queuing Telemetry Transport (MQTT)

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

マイクロサービス

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

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

マイクロサービスアーキテクチャ

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

Migration Acceleration Program (MAP)

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

大規模な移行

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

移行ファクトリー

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

移行メタデータ

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

移行パターン

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

Migration Portfolio Assessment (MPA)

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

移行準備状況評価 (MRA)

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

移行戦略

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

ML

「機械学習」を参照してください。

モダナイゼーション

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

モダナイゼーション準備状況評価

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

モノリシックアプリケーション (モノリス)

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

MPA

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

MQTT

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

多クラス分類

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

ミュータブルなインフラストラクチャ

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

O

OAC

「オリジンアクセス制御」を参照してください。

OAI

「オリジンアクセスアイデンティティ」を参照してください。

OCM

「組織変更管理」を参照してください。

オフライン移行

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

OI

「オペレーション統合」を参照してください。

Ola

「オペレーショナルレベルアグリーメント」を参照してください。

オンライン移行

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

OPC-UA

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

Open Process Communications - Unified Architecture (OPC-UA)

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

オペレーショナルレベルアグリーメント (OLA)

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

運用準備状況レビュー (ORR)

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

運用テクノロジー (OT)

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

オペレーション統合 (OI)

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

組織の証跡

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

組織変更管理 (OCM)

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

オリジンアクセス制御 (OAC)

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

オリジンアクセスアイデンティティ (OAI)

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

ORR

「運用準備状況レビュー」を参照してください。

OT

「運用テクノロジー」を参照してください。

アウトバウンド (送信) VPC

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

P

アクセス許可の境界

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

個人を特定できる情報 (PII)

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

PII

「個人を特定できる情報」を参照してください。

プレイブック

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

PLC

「プログラマブルロジックコントローラー」を参照してください。

PLM

「製品ライフサイクル管理」を参照してください。

ポリシー

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

多言語の永続性

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

ポートフォリオ評価

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

述語

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

述語プッシュダウン

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

予防的コントロール

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

プリンシパル

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

プライバシーバイデザイン

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

プライベートホストゾーン

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

プロアクティブコントロール

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

製品ライフサイクル管理 (PLM)

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

本番環境

「環境」を参照してください。

プログラマブルロジックコントローラー (PLC)

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

プロンプトチェイニング

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

仮名化

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

発行/サブスクライブ (pub/sub)

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

Q

クエリプラン

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

クエリプランのリグレッション

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

R

RACI マトリックス

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

RAG

「検索拡張生成」を参照してください。

ランサムウェア

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

RASCI マトリックス

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

RCAC

「行と列のアクセス制御」を参照してください。

リードレプリカ

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

リアーキテクト

「7 Rs」を参照してください。

目標復旧時点 (RPO)

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

目標復旧時間 (RTO)

サービスの中断から復旧までの最大許容遅延時間。

リファクタリング

「7 Rs」を参照してください。

リージョン

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

リグレッション

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

リホスト

「7 Rs」を参照してください。

リリース

デプロイプロセスで、変更を本番環境に昇格させること。

再配置

「7 Rs」を参照してください。

リプラットフォーム

「7 Rs」を参照してください。

再購入

「7 Rs」を参照してください。

回復性

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

リソースベースのポリシー

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

実行責任者、説明責任者、協業先、報告先 (RACI) に基づくマトリックス

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

レスポンシブコントロール

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

保持

「7 Rs」を参照してください。

廃止

「7 Rs」を参照してください。

検索拡張生成 (RAG)

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

ローテーション

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

行と列のアクセス制御 (RCAC)

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

RPO

「目標復旧時点」を参照してください。

RTO

「目標復旧時間」を参照してください。

ランブック

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

S

SAML 2.0

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

SCADA

「監視制御とデータ取得」を参照してください。

SCP

「サービスコントロールポリシー」を参照してください。

シークレット

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

セキュリティバイデザイン

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

セキュリティコントロール

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

セキュリティ強化

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

Security Information and Event Management (SIEM) システム

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

セキュリティレスポンスの自動化

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

サーバー側の暗号化

送信先で、それ AWS のサービス を受け取る によるデータの暗号化。

サービスコントロールポリシー (SCP)

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

サービスエンドポイント

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

サービスレベルアグリーメント (SLA)

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

サービスレベルインジケータ (SLI)

エラー率、可用性、スループットといった、サービスパフォーマンス面の指標。

サービスレベル目標 (SLO)

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

責任共有モデル

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

SIEM

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

単一障害点 (SPOF)

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

SLA

「サービスレベルアグリーメント」を参照してください。

SLI

「サービスレベルインジケータ」を参照してください。

SLO

「サービスレベルの目標」を参照してください。

スプリットアンドシードモデル

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

SPOF

「単一障害点」を参照してください。

スタースキーマ

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

strangler fig パターン

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

サブネット

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

監視制御とデータ取得 (SCADA)

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

対称暗号化

データの暗号化と復号に同じキーを使用する暗号化のアルゴリズム。

合成テスト

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

システムプロンプト

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

T

タグ

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

ターゲット変数

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

タスクリスト

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

テスト環境

「環境」を参照してください。

トレーニング

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

トランジットゲートウェイ

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

トランクベースのワークフロー

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

信頼されたアクセス

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

チューニング

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

ツーピザチーム

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

U

不確実性

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