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

上に六角形アーキテクチャを構築AWS

ファーカン・オルク、ドミニク・ゴビー、ダリウス・クンス、ミハル・プロスキ、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一連のパターンについて説明します。これらのアーキテクチャは、製品の採用が進むにつれて、組織全体で簡単に保守、拡張、拡張できます。Amazon Web Services (AWS) などのクラウドハイパースケーラーは、中小企業から大企業まで、さまざまな企業がイノベーションを起こして新しいソフトウェア製品を開発するための基盤となります。このような新しいサービスや機能の導入が急速に進んでいるため、ビジネス関係者は、新しいアイデアをできるだけ早くテストして検証できるように、開発チームが新しいMVP (Minimum Viable Products) をより早くプロトタイプ化することを開発チームに期待するようになっています。多くの場合、これらの MVP 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となります。こうした MVP を作成する過程で、チームが 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放棄することがあります。彼らは、このアプローチが開発をスピードアップし、市場投入までの時間を短縮すると考えています。しかし、すべてのレベルでソフトウェアアーキテクチャの基礎モデルとフレームワークを作成できなければ、製品の新機能の開発は困難になるか、不可能にさえなります。確実性の欠如や要件の変化は、開発プロセスの進行を遅らせることにもなりかねません。

このガイドでは、提案されているソフトウェアアーキテクチャについて、低レベルの六角形アーキテクチャから、ドメイン駆動設計 (DDD) を使用してこれらの課題に対処する高レベルのアーキテクチャと組織的な分解までを順を追って説明します。DDDは、ビジネスの複雑さを管理し、新機能が開発されるたびにエンジニアリングチームを拡大するのに役立ちます。ユビキタスな言葉を使うことで、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結びつけます。ヘキサゴナル・アーキテクチャは、バウンディッド・コンテクストと呼ばれる非常に特殊な領域において、このアプローチを技術的に可能にします。境界付きコンテキストとは、ビジネス上の問題の中でもまとまりが強く、結合が緩いサブエリアです。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は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ビジネス上の問題を最初に解決することを奨励しますが、従来の階層型アーキテクチャでは、エンジニアリングの焦点はドメインから技術的な問題を解決することにシフトします。さらに、ソフトウェアが六角形のアーキテクチャに従うと、テスト駆動型開発アプローチを採用しやすくなり、開発者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減ります。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の単

一責任とオープンクローズドな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作成され、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開発者を対象としています。AWS六角形アーキテクチャをサポートする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も含まれています。実装例については、AWS Prescriptive Guidance ウェブサイトの「PythonAWS Lambda プロジェクトを使用して六角形アーキテクチャを構築する」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は依存しません。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 API などのモジュールはすべてドメインに依存しています。

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、ソリューションを限定的なコンテキストに分解します。この方法の利点については、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。

DDD は、チームが六角形のアーキテクチャを使用する方が簡単に実装できます。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。ポートとアダプタを介して他のモジュールから切り離され、他のモジュールに依存しません。これは、ビジネス上の問題を解決するアプリケーションの中核をドメインとするDDDと完全に一致します。このガイドでは、六角形アーキテクチャの中核を境界のある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、六角形のアーキテクチャについて詳しく説明します。

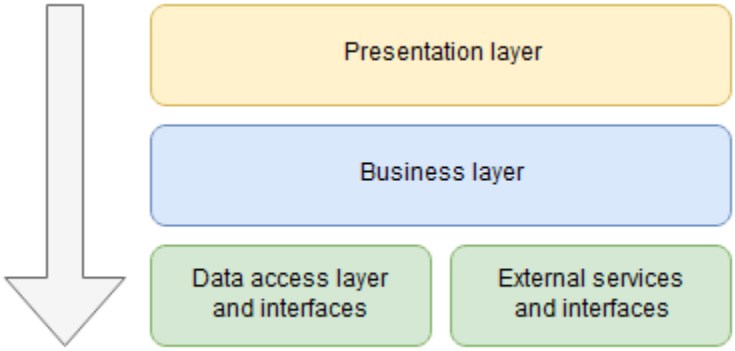
このガイドでは、非常に幅広いトピックである DDD のすべてを網羅しているわけではありません。理解を深めるには、ドメイン言語 Web サイトにリストされている DDD リソースを確認してください。

六角アーキテクチャ

ポートアンドアダプターまたはオニオンアーキテクチャとも呼ばれるヘキサゴナルアーキテクチャは、ソフトウェアプロジェクトにおける依存関係の逆転を管理するための原則です。ヘキサゴナルアーキテクチャでは、ソフトウェア開発時にコアドメインのビジネスロジックに重点が置かれ、外部統合ポイントは二次的なものとして扱われます。ヘキサゴナルアーキテクチャは、ソフトウェアエンジニアがテスト駆動開発 (TDD) などの優れた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。

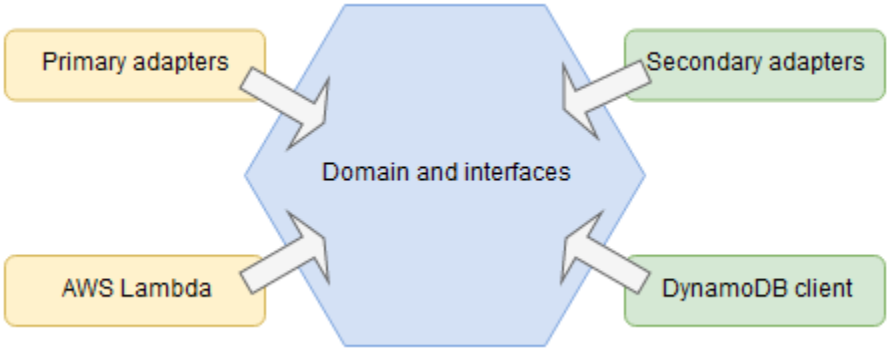
六角形のアーキテクチャを、構造化されたソフトウェアプロジェクトのモデル化で最も一般的な選択肢である従来の階層型アーキテクチャと比較してみましょう。この 2 つのアプローチには微妙ですが強力な違いがあります。

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、さまざまな懸念事項があります。このアーキテクチャでは依存関係階層を使用します。この階層では、最上位レイヤーはその下のレイヤーに依存しますが、その逆はありません。次の図では、プレゼンテーションレイヤーがユーザー操作を行うため、ユーザーインターフェイス、API、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ています。プレゼンテーション層は、ドメインロジックを実装するビジネス層に依存しています。一方、ビジネス層はデータアクセス層と複数の外部サービスに依存しています。



この構成の主な欠点は、依存構造です。たとえば、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が及びます。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。そのため、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が発生する可能性が高くなります。

次の図に示すように、六角アーキテクチャの依存関係はさまざまな方法で定義されます。意思決定は、すべてのインターフェースを定義するドメインビジネスロジックに集中します。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相互作用します。ポートは、ドメインと外部世界との相互作用を定義する抽象概念です。各インフラストラクチャコンポーネントに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を変更してもコアドメインロジックには影響しません。



周囲のコンポーネントはアダプターと呼ばれます。アダプターは外部世界と内部世界との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、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プライマリー・アダプタの良い例です。LambdaAWS 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サービスと統合されています。セカンダリアダプターは、外部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の良い例としては、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のアーキテクチャは、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
- ソフトウェア品質の向上
- 変化へのより容易な適応

これらのプロセスについて詳しくは、以下のセクションで説明します。

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場合、ランタイム環境を開発マシン上でローカルに複製することは非常に難しいため、ソフトウェアエンジニアにとって新たな課題が生じます。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展開が含まれている場合、長いフィードバックサイクルを伴います。このフィードバックサイクルを改善することで、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

Amazon API Gateway 内のゲートウェイ、AWS Lambda関数、Amazon DynamoDB テーブル、AWS Identity and Access Management (IAM) 権限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は、クラウドで直接テストすることです。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かもしれません。AWS一部のサービス (DynamoDB など) はローカルにデプロイできますが、そのほとんどはローカル設定ではレプリケートできません。同時に、MotoAWS などのサードパーティツールやテスト目的のモックサービスでは、実際のサービス API LocalStackコントラクトが正確に反映されていないか、機能の数が限ら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ほど頻繁に変更されないため、新しいビジネス要件に対応する必要があります。そのため、クラウドでのビジネスロジックのテストは、コードを変更し、デプロイを開始し、環境が整うのを待って、変更を検証するという大変な作業になります。デプロイにかかる時間が 5 分程度であれば、ビジネスロジックを 10 回変更してテストするには 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の完了を待つだけで何日もかかる場合があります。チームに複数の機能やエンジニアがいる場合、その延長期間はすぐにビジネスに気付かされます。

ローカルテスト中

六角形のアーキテクチャにより、開発者はインフラストラクチャの技術的な問題に煩わされることなく、その分野に集中できます。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内のユニットテストツール) を使用してドメインロジックの要件に対応します。ビジネスロジックをテストするために、技術的な統合の問題の解決に時間をかけたり、ソフトウェアをクラウドにデプロイしたりする必要はありません。ユニットテストをローカルで実行して、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 秒で完了すれば、ミスの検出にかかる時間を大幅に短縮できます。このアプローチについては、最初にビジネスロジックをテストするこのガイドの後のセクションで詳しく説明します。

開発の並列化

六角形の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行して行うことができます。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、各コンポーネントを分離し、各コンポーネント間に定義されたインターフェースによって可能になります。

製品の市場投入までの時間

ローカルユニットテストは、特に前述のように複雑なビジネスロジックを含む場合に、開発フィードバックサイクルを改善し、新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るコードカバレッジを増やすことで、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減少します。また、ユニットテストの対象範囲により、コードベースを継続的にリファクタリングして整理しておくことができるため、新人エンジニアのオンボーディングプロセスがスピードアップします。これについては、品質バイデザイン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十分に分離され、テストされていれば、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。これについては、変化への適応セクションで詳しく説明します。

品質バイデザイン

六角形の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

ヘキサゴナル・アーキテクチャー・アプローチを使用すると、開発者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、あるクラスまたは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まとまりが促進されます。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ースを使用することで、コードの可読性を高めることができます。問題やコーナーケースの特定が容易になります。

このアプローチにより、開発中のコードレビューが容易になり、気付かない変更や技術的負債の発生も抑えられます。

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、プロジェクトに導入 end-to-end、統合、ユニットテストを行うことで実現できます。End-to-end テストは、受信リクエストのライフサイクル全体をカバーします。通常、アプリケーションのエントリポイントを呼び出し、それが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て期待どおりの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し、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て結果をアサートするように構成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。テストシナリオの環境をセットアップして結果を確認するには、開発者が多くの時間を費やすことがあります。

ヘキサゴナル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のテストを行います。ビジネスロジックテストでは、モック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単体テストを組み合わせて、低い結合率で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ことをお勧めします。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、テスト駆動開発 (TDD) を使用して、開発の初期段階から適切なテストを行って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。最初にテストを作成しておく

と、ドメインに必要なインターフェースの模擬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。現時点では、ドメインロジックの実装は理想的ではないかもしれません。その後、デザインパターンを導入したり、コード全体を再配置したりすることで、既存のアーキテクチャをリファクタリングして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、プロジェクトの成長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インテグレーションプロセスで実行する自動テストと組み合わせることで、潜在的なバグの数を本番環境に移行する前に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は、 end-to-end 手動で統合とテストを行うために、AWSアカウントに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、新しい変更がリポジトリにプッシュされるたびにテストを自動化することをお勧めします。

メンテナンス性

保守性とは、アプリケーションを運用および監視して、すべての要件を満たしていることを確認し、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、future トラフィックや運用要件に合わせてシステムを調整する必要があります。また、クライアントへの影響を最小限またはまったく伴わずに、可用性が高く、簡単に導入できるようにする必要があります。

システムの現在および過去の状態を理解するには、システムを監視できるようにする必要があります。これは、オペレーターがシステムが期待どおりに動作することを確認し、バグを追跡するために使用できる特定のメトリクス、ログ、およびトレースを提供することで実現できます。また、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む必要なく、根本原因の分析を行うことができるはずです。

六角形のアーキテクチャは、Web アプリケーションの保守性を高め、コード全体の作業量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の実装から切り離すことで、オペレーターがシステムを深く理解し、プライマリアダプターまたはセカンダ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ックとログを作成できます。

変化への適応

ソフトウェアシステムは複雑になりがちです。その理由の 1 つは、ビジネス要件が頻繁に変更され、それに応じてソフトウェアアーキテクチャを調整する時間がほとんどないことが考えられます。また、頻繁な変更に対応するために、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するための投資が不十分である可能性もあります。理由が何であれ、ソフトウェアシステムは複雑になり、変更を加えることがほとんど不可能になる可能性があります。したがって、プロジェクトの最初から保守可能なソフトウェアアーキテクチャを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、変更に容易に適応できます。

このセクションでは、非機能要件やビジネス要件に容易に適応できる六角形のアーキテクチャを使用して、保守が容易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。

ポートとアダプタの使用による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求められる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行います。たとえば、データベースリポジトリ用に 1 つのアダプタがあり、サードパーティの API とやり取りするためのアダプタがもう 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ないため、あるアダプターを別のアダプターに簡単に置き換えることができます。たとえば、アプリケーションが SQL データベースから NoSQL データベースに切り替わる場合があります。この場合、ドメインモデルで定義されているポートを実装するための新しいアダプターを開発する必要があります。ドメインはデータベースリポジトリに依存せず、抽象化を使用して相互作用するため、ドメインモデルを変更する必要はありません。そのため、六角形アーキテクチャは非機能要件にも容易に適応できます。

コマンドとコマンドハンドラーの使用による新しいビジネス要件への適応

従来の階層型アーキテクチャでは、ドメインはパーシスタンス層に依存します。ドメインを変更する場合は、パーシスタンスレイヤーも変更する必要があります。それに比べて、六角形のアーキテクチャでは、ドメインはソフトウェア内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中核であり、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依存します。ドメインは依存性反転の原則を使用して、ポートを介して外部と通信します。依存関係の逆転の

利点は、コードの他の部分を壊すことを恐れずにドメインモデルを自由に変更できることです。ドメインモデルには解決しようとしているビジネス上の問題が反映されるため、変化するビジネス要件に合わせてドメインモデルを更新しても問題はありません。

ソフトウェアを開発する場合、考慮事項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、少し変更したコマンドパターンを使用できます。これは、操作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。その後、これらの操作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、コマンドを受け取り、ドメインの状態を変更し、呼び出し元に応答を返すメソッドです。同期 API や非同期キューなどの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。ドメインのすべての操作には、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えば、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して新しい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への適応が容易になります。

サービスファサードまたは CQRS パターンによるコンポーネントの分離

六角形アーキテクチャでは、プライマリアダプタはクライアントから受信する読み取り要求と書き込み要求をドメインに緩やかに結合する役割を果たします。この緩い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方法と、コマンドクエリ責任分離（CQRS）パターンを使用する方法の 2 つがあります。

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サービスを提供するためのフロントフェーシング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操作を提供します。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応答をクライアントにマッピングします。サービス・ファサードの利用は、単一の責任で複数の操作を行うマイクロサービスにとっては簡単です。ただし、サービスファサードを使用する場合、単一責任やオープンクローズ原則に従うことは困難です。単一責任の原則では、各モジュールはソフトウェアの単一機能に対してのみ責任を負うべきであるとされています。オープンクローズの原則では、コードは拡張する場合はオープンにし、変更するにはクローズする必要があります。サービスファサードが拡大するにつれて、すべての操作が1つのインターフェイスにまとめられ、依存関係がカプセル化され、より多くの開発者が同じファサードの変更を開始するようになります。そのため、開発中にサービスがそれほど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すすめします。

プライマリー・アダプターをヘキサゴナル・アーキテクチャで実装するもう1つの方法は、クエリーとコマンドを使用して読み取り操作と書き込み操作を分離するCQRSパターンを使用することです。

前に説明したように、コマンドはドメインの状態を変更するの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で実装されます。コマンドは非同期的に処理されるイベントとして実装できるのに対し、クエリは API を使用して同期的に実行できるため、これはイベント駆動型アーキテクチャを採用するプロジェクトにとって特に有利です。クエリには、そのクエリに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。長期的に規模拡大と維持を計画しているプロジェクトには、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、特に大規模プロジェクトにおいて、単一責任の原則を適用し、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は長期的には大きなメリットがありますが、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/書き込み操作を分けることなく、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築できます。これにより、後でそのアプローチを採用することにした場合に、CQRS用にプロジェクトを簡単にリファクタリングできます。

組織スケーリング

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 を組み合わせることで、組織は製品を迅速に拡張できます。コンウェイの法則によれば、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して進化する傾向があります。大規模な組織では、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専門知識に基づいてチームを構成することが多いため、この観察にはこれまで否定的な意味合いがありました。このアプローチの問題点は、製品や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分野横断的な懸念が伴い、チーム間の絶え間ないコミュニケーションが必要となることです。技術的な特徴に基づいてチームを編成すると、組織内に不必要なサイロ化が生じ、コミュニケーションが不十分になり、オーナーシップが欠如し、全体像が見えなくなります。最終的に、これらの組織上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、インバースコンウェイマヌーバは、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。たとえば、部門横断型のチームには、DDD とイベントストーミングを使用して特定される、特定のコンテキストセットに対する責任が割り当てられます。これらの境界のあるコンテキストは、製品の特定の機能を反映している場合があります。たとえば、アカウントチームが支払い状況を担当する場合があります。各新機能は、まとまりが強く、役割分担が緩い新しいチームに割り当てられるため、新機能はその機能の提供のみに集中でき、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、作成す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、各分野の専門家とソフトウェアの専門家が協力してビジネスドメインの複雑さを調査します。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および開発プロセスを開始します。

テストを最初から作成して実行する

テスト駆動開発 (TDD) を使用して、開発中のソフトウェアの正確性を検証します。TDD はユニットテストレベルで最もよく機能します。開発者は、まずテストを作成し、そのコンポーネントを呼び出すことでソフトウェアコンポーネントを設計します。このコンポーネントは最初は実装されていないため、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外部依存関係またはポートの動作をシミュレートし、コンポーネントの機能を実装します。テストが成功すると、開発者は実際のアダプターを実装して続行できます。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しているため、ソフトウェアの品質が向上し、コードが読みやすくなります。ヘキサゴナルアーキテクチャは、アプリケーションコアを分離することでTDD方法論をサポートします。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作成します。テストを実行するために複雑なアダプターを作成する必要はありません。代わりに、単純なモックオブジェクトとフィクスチャを使用できます。

行動主導型開発 (BDD) を使用して、 end-to-end 機能レベルで受け入れられるようにします。BDD では、開発者が機能のシナリオを定義し、ビジネス関係者と検証します。BDDテストでは、これを達成するためにできるだけ多くの自然言語を使用します。六角形アーキテクチャは、一次アダプタと二次アダプタという概念でBDD方法論をサポートしています。開発者は、外部サービスを呼び出さずにローカルで実行できるプライマリー・アダプターとセカンダリー・アダプターを作成できます。ローカルの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 テストスイートを構成します。

継続的インテグレーションパイプラインの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

ドメインをエンティティ、値オブジェクト、集約に分解し (ドメイン駆動型設計の実装について読んでください)、それらの動作を定義します。プロジェクトの最初に作成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。ドメインオブジェクトが動作を完了した後に発生するイベントを定義します。

アダプターがドメインとのやり取りに使用できるインターフェースを定義します。

テストとデプロイの自動化

最初に概念実証を行った後は、 DevOps時間をかけて実践することをおすすめします。たとえば、継続的インテグレーションと継続的デリバリー (CI/CD) パイプライン、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、コードをマージする前にテストします。
- 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、 end-to-endまたは自動統合と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リングする

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拡張してください。六角形アーキテクチャが提供する移植性を利用してパフォーマンスを向上させてください。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 API に分割します。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の採用を検討してください。

新機能のリクエストが多い場合は、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。組織スケーリング前のセクションで説明したように、チームが境界コンテキストとして 1 つ以上の機能を所有するようにチームを構成します。その後、これらのチームは六角形の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。

六角形のアーキテクチャコンセプトに即したプロジェクト構造を設計

Infrastructure as code (IaC) は、クラウド開発で広く採用されている手法です。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および管理できます。この方法では、バージョン管理システムを使用してアーキテクチャの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および移動できます。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存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保守できます。

アプリケーションを六角形アーキテクチャの概念に対応する 3 つのフォルダーまたはプロジェクトに分割することをお勧めします。つまり、`entrypoints` (プライマリアダプター)、`domain` (ドメインとインターフェイス)、`adapters` (セカンダリアプター) です。

以下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、以前に推奨したように、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
```

前に説明したように、ドメインはアプリケーションの中核であり、他のモジュールには依存しません。`domain`以下のサブフォルダーを含むようにフォルダーを構成することをお勧めします。

- `command handlers`ドメイン上でコマンドを実行するメソッドまたはクラスが含まれます。

- commandsには、ドメイン上で操作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ます。
- eventsには、ドメインを介して送信され、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ている既知のエラーが含まれます。
- modelには、ドメインエンティティ、値オブジェクト、およびドメインサービスが含まれます。
- portsドメインがデータベース、API、またはその他の外部コンポーネントと通信する際の抽象化が含まれます。
- testsには、ドメインで実行されるテストメソッド (ビジネスロジックテストなど) が含まれます。

プライマリアダプタは、entrypointsフォルダで表されるアプリケーションのエントリポイントです。この例では、apiフォルダーをプライマリアダプターとして使用します。このフォルダーには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このフォルダには 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、調和して機能することを検証する浅いテストです。

adaptersフォルダーで表されるセカンダリアダプターは、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好例です。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ている実装を使用して新しいアダプターを作成できます。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サブフォルダーには、各アダプターの外部統合テストが含まれています。

のインフラストラクチャの例AWS

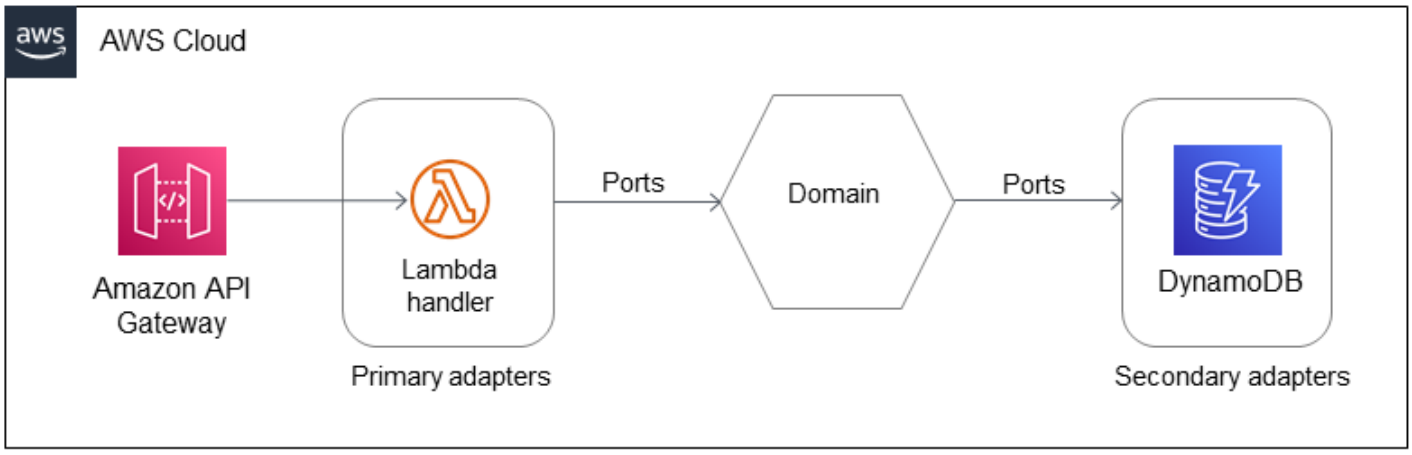
このセクションでは、AWSヘキサゴナルアーキテクチャの実装に使用できるアプリケーションのインフラストラクチャを設計する例を示します。最小限の実行可能な製品 (MVP) を構築するためのシンプルなアーキテクチャから始めることをお勧めします。ほとんどのマイクロサービスには、クライアント要求を処理するための単一のエントリポイント、コードを実行するコンピューティングレイヤー、およびデータを保存するためのパーシスタンスレイヤーが必要です。AWS以下のサービスは、六角形アーキテクチャのクライアント、プライマリアダプタ、セカンダリアダプタとしての使用に適しています。

- クライアント:Amazon API Gateway、Amazon Simple Queue Service (Amazon SQS)、Elastic Load Balancing、Amazon EventBridge
- プライマリーアダプタ:AWS Lambda、、
- セカンダリアダプタ:Amazon DynamoDB、、、 EventBridge、、、、、、、、、、、、、、、、、、Amazon S

次のセクションでは、ヘキサゴナルアーキテクチャを詳しく見て、六角アーキテクチャを詳しく見て、ヘキサゴナルアーキテクチャを詳しく見て、六角アーキテクチャを詳しく見て

シンプルに始める

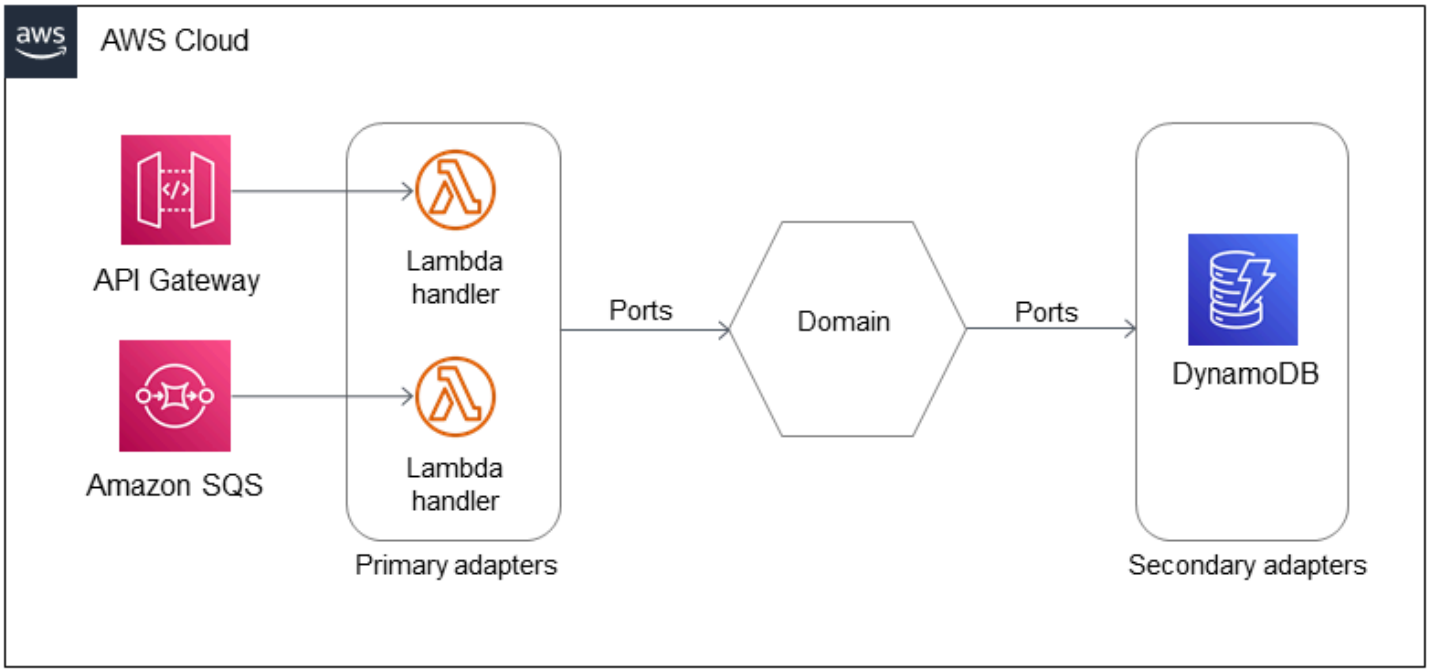
ヘキサゴナルアーキテクチャを使用してアプリケーションを設計する場合は、シンプルなものから始めることをお勧めします。この例では、API Gateway がクライアント (REST API) として使用され、Lambda がプライマリアプター (コンピューティング) として使用され、DynamoDB がセカンダリアプター (パーシスタンス) として使用されます。ゲートウェイクライアントはエントリーポイント（この場合は Lambda ハンドラー）を呼び出します。



このアーキテクチャは完全にサーバーレスであり、アーキテクトは出発点として適しています。コードの管理が容易になり、新しいビジネス要件や非機能要件に適応できるため、ドメイン内のコマンドパターンを使用することをお勧めします。このアーキテクチャは、いくつかの操作だけで簡単なマイクロサービスを構築するには十分かもしれません。

CQRS パターンを適用する

ドメインの操作数が増える場合は、CQRS パターンに切り替えることをお勧めします。次の例を使用すると、CQRS パターンを完全なサーバーレスアーキテクチャとして適用できます。AWS

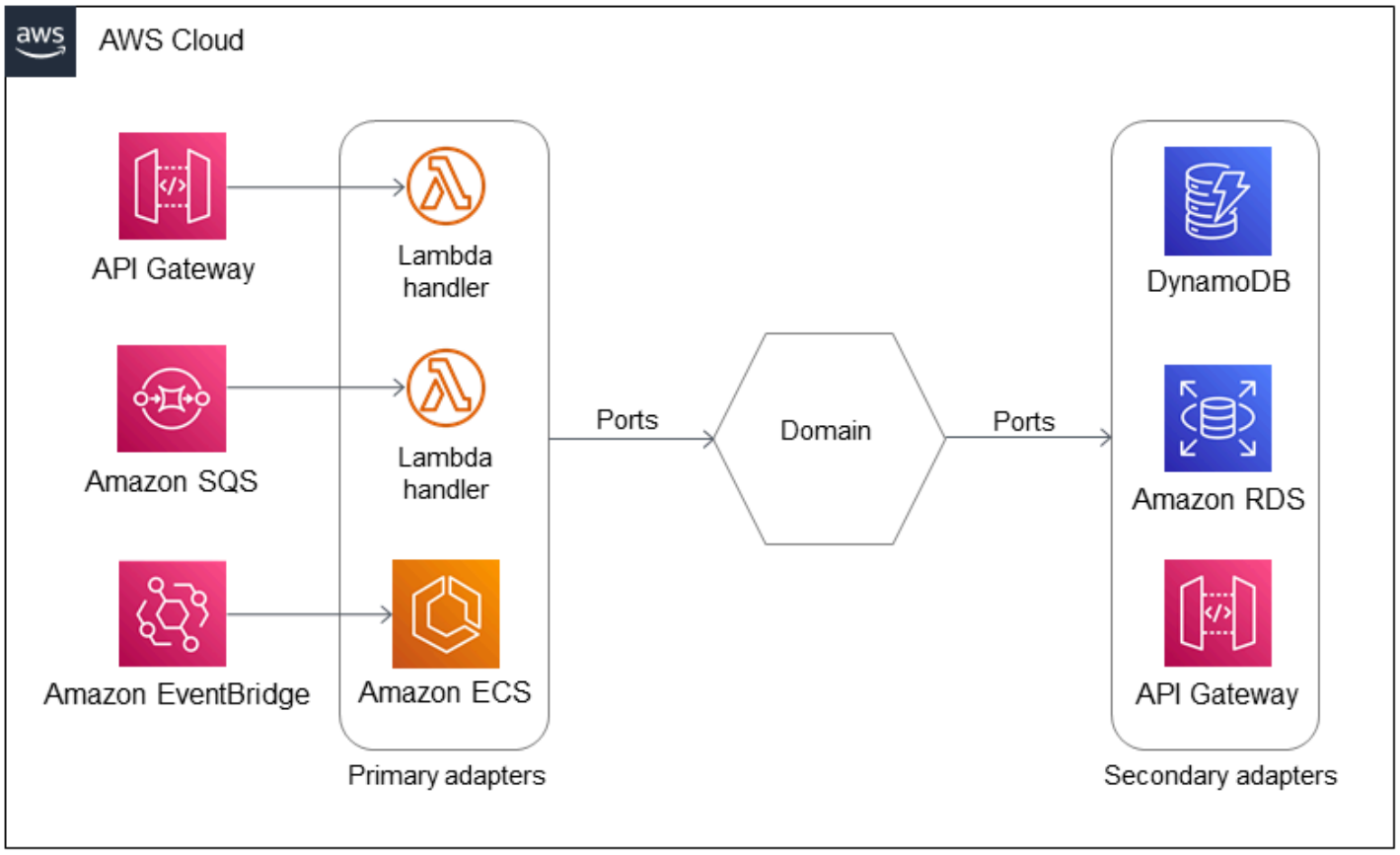


この例では、2 つの Lambda ハンドラーを使用します。1 つはクエリ用、もう 1 つはコマンド用です。クエリは、API ゲートウェイをクライアントとして使用して同期的に実行されます。コマンドは Amazon SQS をクライアントとして使用して非同期に実行されます。

このアーキテクチャには、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クライアント (API Gateway と Amazon SQS) と複数のプライマリアダプタ (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しているため、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ましょう

コンテナは、実行時間の長いタスクに適しています。データスキーマがあらかじめ定義されていて、SQL 言語の利点を活用したい場合は、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 API と通信する必要があります。アーキテクチャのサンプルを拡張して、次の図に示されているように、アーキテクチャの要件を詳しく見て、アーキテクチャを詳しく見て、アーキテクチャを詳しく見て、アーキテクチャを詳しく見て

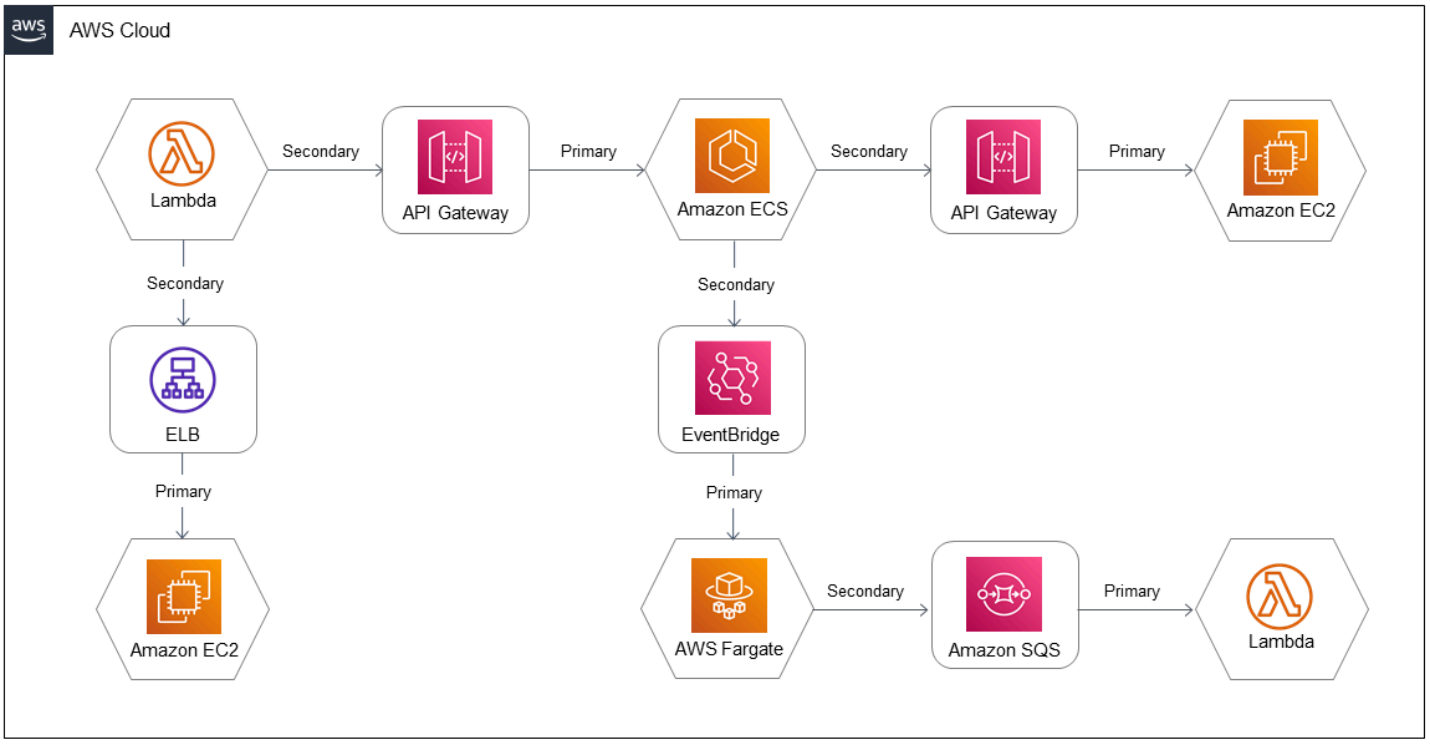


この例では、ドメイン内で長時間実行されるタスクを起動するためのプライマリアダプターとして Amazon ECS を使用しています。Amazon EventBridge (クライアント) は、特定のイベントが発生すると Amazon ECS タスク (エントリポイント) を開始します。このアーキテクチャには、リレーショナルデータを保存するためのもう 1 つのセカンダリアダプタとして Amazon RDS が含まれています。また、外部 API 呼び出しを呼び出すためのセカンダリアダプターとして、別の API ゲートウェイも追加されます。そのため、このアーキテクチャでは、1 つのビジネスドメイン内の基盤となるさまざまな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ー・アダプターとセカンダリー・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によってすべてのプライマリー・アダプターおよびセカンダリー・アダプターと緩やかに結合されます。ドメインは、ポートを使用して外部に何を要求するかを定義します。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作成することで 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 (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合致しています。これまで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とセカンダリアダプタ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、他のドメインとゆるく結合されています。



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セットを使用します。(前の例のように、各ドメインには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、プライマリー・アダプターとセカンダリー・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てもドメインは影響を受けません。さらに、ドメインは互いに切り離されています。

前の図に示した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として使用されています。API Gateway、Elastic Load Balancing EventBridge、および Amazon SQS がセカンダリアダプタとして使用されます。

よくある質問

六角形のアーキテクチャを使用する理由は何ですか?

六角形のアーキテクチャは、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。

ドメイン駆動設計を使用する理由は何ですか?

ドメイン駆動型設計 (DDD) では、ビジネス関係者とエンジニアの間で共通の言語を使用して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を管理するのに役立ち、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。

ヘキサゴナルアーキテクチャなしでテスト駆動開発を実践することはできますか?

はい。テスト駆動開発 (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、六角形のアーキテクチャを使用すると、TDDの練習が容易になります。

六角形アーキテクチャやドメイン駆動型設計なしで製品をスケーリングできますか?

はい。技術的および組織的な製品スケーリングは、ほとんどのデザインパターンで実現できます。ただし、六角形のアーキテクチャと DDD の方がスケーリングが容易で、長期的には大規模プロジェクトの方が効果的です。

ヘキサゴナルアーキテクチャの実装にはどのテクノロジーを使用すればよいですか?

六角形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。依存関係の逆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。

実用最小限の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のに時間を費やすのは理にかなっていますか？

はい。MVP には使い慣れたデザインパターンを使用することをおすすめします。エンジニアが慣れるまで、六角形のアーキテクチャを試して練習することをお勧めします。新しいプロジェクトのために六角形のアーキテクチャを確立することは、アーキテクチャなしで始める場合と比べてそれほど大きな時間を費やす必要はありません。

実用最小限の製品を開発中ですが、テストを書く時間がありません。

MVPにビジネスロジックが含まれている場合は、そのための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減り、時間を節約できます。

六角形アーキテクチャでは他にどのようなデザインパターンを使用できますか?

システム全体のスケーリングをサポートするには、CQRS パターンを使用してください。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。作業単位パターンを使用してトランザクションプロセスのステップを管理します。ドメインアグリゲート、エンティティ、バリューオブジェクトをモデル化するには、継承よりも合成を使用します。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

- リソースこのセクションにまとめられたリンクを読んで、ドメイン主導型の設計コンセプトについてさらに理解を深めてください。
- 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てください。
- 既存のプロジェクトを実装している場合は、読み取り専用操作と書き込み専用操作に分けることができるコードを特定してください。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。
- 基本的なプロジェクト構造が整ったら、ユニットテストを記述し、テスト自動化による継続的インテグレーション (CI) を確立し、テスト駆動開発 (TDD) プラクティスに従います。

リソース

リファレンス:

- AWS Lambda(AWS規範的ガイダンスパターン) を使用して六角形アーキテクチャの Python プロジェクトを構築する
- アジャイルチーム (スケーリングアジャイルフレームワークウェブサイト)
- ハリー・パーシバルとボブ・グレゴリーによるPythonを使ったアーキテクチャパターン（オライリー・メディア、2020年3月31日）。具体的には以下の章をご覧ください。
  - コマンドとコマンドハンドラ
  - コマンドクエリ責任分離 (CQRS)
  - リポジトリパターン
- Alberto Brandolini 著「イベントストーミング：サイロの境界を越えたコラボレーションへの最もスマートなアプローチ」（Event Storming Web サイト）
- サム・ニューマン著「コンウェイの法則の謎解き」（Thoughtworksウェブサイト、2014年6月30日）
- ジェームズ・ベズウィック氏による進化的アーキテクチャの開発（AWSCompute ブログ、2021 年 7 月 8 日）AWS Lambda
- ドメイン言語:ソフトウェアの中心にある複雑さへの取り組み (ドメイン言語ウェブサイト)
- ファサード、アレクサンダー・シュヴェッツ著「ダイブ・イントゥ・デザイン・パターン」（電子書籍、2018年12月5日）
- GivenWhenThen、マーティン・ファウラー著（2013年8月21日）
- ドメイン主導型デザインの実装、ヴォーン・バーノン著（アディソン・ウェスリー・プロフェッショナル、2013年2月）
- 逆コンウェイ法（ソートワークスウェブサイト、2014年7月8日）
- 今月のパターン:レッドグリーンリファクタリング (dZone ウェブサイト、2017 年 6 月 2 日)
- 確かな設計原則の解説：依存関係逆転の原理とコード例、Thorben Janssen（Stackify ウェブサイト、2018 年 5 月 7 日）
- 「確固たる原則：説明と事例」サイモン・ホイバーグ（ITNEXTウェブサイト、2019年1月1日）
- ジェームズ・ショアとシェーン・ウォーデン著「アジャイル開発の芸術：テスト主導型開発」（オライリー・メディア、2010年3月25日）
- 平易な英語で説明されているオブジェクト指向プログラミングの確かな原則、Yigit Kemal Erinc著（freeCodeCampオブジェクト指向プログラミングの投稿、2020年8月20日）

- イベント駆動型のアーキテクチャとは (AWSウェブサイト)

AWS のサービス

- 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)
- Elastic Load Balancing
- 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

- モト
- LocalStack

ドキュメント履歴

このガイドは、このドキュメントの大きな変更点をまとめたものです。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
初版出版	—	2022 年 6 月 15 日

AWS 規範的ガイダンスの用語集

以下は、 AWS 規範的ガイダンスが提供する戦略、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します。
- 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: オンプレミスの Oracle データベースを の Oracle 用 Amazon Relational Database Service (Amazon RDS) に移行します AWS クラウド。
- 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: カスタマーリレーションシップ管理 (CRM) システムを Salesforce.com に移行します。
- 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: オンプレミスの Oracle データベースを の EC2 インスタンス上の Oracle に移行します AWS クラウド。
- 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。サーバーを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。

- 使用停止 — 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コントロール」を参照してください。

抽象化されたサービス

「 マネージドサービス」を参照してください。

ACID

「アトミック性、一貫性、分離性、耐久性」を参照してください。

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ですが、より多くの作業が必要です。

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の方法。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

行のグループを操作し、グループの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 などがありますMAX。

AI

「人工知能」を参照してください。

AIOps

「人工知能オペレーション」を参照してください。

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。

アプリケーションコントロール

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 の需要要素であり、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?」を参照してください。

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

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)

組織がクラウドに正常に移行 AWS するための効率的で効果的な計画を立てるのに役立つ、 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。 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編成します。ビジネス、人材、ガバナンスの観点では、ビジネススキルとプロセスに重点を置き、プラットフォーム、セキュリティ、オペレーションの視点は技術的なスキルとプロセスに焦点を当てています。例えば、人材の観点では、人事 (HR)、人材派遣機能、および人材管理を扱うステークホルダーを対象としています。この観点から、 AWS CAF は、クラウド導入を成功させるための組織の準備に役立つ人材開発、トレーニング、コミュニケーションに関するガイダンスを提供します。詳細については、AWS CAF ウェブサイト と AWS CAF のホワイトペーパー を参照してください。

AWS ワークロード認定フレームワーク (AWS WQF)

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

B

不正なボット

個人または組織に混乱や損害を与えることを目的としたボット。

BCP

事業継続計画を参照してください。

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Data in a behavior graph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。エンディアンネス も参照してください。

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

2 つの異なる同一の環境を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は 1 つの環境 (青) で実行し、新しいアプリケーションバージョンは他の環境 (緑) で実行します。この戦略は、影響を最小限に抑えながら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インタラクション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、個人や組織に混乱を与えたり、損害を与えたりすることを意図しているものがあります。

ボットネット

マルウェアに感染し、ボットのヘルダーまたはボットオペレーター と呼ばれる、単一関係者の管理下にあるボットのネットワーク。ボットは、ボットとその影響をスケールするための最もよく知られているメカニズムです。

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、「ブランチについて (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

例外的な状況や承認されたプロセスを通じて、ユーザーが通常アクセス許可を持たない AWS アカウント 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、 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。 AWS

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、ホワイトペーパー AWSでのコンテナ化されたマイクロサービスの実行 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。

Canary デプロイ

エンドユーザーへのバージョンの低速かつ増分的なリリース。確信できたら、新しいバージョンをデプロイし、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

「データキャプチャの変更」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します。AWS Fault Injection Service （ AWS FIS ） を使用して、 AWS ワークロードに負荷をかけてレスポンス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 の投稿」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テクノロジー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 に移行するときに通常実行する 4 つのフェーズ AWS クラウド :

- 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。移行戦略とどのように関連しているかについては、 AWS 「移行準備ガイド」を参照してください。

CMDB

「設定管理データベース」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 または が含まれます AWS CodeCommit。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などのビジュアル形式から情報を分析および抽出する AI の分野。例えば、 はオンプレミスのカメラネットワークに CV を追加するデバイス AWS Panorama を提供し、Amazon SageMaker は CV の画像処理アルゴリズムを提供します。

設定ドリフト

ワークロードの場合、設定は想定した状態から変化します。これにより、ワークロードが非準拠になる可能性があり、通常は段階的かつ意図的ではありません。

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 リージョンで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

一元化された管理とガバナンスにより、分散型の分散型データ所有権を提供す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していることを確認できます。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通常、大量の履歴データが含まれており、クエリや分析によく使用されます。

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る。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。

ディープラーニング

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。

defense-in-depth

一連のセキュリティメカニズムとコントロールをコンピュータネットワーク全体に層状に重ねて、ネットワークとその内部にあるデータの機密性、整合性、可用性を保護する情報セキュリティの手法。この戦略を に採用するときは AWS、 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。例えば、 defense-in-depth 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。

デプロイメント

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、Implementing security controls on AWSのDetective controls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 では、ファクトテーブル内の量的データに関するデータ属性を含む小さなテーブル。ディメンションテーブル属性は通常、テキストフィールドまたはテキストのように動作する離散数値です。これらの属性は、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一般的に使用されます。

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)

災害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に使用する戦略とプロセス。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の追跡。例えば、 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

「開発値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。

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、応答時間を短縮できます。

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プロセス。

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 (アカウンティング、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 の中央テーブル。事業運営に関する定量的データを保存します。通常、ファクトテーブルには、メジャーを含む列とディメンションテーブルへの外部キーを含む列の 2 種類の列が含まれます。

フェイルファスト

頻繁で段階的なテストを使用して開発ライフサイクルを短縮する哲学。これはアジャイルアプローチの重要な部分です。

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向上させる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です。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。

FGAC

「きめ細かなアクセスコントロール」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

段階的なアプローチを使用するのではなく、変更データキャプチャによる継続的なデータレプリケーションを使用して、可能な限り短い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。

G

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)

Amazon では CloudFront、特定の国のユーザーがコンテンツディストリビューションにアクセスできないようにするオプションです。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 ワークフローはレガシーと見なされ、トランクベースのワークフローはモダンで推奨されるアプローチです。

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、、Amazon AWS Security Hub、GuardDuty、Amazon Inspector AWS Trusted Advisor、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。

ハイアベイラビリティ (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。

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。緊急性のため、通常、修正は一般的な DevOps リリースワークフローの外で行われます。

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

「産業モノのインターネット」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

既存のインフラストラクチャを更新、パッチ適用、または変更するのではなく、本番ワークロード用の新しいインフラストラクチャをデプロイするモデル。イミュータブルインフラストラクチャは、本質的に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、予測性が高くなります。詳細については、 AWS Well-Architected フレームワークの「イミュータブルインフラストラクチャを使用したデプロイ」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

接続、リアルタイムデータ、自動化、分析、AI/ML の進歩を通じて、 のビジネスプロセスのモダナイズを指すために 2016 年に Klaus Schwab によって導入された用語。

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。

産業分野における 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、「Building an industrial Internet of Things (IIoT) digital transformation strategy」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、「AWS を使用した機械学習モデルの解釈」を参照してください。

IoT

「モノのインターネット」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。

大規模な移行

300 台以上のサーバの移行。

LBAC

「ラベルベースのアクセスコントロール」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 R」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。エンディアンネス も参照してください。

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またはプライバシーを侵害するように設計されているソフトウェア。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、不正アクセスにつながる

可能性があります。マルウェアの例としては、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 が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、ユーザーが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。Amazon Simple Storage Service (Amazon S3) と Amazon DynamoDB は、 マネージドサービスの例です。これら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、制御するためのソフトウェアシステム。これにより、加工品を現場の完成製品に変換します。

MAP

「移行促進プログラム」を参照してください。

メカニズム

ツールを作成し、ツールの導入を推進し、調整のために結果を検査する完全なプロセス。メカニズムとは、動作中にそれ自体を強化して改善するサイクルです。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

内の組織の一部である管理アカウント AWS アカウント を除くすべての 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。

メッセージキューイングテレメトリトランスポート (MQTT)

リソースに制約のある IoT デバイス用の、パブリッシュ/サブスクライブパターンに基づく軽量の machine-to-machine (M2M) 通信プロトコル。

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)

組織がクラウドへの移行のための強固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、オペレーション、ビジネスアナリストと所有者、移行エンジニア、デベロッパー、スプリントに取り組む 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には、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: Application Migration Service を使用して Amazon EC2 AWS への移行をリホストします。

Migration Portfolio Assessment (MPA)

に移行するためのビジネスケースを検証するための情報を提供するオンラインツール AWS クラウド。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 に移行するために使用されるアプローチ AWS クラウド。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、「」の「アプリケーションをモダナイズするための戦略 AWS クラウド」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、「」の「アプリケーションのモダナイゼーション準備状況の評価 AWS クラウド」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

「移行ポートフォリオ評価」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。

変更可能なインフラストラクチャ

本番ワークロードの既存のインフラストラクチャを更新および変更するモデル。Well-Architected AWS Framework では、一貫性、信頼性、予測可能性を向上させるために、イミュータブルな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

「オリジンアクセスコントロール」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

「運用レベルの契約」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。

オープンプロセス通信 - 統合アーキテクチャ (OPC-UA)

産業用オートメーション用の machine-to-machine (M2M) 通信プロトコル。OPC-UA は、データの暗号化、認証、認可スキームを備えた相互運用性標準を提供します。

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)

インシデントや潜在的な障害の理解、評価、防止、または範囲の縮小に役立つ質問とそれに関連するベストプラクティスのチェックリスト。詳細については、 AWS Well-Architected フレームワークの「運用準備状況レビュー (ORR)」を参照してください。

運用テクノロジー (OT)

産業運用、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するハードウェアおよびソフトウェアシステム。製造では、OT と情報技術 (IT) システムの統合が、Industry 4.0 トランスフォーメーションの主要な焦点です。

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

の組織 AWS アカウント 内のすべての のすべてのイベントを口グ AWS CloudTrail 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、 ドキュメントの「組織の証跡の作成」を参照してください。 CloudTrail

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から、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。

オリジンアクセスコントロール (OAC)

では CloudFront、Amazon Simple Storage Service (Amazon S3) コンテンツを保護するためのアクセスを制限するための拡張オプションです。OAC は、すべての のすべての S3 バケット AWS リージョン、 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)

では CloudFront、Amazon S3 コンテンツを保護するためのアクセスを制限するオプションです。OAI を使用すると、 は Amazon S3 が認証できるプリンシパル CloudFront を作成します。認証されたプリンシパルは、特定の CloudFront ディストリビューションを介してのみ S3 バケット内のコンテンツにアクセスできます。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コントロールが可能です。

ORR

「運用準備状況レビュー」を参照してください。

OT

「運用技術」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る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

「プログラム可能な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

アクセス許可の定義 (アイデンティティベースのポリシー を参照)、アクセス条件の指定 (リソースベースのポリシー を参照)、または の組織内のすべてのアカウントに対する最大アクセス許可の定義 AWS Organizations (サービスコントロールポリシー を参照) が可能なオブジェクト。

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。詳細については、マイクロサービスでのデータ永続性の有効化 を参照してください。

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、「移行準備状況ガイド」を参照してください。

述語

`true` または を返すクエリ条件。`false`通常は `WHERE`句にあります。

述語のプッシュダウン

転送前にクエリ内のデータをフィルタリングするデータベースクエリ最適化手法。これにより、リレーショナルデータベースから取得して処理する必要があるデータの量が減少し、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、Implementing security controls on AWSのPreventative controls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 ロール AWS アカウント、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内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

エンジニアリングプロセス全体を通してプライバシーを考慮に入れた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

非準拠のリソースのデプロイを防止する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、プロビジョニング前にリソースをスキャンします。リソースがコントロールに準拠していない場合、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「 での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)

設計、開発、発売から成長と成熟まで、製品のデータとプロセスのライフサイクル全体にわたる管理、および辞退と削除。

本番環境

「環境」を参照してください。

プログラミング可能ロジックコントローラー (NAL)

製造では、マシンをモニタリングし、承認プロセスを自動化する、信頼性が高く、適応性の高いコンピュータです。

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。

パブリッシュ/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。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 では、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できるチャネルにイベントメッセージを発行できます。システムは、公開サービスを変更せずに新しいマイクロサービスを追加できます。

Q

クエリプラン

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る手順などの一連のステップ。

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

責任、説明責任、相談、通知 (RACI) 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

責任、説明責任、相談、通知 (RACI) を参照してください。

RCAC

「行と列のアクセスコントロール」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。

再構築

「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 とは分離され、独立しています。詳細については、AWS リージョン 「 を使用できるアカウントを指定する」を参照してください。

回帰

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 R」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。

プラットフォーム変更

「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

中断に耐えたり、中断から回復したりするアプリケーションの機能。で障害耐性を計画する場合、高可用性とディザスタリカバリが一般的な考慮事項です AWS クラウド。詳細については、AWS クラウド 「レジリエンス」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。サポートを含めると、そのマトリックスは RASCI マトリックスと呼ばれ、サポートを除外すると 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、Implementing security controls on AWSのResponsive controls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 R」を参照してください。

ローテーション

攻撃者が認証情報にアクセスすることをより困難にするために、シークレットを定期的に更新するプロセス。

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 (IdPs) が使用するオープンスタンダード。この機能により、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 にログイン AWS

Management Console したり AWS 、 API オペレーションを呼び出したりできます。組織内のすべてのユーザーに対して IAM でユーザーを作成する必要はありません。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

「監視コントロールとデータ収集」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

では AWS Secrets Manager、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。シークレット値は、バイナリ、単一の文字列、または複数の文字列にすることができます。詳細については、Secrets Manager ドキュメントの「Secrets Manager シークレットの内容」を参照してください。

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、予防的 、検出的 、???応答的 、プロアクティブ の 4 つの主なタイプがあります。

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に自動的に応答または修正するように設計された、事前定義されプログラムされたアクション。これらのオートメーションは、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としては、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、認証情報のローテーションなどがあります。

サーバー側の暗号化

送信先にあるデータの、それを受け取る AWS のサービス 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、スループットなど、サービスのパフォーマンス側面の測定。

サービスレベルの目標 (SLO)

サービスレベルのインジケータ によって測定される、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

「セキュリティ情報とイベント管理システム」を参照してください。

単一障害点 (SPOF)

システムを中断させる可能性のあるアプリケーションの単一の重要なコンポーネントの障害。

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。

split-and-seed 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、「」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 AWS クラウド」を参照してください。

SPOF

単一障害点 を参照してください。

star スキーマ

トランザクションデータまたは測定データを保存するために 1 つの大きなファクトテーブルを使用し、データ属性を保存するために 1 つ以上の小さなディメンションテーブルを使用するデータベース組織構造。この構造は、データウェアハウスまたはビジネスインテリジェンスの目的で使用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。

監視統制とデータ収集 (SCADA)

製造では、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生産オペレーション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

ユーザーインタラクションをシミュレートして潜在的な問題を検出したり、パフォーマンスをモニタリングしたりする方法でシステムをテストします。Amazon CloudWatch Synthetics を使用してこれらのテストを作成できます。

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するサービスへの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「 AWS Organizations を他の AWS のサービスで使用する 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 つのピザを食べることができる小さな 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

現在のレコードに関連する行のグループに対して計算を実行する SQL 関数。ウィンドウ関数は、移動平均の計算や、現在の行の相対位置に基づく行の値へのアクセスなどの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

「書き込み 1 回」を参照し、多くの を読み取ります。

WQF

「AWS ワークロード認定フレームワーク」を参照してください。

Write Once, Read Many (WORM)

データを 1 回書き込み、データの削除や変更を防ぐストレージモデル。承認されたユーザーは、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な 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 を利用する攻撃、通常はマルウェア。

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。

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。