AWS ホワイトペーパー

でのマイクロサービスの実装 AWS

でのマイクロサービスの実装 AWS: AWS ホワイトペーパー

Table of Contents

でのマイクロサービスの実装 AWS

公開日: 2023 年 7 月 31 日 (ドキュメント履歴)

マイクロサービスは、デプロイの迅速化、イノベーションの促進、保守性の向上、スケーラビリティの向上を実現する、ソフトウェア開発への効率的なアプローチを提供します。この方法は、自律型チームによって管理される明確に定義された APIs を介して通信する、疎結合された小さなサービスに依存しています。マイクロサービスの導入には、スケーラビリティ、耐障害性、柔軟性の向上、開発サイクルの短縮などのメリットがあります。

このホワイトペーパーでは、API 駆動型、イベント駆動型、データストリーミングの 3 つの一般的なマイクロサービスパターンについて説明します。各アプローチの概要、マイクロサービスの主な機能の概要、開発における課題への対処、アプリケーションチームがこれらの障害にどのように対処できるかを説明します (AWS) 。

データストア、非同期通信、サービス検出などのトピックの複雑な性質を考慮すると、アーキテクチャ上の意思決定を行う際に提供されるガイダンスとともに、アプリケーションの特定のニーズとユースケースを比較検討することをお勧めします。

序章

マイクロサービスアーキテクチャは、次のようなさまざまな分野の成功した概念と実証済みの概念を組み合わせています。

- アジャイルソフトウェア開発
- サービス指向アーキテクチャ
- API ファースト設計
- 継続的インテグレーション/継続的デリバリー (CI/CD)

多くの場合、マイクロサービスには 12 要素アプリの設計パターンが組み込まれています。

マイクロサービスには多くの利点がありますが、ユースケース固有の要件と関連するコストを評価することが重要です。モノリシックアーキテクチャや代替アプローチが適している場合もあります。マイクロサービスまたはモノリスは、規模、複雑さ、特定のユースケースなどの要因を考慮して、case-by-caseケースで決定する必要があります。

まず、スケーラビリティと耐障害性に優れたマイクロサービスアーキテクチャ (ユーザーインターフェイス、マイクロサービス実装、データストア) について調べ、コンテナテクノロジー AWS を使

用して で構築する方法を示します。次に、一般的なサーバーレスマイクロサービスアーキテクチャを実装し、運用の複雑さを軽減するための AWS サービスを提案します。

サーバーレスは、次の原則によって特徴付けられます。

- プロビジョニングまたは管理するインフラストラクチャがない
- 消費単位による自動スケーリング
- 「価値の支払い」請求モデル
- 組み込みの可用性と耐障害性
- イベント駆動型アーキテクチャ (EDA)

最後に、システム全体を調べ、分散モニタリング、ログ記録、トレース、監査、データ整合性、非同期通信など、マイクロサービスアーキテクチャのクロスサービスの側面について説明します。

このドキュメントでは AWS クラウド、ハイブリッドシナリオと移行戦略を除く、 で実行されているワークロードに焦点を当てています。移行戦略の詳細については、「コンテナ移行方法」ホワイトペーパーを参照してください。

Well-Architected の実現状況の確認

AWS Well-Architected フレームワークは、クラウド内でのシステム構築に伴う意思決定の長所と短所を理解するのに役立ちます。このフレームワークの 6 つの柱により、信頼性、安全性、効率、費用対効果、持続可能性の高いシステムを設計および運用するための、アーキテクチャのベストプラクティスを確認できます。AWS マネジメントコンソール で無料で提供されている AWS Well-Architected Tool を使用すると、柱ごとに一連の質問に答えることで、これらのベストプラクティスに照らしてワークロードを評価できます。

サーバーレスアプリケーションレンズでは、サーバーレスアプリケーションを設計するためのベストプラクティスに焦点を当てています AWS。

クラウドアーキテクチャに関する専門的なガイダンスやベストプラクティス (リファレンスアーキテクチャのデプロイ、図、ホワイトペーパー) については、AWS アーキテクチャセンターを参照してください。

マイクロサービスへのモダナイズ

マイクロサービスは基本的に、アプリケーションを構成する小さな独立したユニットです。従来のモノリシック構造からマイクロサービスへの移行は、さまざまな戦略に従うことができます。

この移行は、組織の運営方法にも影響します。

- これにより、チームがクイックサイクルで作業するアジャイル開発が促進されます。
- チームは通常小規模で、2 つのピザチームとも呼ばれます。2 つのピザがチーム全体に餌を与えるのに十分な大きさです。
- チームは、作成からデプロイ、メンテナンスまで、サービスに対して全責任を負います。

のシンプルなマイクロサービスアーキテクチャ AWS

一般的なモノリシックアプリケーションは、プレゼンテーションレイヤー、アプリケーションレイヤー、データレイヤーのさまざまなレイヤーで構成されています。一方、マイクロサービスアーキテクチャは、技術レイヤーではなく、特定のドメインに従って機能をまとまりのある垂直方向に分離します。図 1 は、一般的なマイクロサービスアプリケーションのリファレンスアーキテクチャを示しています AWS。

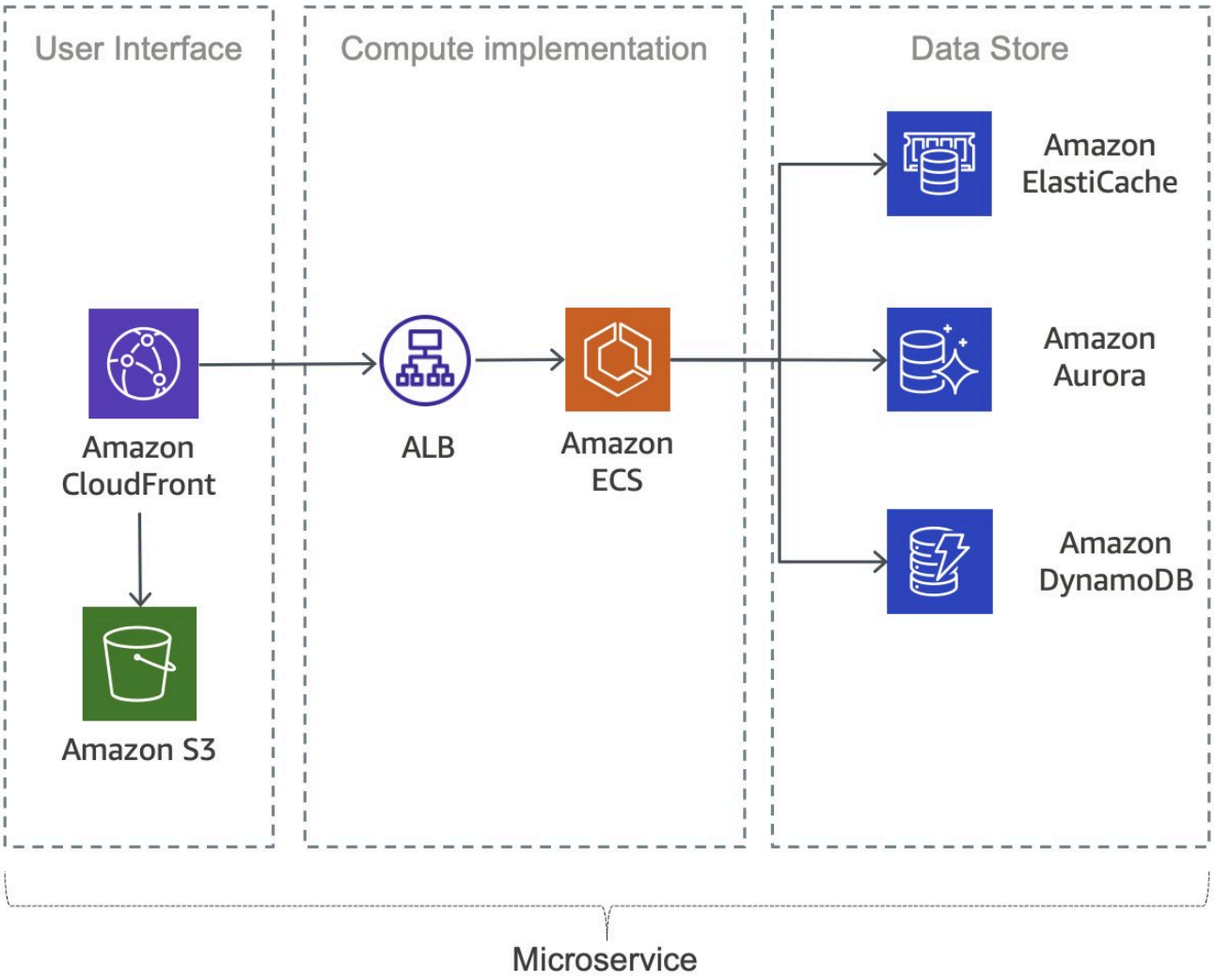

図 1: の一般的なマイクロサービスアプリケーション AWS

[ユーザーインターフェイス]

最新のウェブアプリケーションでは、多くの場合、JavaScript フレームワークを使用して、バックエンド APIs。これらの APIsは通常、表現状態転送 (REST) または RESTful APIs、または GraphQL APIs。静的ウェブコンテンツは、Amazon Simple Storage Service (Amazon S3) と Amazon CloudFront を使用して提供できます。 Amazon CloudFront

マイクロサービス

APIsはアプリケーションロジックのエントリポイントであるため、マイクロサービスの玄関先と見なされます。通常、RESTful ウェブサービス API または GraphQL APIsが使用されます。これらの APIs、クライアント呼び出しを管理および処理し、トラフィック管理、リクエストフィルタリング、ルーティング、キャッシュ、認証、認可などの機能を処理します。

マイクロサービスの実装

AWS は、コンテナオーケストレーションエンジンの選択肢として Amazon ECS や Amazon EKS、ホスティングオプションとして AWS Fargate EC2 など、マイクロサービスを開発するための構成要素を提供します。 AWS Lambda は、マイクロサービスを構築するもう 1 つのサーバーレス方法です AWS。これらのホスティングオプションの選択は、基盤となるインフラストラクチャを管理するためのお客様の要件によって異なります。

AWS Lambda を使用すると、コードをアップロードし、高可用性で実行を自動的にスケーリングおよび管理できます。これにより、インフラストラクチャ管理が不要になるため、迅速に移行し、ビジネスロジックに集中できます。Lambda は複数のプログラミング言語をサポートしており、他の AWS サービスによってトリガーしたり、ウェブやモバイルアプリケーションから直接呼び出すことができます。

コンテナベースのアプリケーションは、移植性、生産性、効率性により人気が高まっています。AWS は、コンテナを構築、デプロイ、管理するための複数のサービスを提供しています。

- App2Container は、Java および .NET ウェブアプリケーションをコンテナ形式に移行およびモダナイズするためのコマンドラインツールです。 AWS A2C は、ベアメタル、仮想マシン、Amazon Elastic Compute Cloud (EC2) インスタンス、またはクラウドで実行されているアプリケーションのインベントリを分析して構築します。

- Amazon Elastic Container Service (Amazon ECS) と Amazon Elastic Kubernetes Service (Amazon EKS) は、コンテナインフラストラクチャを管理し、コンテナ化されたアプリケーションの起動と保守を容易にします。
  - Amazon EKS は、 AWS クラウドおよびオンプレミスのデータセンター (Amazon EKS Anywhere) で Kubernetes を実行するためのマネージド Kubernetes サービスです。これにより、クラウドサービスがオンプレミス環境に拡張され、低レイテンシー、ローカルデータ処理、高データ転送コスト、またはデータレジデンシー要件に対応できるようになります (Amazon EKS Anywhere によるハイブリッドコンテナワークロードの実行」に関するホワイトペーパーを参照) 。EKS では、Kubernetes コミュニティのすべての既存のプラグインとツールを使用できます。
  - Amazon Elastic Container Service (Amazon ECS) は、コンテナ化されたアプリケーションのデプロイ、管理、スケーリングを簡素化するフルマネージドコンテナオーケストレーションサービスです。お客様は、シンプルさと AWS サービスとの深い統合のために ECS を選択します。

詳細については、ブログ「Amazon ECS vs Amazon EKS: AWS コンテナサービスの意味」を参照してください。

- AWS App Runner はフルマネージド型のコンテナアプリケーションサービスで、コンテナ化されたウェブアプリケーションと API サービスを構築、デプロイ、および実行できます。インフラストラクチャやコンテナに関する経験は必要ありません。
- AWS Fargateサーバーレスコンピューティングエンジンである は、Amazon ECS と Amazon EKS の両方と連携して、コンテナアプリケーションのコンピューティングリソースを自動的に管理します。
- Amazon ECR は、高性能ホスティングを提供するフルマネージドコンテナレジストリであるため、アプリケーションイメージとアーティファクトをどこにでも確実にデプロイできます。

継続的インテグレーションと継続的デプロイ (CI/CD)

継続的インテグレーションと継続的デリバリー (CI/CD) は、迅速なソフトウェア変更のための DevOps イニシアチブの重要な部分です。 はマイクロサービスに CI/CD を実装するサービス AWS を提供しますが、詳細な説明は本書の範囲外です。詳細については、ホワイトペーパーの「継続的インテグレーションと継続的デリバリーの実践 AWS」を参照してください。

プライベートネットワーク設定

AWS PrivateLink は、Virtual Private Cloud (VPC) とサポートされている サービス間のプライベート接続を許可することで、マイクロ AWS サービスのセキュリティを強化するテクノロジーです。マイクロサービストラフィックを分離して保護し、パブリックインターネットを経由しないようにします。これは、PCI や HIPAA などの規制に準拠するために特に役立ちます。

データストア

データストアは、マイクロサービスに必要なデータを永続化するために使用します。セッションデータの一般的なストアは、Memcached や Redis. AWS offers などのインメモリキャッシュであり、両方のテクノロジーを managedAmazon ElastiCacheservice の一部として提供します。

アプリケーションサーバーとデータベース間にキャッシュを配置することは、データベースの読み取り負荷を減らすための一般的なメカニズムであり、これにより、より多くの書き込みをサポートするためにリソースを使用できるようになります。キャッシュはレイテンシーを改善することもできます。

リレーショナルデータベースは、構造化データとビジネスオブジェクトを保存するために依然として非常に人気があります。 は、Amazon Relational Database Service (Amazon RDS) を通じて マネージドサービスとして 6 つのデータベースエンジン (Microsoft SQL Server、Oracle、MySQL、MariaDB、PostgreSQL、Amazon Aurora) AWS を提供しています。

ただし、リレーショナルデータベースは無限規模向けに設計されていないため、多数のクエリをサポートする手法を適用するのが難しく、時間がかかる可能性があります。

NoSQL データベースは、リレーショナルデータベースの一貫性よりもスケーラビリティ、パフォーマンス、可用性を優先するように設計されています。NoSQL データベースの重要な要素の 1 つは、通常、厳密なスキーマを適用しないことです。データは、水平方向にスケーリングできるパーティションに分散され、パーティションキーを使用して取得されます。

個々のマイクロサービスは 1 つのことをうまく実行するように設計されているため、通常、NoSQL 永続性に適した簡易データモデルがあります。NoSQL データベースにはリレーショナルデータベースとは異なるアクセスパターンがあることを理解することが重要です。例えば、テーブルを結合することはできません。これが必要な場合は、ロジックをアプリケーションに実装する必要があります。Amazon DynamoDB を使用して、任意の量のデータを保存および取得し、任意のレベルのリクエストトラフィックを処理できるデータベーステーブルを作成できます。DynamoDB は 1 桁ミリ秒のパフォーマンスを提供しますが、マイクロ秒単位の応答時間を必要とするユースケースもあります。DynamoDB Accelerator (DAX) には、データにアクセスするためのキャッシュ機能があります。

DynamoDB には、実際のトラフィックに応じてスループットキャパシティを動的に調整する自動スケーリング機能も用意されています。ただし、アプリケーションのアクティビティの急増が短時間で発生するため、キャパシティプランニングが困難または不可能な場合があります。このような状況では、DynamoDB にはオンデマンドオプションが用意されており、pay-per-requestが提供されます。DynamoDB オンデマンドは、キャパシティプランニングなしで 1 秒あたり数千のリクエストを即座に処理できます。

詳細については、分散データ管理「」および「データベースの選択方法」を参照してください。

オペレーションの簡素化

マイクロサービスの実行、保守、モニタリングに必要な運用作業をさらに簡素化するために、フルサーバーレスアーキテクチャを使用できます。

Lambda ベースのアプリケーションのデプロイ

Lambda コードをデプロイするには、zipファイルアーカイブをアップロードするか、有効な Amazon ECR イメージ URI を使用してコンソール UI からコンテナイメージを作成してアップロードします。ただし、Lambda 関数が複雑になった場合、つまりレイヤー、依存関係、アクセス許可がある場合、UI を介したアップロードはコードの変更に扱いにくい場合があります。

AWS CloudFormation と AWS Serverless Application Model （ AWS SAM ）、または Terraform を使用すると AWS Cloud Development Kit (AWS CDK)、サーバーレスアプリケーションを定義するプロセスが合理化されます。CloudFormation でネイティブにサポートされている AWS SAM は、サーバーレスリソースを指定するためのシンプルな構文を提供します。AWS Lambda レイヤーは、複数の Lambda 関数間で共有ライブラリを管理し、関数のフットプリントを最小限に抑え、テナント対応ライブラリを一元化して、開発者エクスペリエンスを向上させるのに役立ちます。Lambda SnapStart for Java は、レイテンシーの影響を受けやすいアプリケーションの起動パフォーマンスを向上させます。

デプロイするには、CloudFormation テンプレートでリソースとアクセス許可ポリシーを指定し、デプロイアーティファクトをパッケージ化し、テンプレートをデプロイします。 AWS CLI ツールである SAM Local は、Lambda にアップロードする前に、サーバーレスアプリケーションのローカル開発、テスト、分析を許可します。

IDE、 AWS CodeBuild、 AWS CodePipeline などのツールとの統合により AWS CodeDeploy、SAM AWS Cloud9 ベースのアプリケーションの作成、テスト、デバッグ、デプロイが効率化されます。

次の図は、CloudFormation および AWS CI/CD ツールを使用した AWS Serverless Application Model リソースのデプロイを示しています。

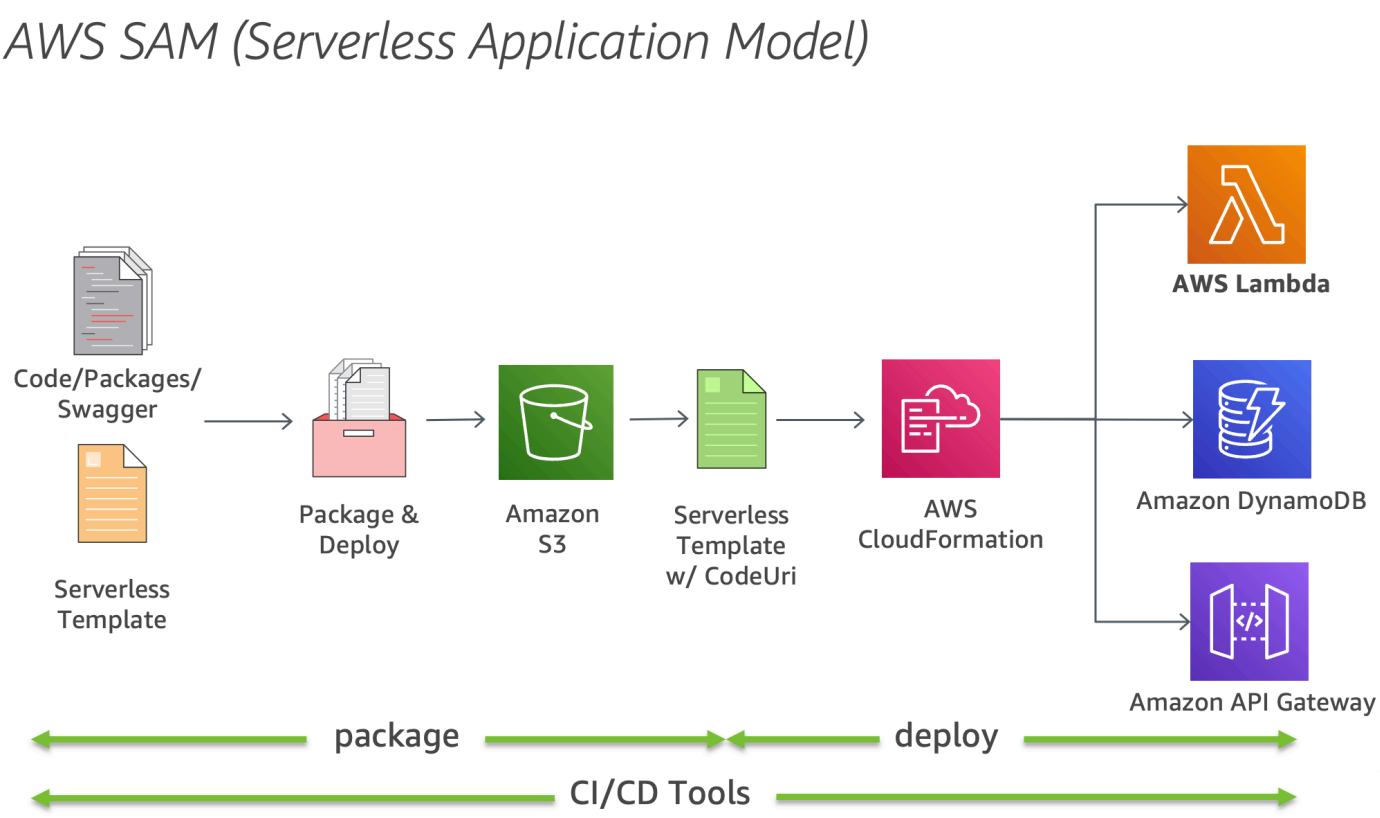

図 2: AWS Serverless Application Model （ AWS SAM ）

マルチテナンシーの複雑さの抽象化

SaaS プラットフォームなどのマルチテナント環境では、マルチテナンシーに関連する複雑さを合理化し、開発者が機能と機能の開発に集中できるようにすることが重要です。これは、クロスカットの問題に対処するための共有ライブラリを提供する AWS Lambda Layers などのツールを使用して実現できます。このアプローチの理論的根拠は、共有ライブラリとツールを正しく使用すると、テナントコンテキストを効率的に管理することです。

ただし、複雑でリスクがあるため、ビジネスロジックのカプセル化にまで拡張すべきではありません。共有ライブラリの根本的な問題は、更新に伴う複雑さが増し、標準コードの重複に比べて管理が難しくなることです。したがって、最も効果的な抽象化を求めるには、共有ライブラリの使用と重複のバランスを取ることが重要です。

API 管理

APIs の管理は、特に複数のバージョン、開発サイクルの段階、認可、スロットリングやキャッシュなどの機能を考慮する場合、時間がかかることがあります。API Gateway とは別に、API 管理に

ALB (Application Load Balancer) または NLB (Network Load Balancer) を使用するお客様もいます。Amazon API Gateway は、RESTful APIs。これにより、プログラムで APIs を作成し、バックエンドサービス、認可とアクセスコントロール、レート制限、キャッシュ、モニタリング、トラフィック管理からデータ、ビジネスロジック、機能にアクセスするための「フロントドア」として機能し、サーバーを管理することなく APIs を実行できます。

図 3 は、API Gateway が API コールを処理し、他のコンポーネントとやり取りする方法を示しています。モバイルデバイス、ウェブサイト、またはその他のバックエンドサービスからのリクエストは、レイテンシーを短縮し、最適なユーザーエクスペリエンスを提供するために、最も近い CloudFront の Point of Presence (PoP) にルーティングされます。

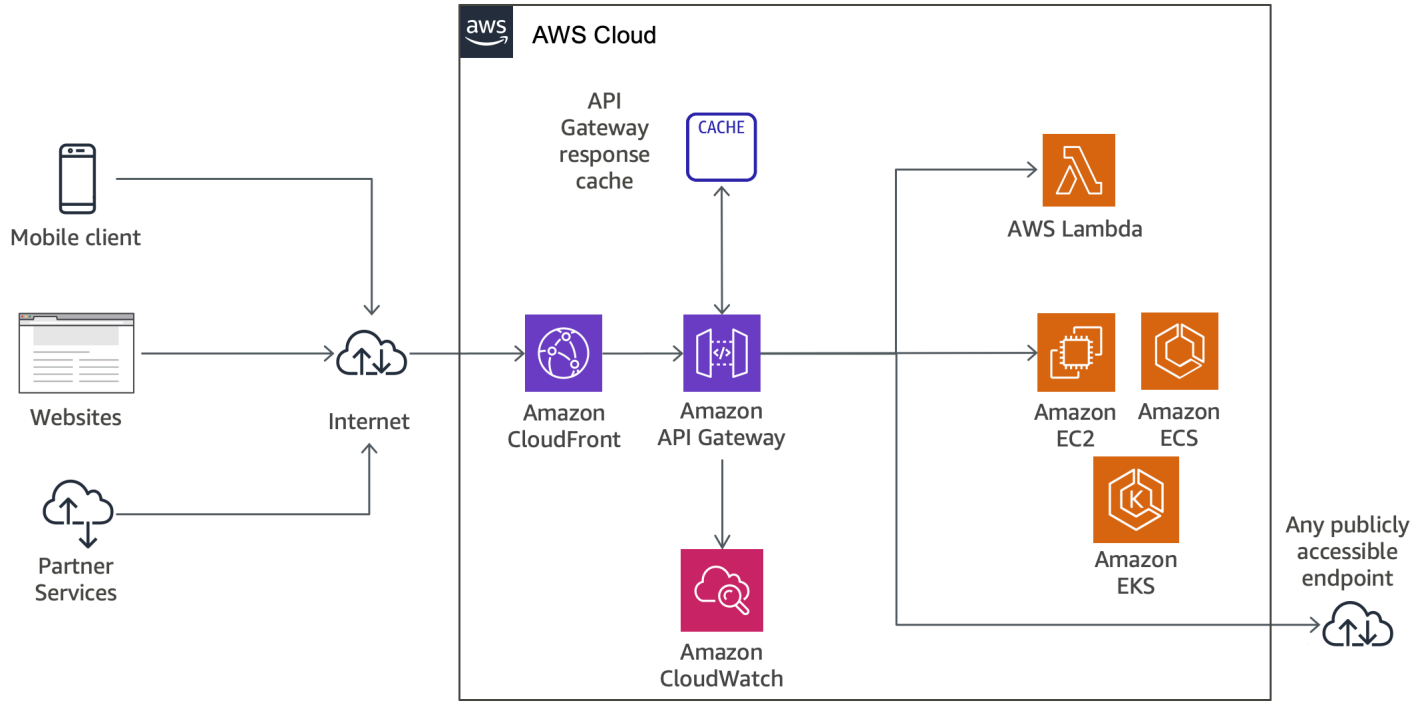

図 3: API Gateway コールフロー

サーバーレステクノロジーのマイクロサービス

サーバーレステクノロジーでマイクロサービスを使用すると、運用の複雑さを大幅に軽減できます。AWS Lambda また AWS Fargate、API Gateway と統合された により、完全なサーバーレスアプリケーションを作成できます。2023 年 4 月 7 日以降、Lambda 関数はレスポンスペイロードをクライアントに段階的にストリーミングし直し、ウェブアプリケーションとモバイルアプリケーションのパフォーマンスを向上させることができます。これ以前は、従来のリクエスト/レスポンス呼び出しモデルを使用する Lambda ベースのアプリケーションは、クライアントに返す前にレスポンスを生成してバッファする必要がありました。これにより、最初のバイトまでの時間が遅延する可能性があります。レスポンスストリーミングを使用すると、関数は準備ができたら部分的なレスポンスをクライアントに送り返すことができるため、最初のバイトまでの時間が大幅に短縮され、ウェブおよびモバイルアプリケーションは特に影響を受けやすくなります。

図 4 は、 AWS Lambda および マネージドサービスを使用したサーバーレスマイクロサービスアーキテクチャを示しています。このサーバーレスアーキテクチャにより、スケールと高可用性のための設計の必要性が軽減され、基盤となるインフラストラクチャの実行とモニタリングに必要な労力が削減されます。

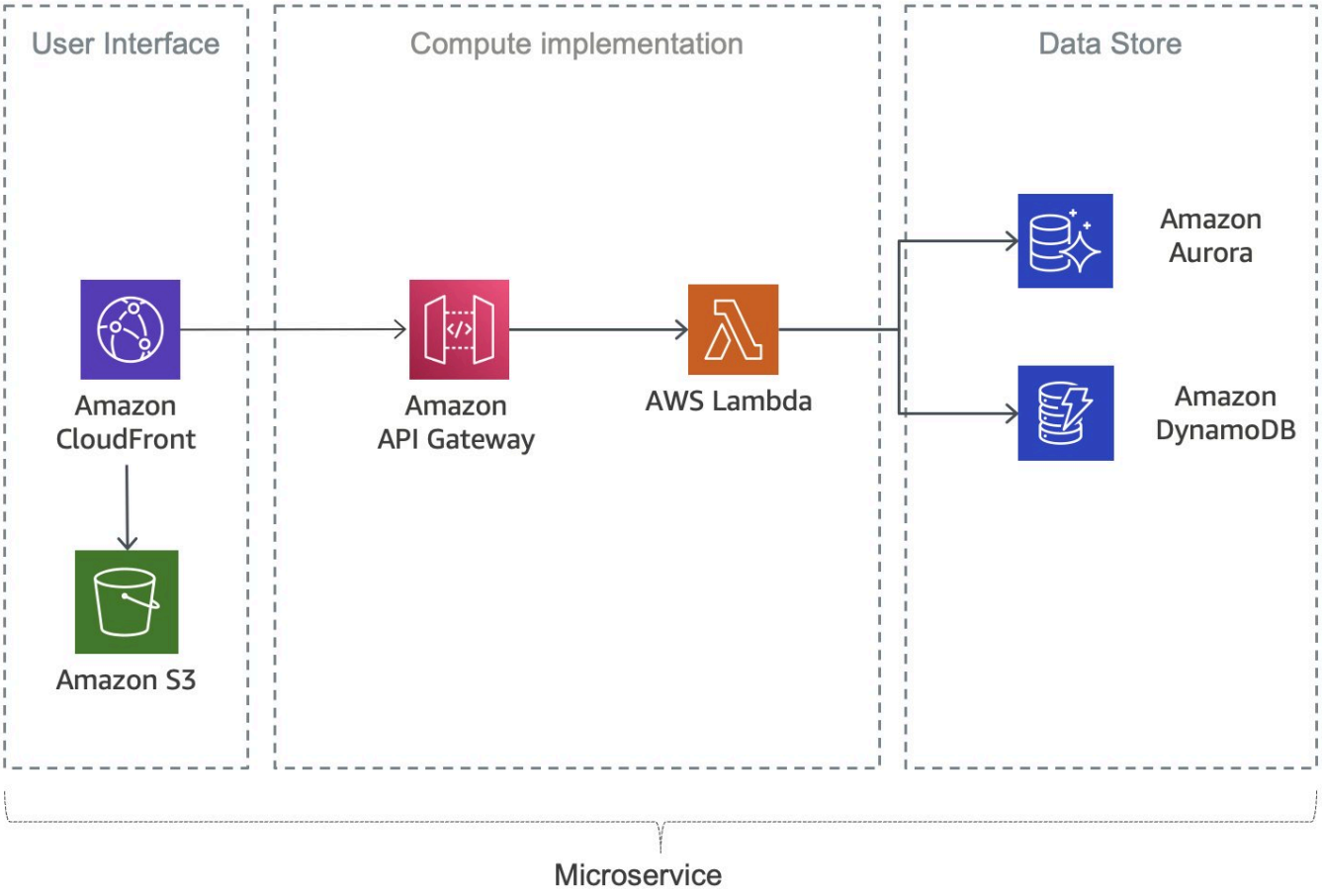

図 4: を使用したサーバーレスマイクロサービス AWS Lambda

図 5 は、 のコンテナを使用した同様のサーバーレス実装を示しています。基盤となるインフラストラクチャに関する懸念 AWS Fargateは排除されています。また、Amazon Aurora Serverless も備えています。Amazon Aurora Serverless は、アプリケーションの要件に基づいて容量を自動的に調整するオンデマンドの自動スケーリングデータベースです。

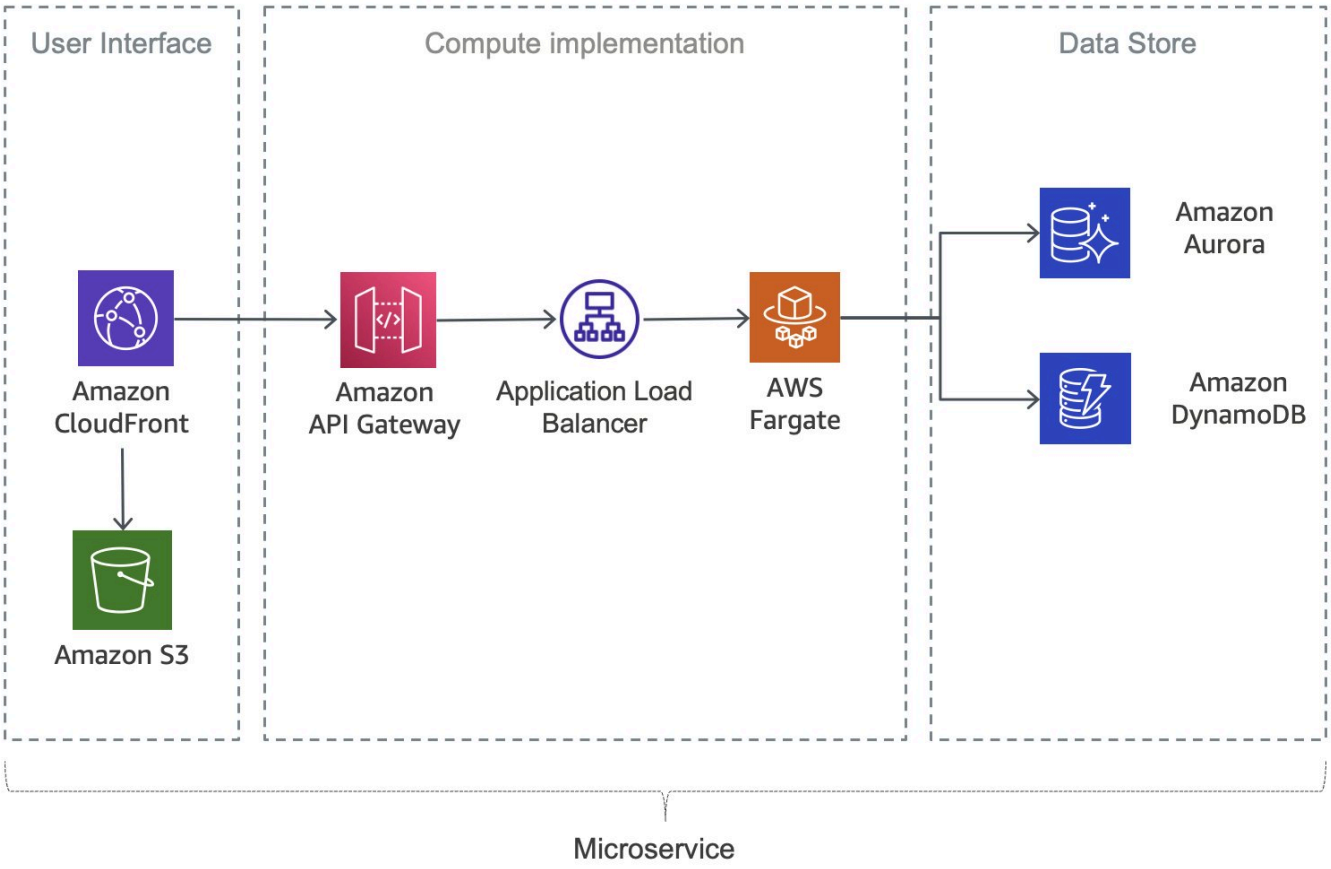

図 5: を使用したサーバーレスマイクロサービス AWS Fargate

回復力と効率に優れたシステム

ディザスタリカバリ (DR)

マイクロサービスアプリケーションは、多くの場合、12 要素のアプリケーションパターンに従います。このプロセスはステートレスで、永続データはデータベースなどのステートフルバッキングサービスに保存されます。これにより、サービスが失敗すると、新しいインスタンスを簡単に起動して機能を復元できるため、ディザスタリカバリ (DR) が簡素化されます。

マイクロサービスのディザスタリカバリ戦略は、ファイルシステム、データベース、キューなど、アプリケーションの状態を維持するダウンストリームサービスに焦点を当てる必要があります。組織は、目標復旧時間 (RTO) と目標復旧時点 (RPO) を計画する必要があります。RTO はサービスの中断と復元の間の最大許容遅延時間であり、RPO は最後のデータ復旧ポイントからの最大時間です。

ディザスタリカバリ戦略の詳細については、 クラウドホワイトペーパーの「ワークロードのディザスタリカバリ AWS: リカバリ」を参照してください。

高可用性 (HA)

マイクロサービスアーキテクチャのさまざまなコンポーネントの高可用性 (HA) を調べます。

Amazon EKS は、複数のアベイラビリティーゾーンで Kubernetes コントロールインスタンスとデータプレーンインスタンスを実行することで、高可用性を実現します。異常なコントロールプレーンインスタンスを自動的に検出して置き換え、バージョンのアップグレードとパッチ適用を自動化します。

Amazon ECR は、ストレージに Amazon Simple Storage Service (Amazon S3) を使用して、コンテナイメージの可用性とアクセス性を高めます。Amazon EKS、Amazon ECS、および で動作 AWS Lambdaし、本番ワークフローへの開発を簡素化します。

Amazon ECS は、リージョン内の複数のアベイラビリティーゾーンで高可用性方式でコンテナの実行を簡素化するリージョンサービスであり、リソースのニーズと可用性要件に合わせてコンテナを配置する複数のスケジューリング戦略を提供します。

AWS Lambda は複数のアベイラビリティーゾーンで動作し、1 つのゾーンでサービスの中断時に可用性を確保します。関数を VPC に接続する場合は、高可用性を実現するために複数のアベイラビリティーゾーンのサブネットを指定します。

分散システムコンポーネント

マイクロサービスアーキテクチャでは、サービス検出とは、分散システム内の個々のマイクロサービスのネットワークロケーション (IP アドレスとポート) を動的に特定するプロセスを指します。

アプローチを選択するときは AWS、次のような要素を考慮してください。

- コード変更： コードを変更せずに利点が得られますか？
- クロス VPC またはクロスアカウントトラフィック： 必要に応じて、システムは異なる VPCs 間の通信を効率的に管理する必要がありますか AWS アカウント？
- デプロイ戦略： システムは Blue-Green や Canary デプロイなどの高度なデプロイ戦略を使用しているか、使用する予定がありますか？
- パフォーマンスに関する考慮事項： アーキテクチャが外部サービスと頻繁に通信する場合、全体的なパフォーマンスにはどのような影響がありますか？

AWS には、マイクロサービスアーキテクチャにサービス検出を実装するための方法がいくつか用意されています。

- Amazon ECS Service Discovery: Amazon ECS は、DNS ベースのメソッドを使用するか、 と統合してサービス検出をサポートします AWS Cloud Map (ECS Service discovery を参照) 。 https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-discovery.htmlECS Service Connect は接続管理をさらに改善します。これは、複数のインタラクションサービスを持つ大規模なアプリケーションに特に役立ちます。
- Amazon Route 53:Route 53 は、ECS や EKS などの他の AWS サービスと統合して、サービスの検出を容易にします。ECS コンテキストでは、Route 53 は ECS サービス検出機能を使用できます。この機能は、自動命名 API を活用して、サービスを自動的に登録および登録解除します。
- AWS Cloud Map： このオプションは、動的な API ベースのサービス検出を提供し、サービス全体に変更を伝達します。

より高度な通信ニーズのために、Amazon VPC Lattice は、サービス間の通信を一貫して接続、監視、保護するアプリケーションネットワークサービスであり、生産性の向上に役立ち、デベロッパーがビジネスにとって重要な機能の構築に集中できるようにします。ネットワークトラフィック管理、アクセス、モニタリングのポリシーを定義して、インスタンス、コンテナ、サーバーレスアプリケーション間でコンピューティングサービスをシンプルで一貫した方法で接続できます。

HashiCorp Consul や Netflix Eureka などのサードパーティー製ソフトウェアをサービス検出に既に使用している場合は、移行時にこれらのソフトウェアを引き続き使用して AWS、よりスムーズな移行を可能にすることをお勧めします。

これらのオプションの選択は、特定のニーズと一致する必要があります。より単純な要件では、Amazon ECS や などの DNS ベースのソリューションで十分 AWS Cloud Map かもしれません。より複雑なシステムや大規模なシステムでは、Amazon VPC Lattice などのサービスメッシュの方が適している場合があります。

結論として、 でマイクロサービスアーキテクチャを設計 AWS することは、特定のニーズを満たす適切なツールを選択することです。説明した考慮事項を念頭に置いて、システムのサービス検出とサービス間通信を最適化するために、十分な情報に基づいた意思決定を行うことができます。

分散データ管理

従来のアプリケーションでは、多くの場合、すべてのコンポーネントが 1 つのデータベースを共有します。対照的に、マイクロサービスベースのアプリケーションの各コンポーネントは独自のデータを維持し、独立性と分散化を促進します。このアプローチは分散データ管理と呼ばれ、新しい課題をもたらします。

このような課題の 1 つは、分散システムの一貫性とパフォーマンスのトレードオフから発生します。多くの場合、即時更新 (即時整合性) を要求するよりも、データ更新のわずかな遅延 (結果整合性) を受け入れる方が現実的です。

場合によっては、ビジネスオペレーションで複数のマイクロサービスが連携する必要があります。1 つのパートが失敗した場合、完了したタスクの一部を元に戻す必要がある場合があります。Saga パターンは、一連の補償アクションを調整することで、これを管理するのに役立ちます。

マイクロサービスの同期を維持するために、一元化されたデータストアを使用できます。このストアは、 AWS Lambda AWS Step Functionsや Amazon EventBridge などのツールを使用して管理されており、データのクリーンアップや重複排除に役立ちます。

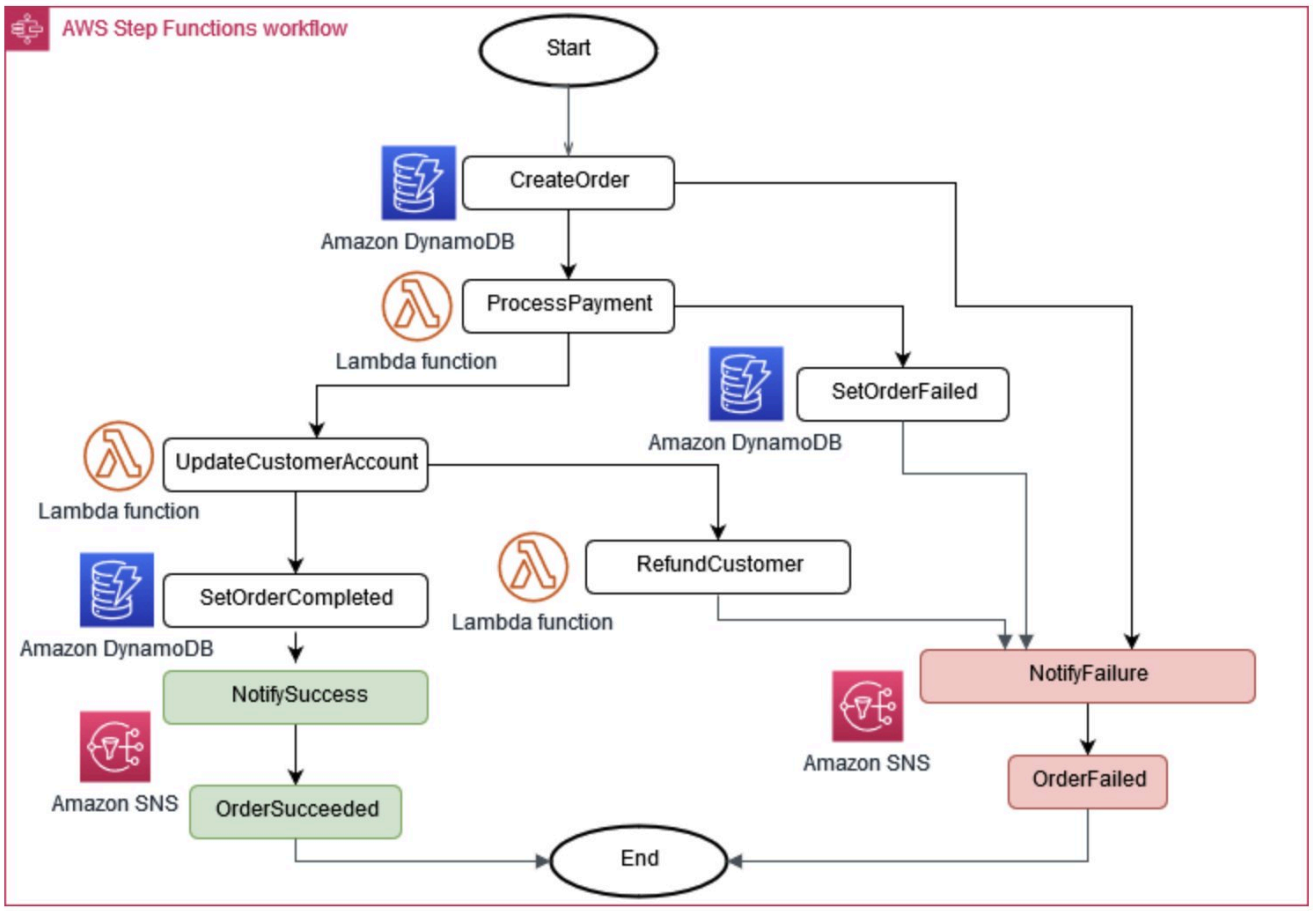

図 6: Saga 実行コーディネーター

マイクロサービス全体の変更を管理する一般的なアプローチは、イベントソーシングです。アプリケーションのすべての変更はイベントとして記録され、システムの状態のタイムラインが作成されます。このアプローチは、デバッグと監査に役立つだけでなく、アプリケーションのさまざまな部分が同じイベントに反応するのも可能になります。

イベントソーシングは、多くの場合、コマンドクエリ責任分離 (CQRS) パターン hand-in-hand と連携して機能します。CQRS パターンでは、データの変更とデータクエリを異なるモジュールに分割して、パフォーマンスとセキュリティを向上させます。

では AWS、サービスの組み合わせを使用してこれらのパターンを実装できます。図 7 に示すように、Amazon Kinesis Data Streams は中央イベントストアとして機能し、Amazon S3 はすべてのイベントレコードに耐久性のあるストレージを提供します。 AWS Lambda、Amazon DynamoDB、および Amazon API Gateway は連携してこれらのイベントを処理および処理します。

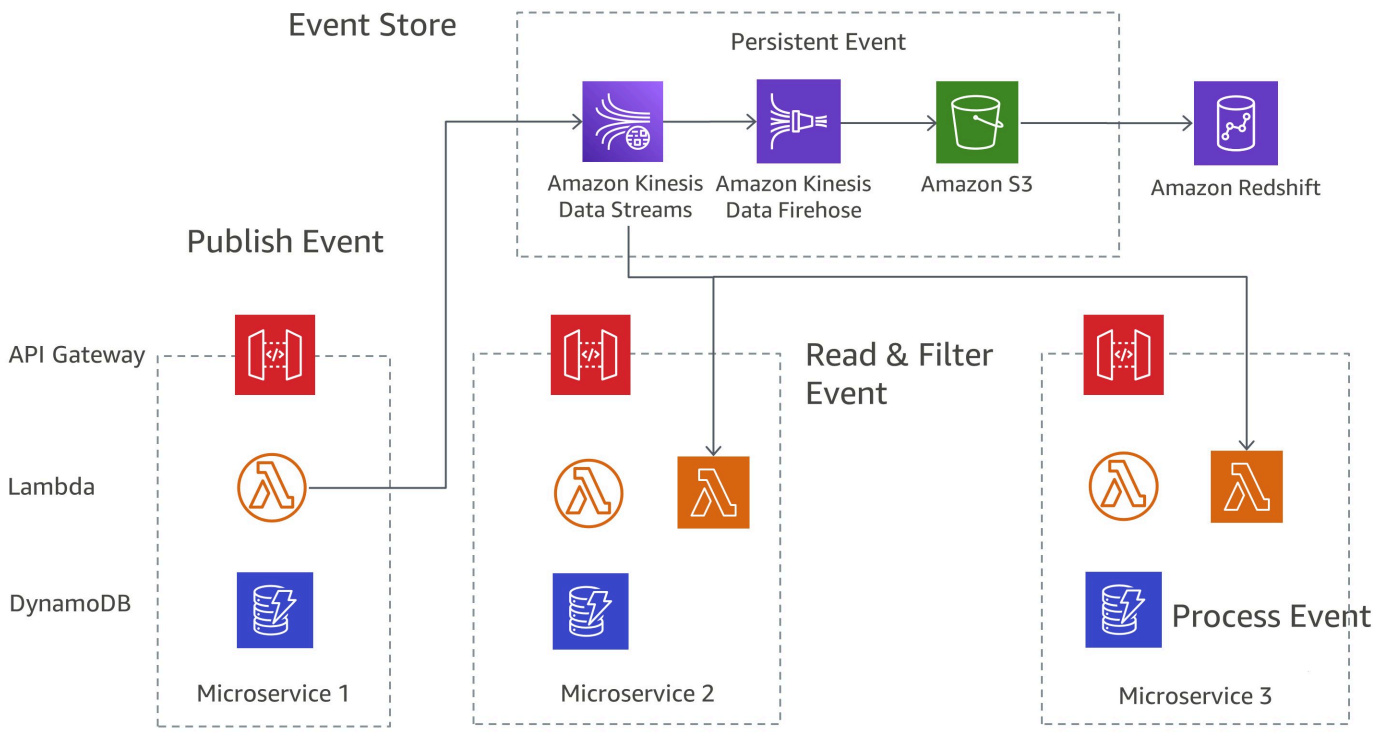

図 7: でのイベントソーシングパターン AWS

分散システムでは、再試行によってイベントが複数回配信される可能性があるため、これを処理するようにアプリケーションを設計することが重要です。

設定管理

マイクロサービスアーキテクチャでは、各サービスはデータベース、キュー、その他のサービスなどのさまざまなリソースとやり取りします。各サービスの接続と運用環境を一貫した方法で設定することが重要です。理想的には、アプリケーションは再起動を必要とせずに新しい設定に適応する必要があります。このアプローチは 12 要素のアプリ原則の一部であり、環境変数に設定を保存することをお勧めします。

別の方法は、AWS App Config を使用することです。これは AWS Systems Manager の機能であり、お客様は機能フラグとアプリケーション設定をすばやく安全に設定、検証、デプロイできます。機能フラグと設定データは、デプロイ前の段階で構文的にも意味的にも検証でき、設定したアラームがトリガーされた場合はモニタリングして自動的にロールバックできます。AppConfig は、 AWS AppConfig エージェントを使用して Amazon ECS および Amazon EKS と統合できます。エージェントは Amazon ECS および Amazon EKS コンテナアプリケーションと並行して実行されるサイドカーコンテナとして機能します。Lambda 関数で AppConfig 機能フラグまたはその他の動的設定データを使用する場合は AWS 、 AWS AppConfig Lambda 拡張機能を Lambda 関数のレイヤーとして追加することをお勧めします。

GitOps は、すべての設定変更の信頼できる情報源として Git を使用する設定管理に対する革新的なアプローチです。つまり、設定ファイルに加えられた変更は、Git を通じて自動的に追跡、バージョニング、監査されます。

シークレットの管理

セキュリティは最優先事項であるため、認証情報をプレーンテキストで渡さないでください。は、AWS Systems Manager パラメータストアや などの安全なサービス AWS を提供します AWS Secrets Manager。これらのツールは、Amazon EKS のコンテナにシークレットをボリュームとして、または Amazon ECS に環境変数として送信できます。では AWS Lambda、環境変数が自動的にコードで使用可能になります。Kubernetes ワークフローの場合、外部シークレットオペレーターはなどのサービスから直接シークレットを取得し AWS Secrets Manager、対応する Kubernetes シークレットを作成します。これにより、Kubernetes ネイティブ設定とシームレスに統合できます。

コストの最適化と持続可能性

マイクロサービスアーキテクチャは、コストの最適化と持続可能性を強化できます。アプリケーションを小さな部分に分割することで、より多くのリソースを必要とするサービスのみをスケールアップし、コストと無駄を削減できます。これは、可変トラフィックを処理する場合に特に便利です。マイクロサービスは独自に開発されています。そのため、デベロッパーは小規模な更新を行い、エンドツーエンドのテストに費やすリソースを削減できます。更新中は、モノリスではなく機能のサブセットのみをテストする必要があります。

アーキテクチャ内のステートレスコンポーネント (ローカルデータストアではなく外部データストアに状態を保存するサービス) は、 AWS クラウドで未使用の Amazon EC2 EC2 スポットインスタンスを使用できます。これらのインスタンスは、オンデマンドインスタンスよりもコスト効率が高く、中断を処理できるワークロードに最適です。これにより、高可用性を維持しながらコストをさらに削減できます。

分離されたサービスでは、Auto Scaling グループごとにコスト最適化コンピューティングオプションを使用できます。例えば、 AWS Graviton は、ARM ベースのインスタンスに適したワークロードに対して、費用対効果の高い高性能コンピューティングオプションを提供します。

コストとリソースの使用量を最適化することで、 Well-Architected フレームワークの持続可能性の柱に沿った環境への影響を最小限に抑えることもできます。 AWS Customer Carbon Footprint Tool を使用して、炭素排出量の削減の進行状況をモニタリングできます。このツールは、 AWS の使用による環境への影響に関するインサイトを提供します。

コミュニケーションメカニズム

マイクロサービスパラダイムでは、アプリケーションのさまざまなコンポーネントがネットワーク経由で通信する必要があります。この一般的なアプローチには、REST ベース、GraphQL ベース、gRPC ベース、非同期メッセージングなどがあります。

REST ベースの通信

HTTP/S プロトコルは、マイクロサービス間の同期通信に広く使用され、RESTful APIs を介して動作することがよくあります。API Gateway は、トラフィック管理、認可、モニタリング、バージョン管理などのタスクを処理するバックエンドサービスへの一元化されたアクセスポイントとして機能する API を効率的に構築する方法を提供します。

GraphQL ベースの通信

同様に、GraphQL は同期通信のための広範な方法で、REST と同じプロトコルを使用しますが、単一のエンドポイントへの公開は制限されます。を使用すると AWS AppSync、 AWS サービスやデータストアと直接やり取りする GraphQL アプリケーションを作成して公開したり、ビジネスロジックに Lambda 関数を組み込むことができます。

gRPC ベースの通信

gRPC は、同期的で軽量、高性能、オープンソースの RPC 通信プロトコルです。gRPC は HTTP/2 を使用し、圧縮やストリームの優先順位付けなどのより多くの機能を有効にすることで、基盤となるプロトコルを改善します。バイナリエンコードされた Protobuf インターフェイス定義言語 (IDL) を使用するため、HTTP/2 バイナリフレーミングを利用します。

非同期メッセージングとイベントパス

非同期メッセージングを使用すると、キューを介してメッセージを送受信することで、 サービスが通信できます。これにより、サービスは疎結合のままになり、サービス検出が促進されます。

メッセージングは、次の 3 つのタイプで定義できます。

- メッセージキュー： メッセージキューは、メッセージの送信者 (プロデューサー) と受信者 (コンシューマー) を分離するバッファとして機能します。プロデューサーはメッセージをキューに

キューに入れ、コンシューマーはメッセージをキューから取り出して処理します。このパターンは、非同期通信、負荷の平準化、トラフィックのバーストの処理に役立ちます。

- Publish-Subscribe: publish-subscribe パターンでは、メッセージがトピックに発行され、関心のある複数のサブスクライバーがメッセージを受信します。このパターンにより、イベントまたはメッセージを複数のコンシューマーに非同期的にブロードキャストできます。
- イベント駆動型メッセージング： イベント駆動型メッセージングでは、システムで発生するイベントをキャプチャして対応します。イベントはメッセージブローカーに発行され、関心のあるサービスは特定のイベントタイプにサブスクライブします。このパターンにより、疎結合が可能になり、サービスは直接的な依存関係なしにイベントに反応できます。

これらの各メッセージタイプを実装するために、 は Amazon SQS、Amazon SNS、Amazon EventBridge、Amazon MQ、Amazon MSK などのさまざまなマネージドサービス AWS を提供します。これらのサービスには、特定のニーズに合わせてカスタマイズされた独自の機能があります。

- Amazon Simple Queue Service (Amazon SQS) と Amazon Simple Notification Service (Amazon SNS): 図 8 に示すように、これら 2 つのサービスは相互に補完し合い、Amazon SQS はメッセージを保存するためのスペースを提供し、Amazon SNS は複数のサブスクライバーへのメッセージの配信を可能にします。同じメッセージを複数の宛先に配信する必要がある場合に有効です。

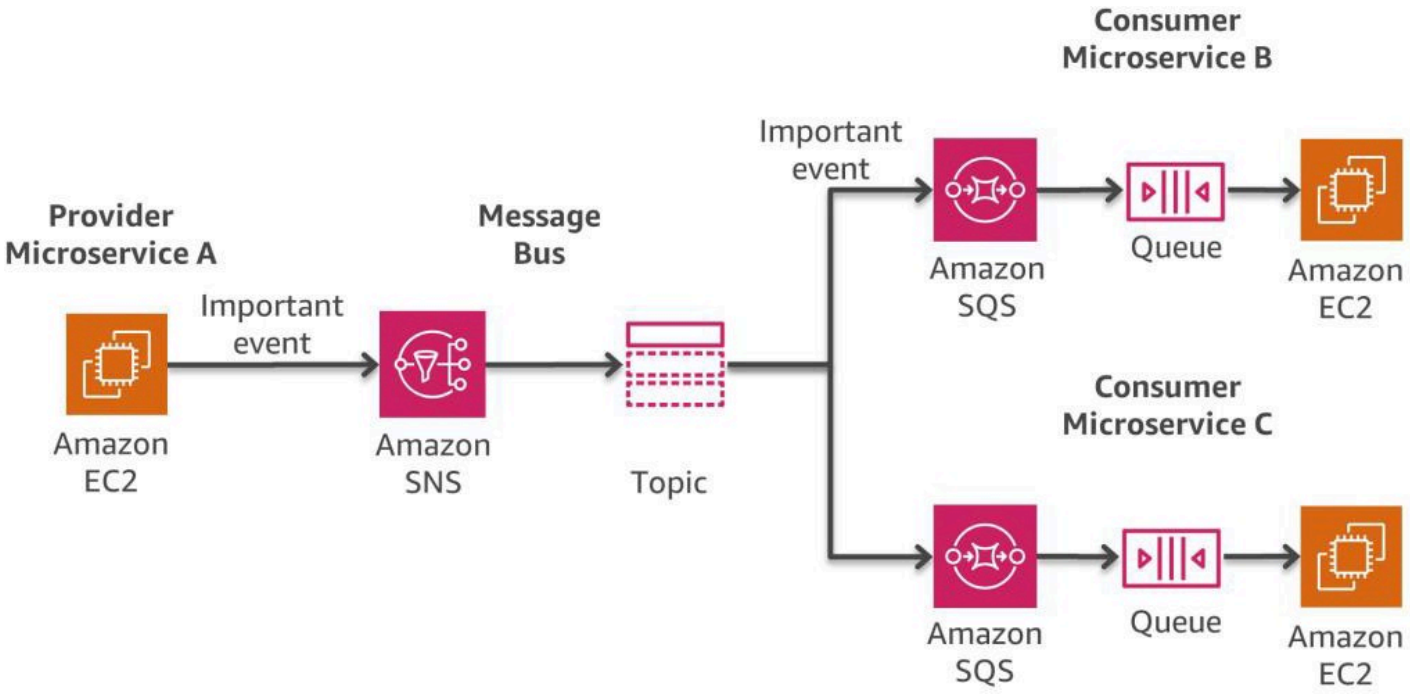

図 8: でのメッセージバスパターン AWS

- Amazon EventBridge: イベントを使用してアプリケーションコンポーネントを接続するサーバーレスサービス。スケーラブルなイベント駆動型アプリケーションを簡単に構築できます。これを使用して、自社開発のアプリケーション、 AWS サービス、サードパーティー製ソフトウェアなどのソースから組織全体のコンシューマーアプリケーションにイベントをルーティングします。EventBridge では、イベントの取り込み、フィルタリング、変換、配信をシンプルかつ一貫性のある方法で行うことができるため、新しいアプリケーションをすばやく構築できます。EventBridge イベントバスは、イベント駆動型サービス間のイベントの多対多のルーティングに適しています。
- Amazon MQ: JMS、AMQP などの標準プロトコルを使用する既存のメッセージングシステムがある場合に適しています。このマネージドサービスは、オペレーションを中断することなく、システムの代わりとなります。
- Amazon MSK (Managed Kafka): メッセージを保存および読み取るためのメッセージングシステム。メッセージを複数回処理する必要がある場合に役立ちます。また、リアルタイムのメッセージストリーミングもサポートしています。
- Amazon Kinesis: ストリーミングデータのリアルタイム処理と分析。これにより、リアルタイムアプリケーションの開発が可能になり、 AWS エコシステムとのシームレスな統合が可能になります。

最適なサービスは特定のニーズに依存するため、各サービスが提供する内容と要件との整合性を理解することが重要です。

オーケストレーションと状態管理

マイクロサービスオーケストレーションとは、オーケストレーターと呼ばれる中央コンポーネントがマイクロサービス間のインタラクションの管理と調整を担当する一元化されたアプローチを指します。複数のマイクロサービス間でワークフローをオーケストレーションするのは難しい場合があります。オーケストレーションコードを サービスに直接埋め込むことはお勧めしません。オーケストレーションコードは、結合が厳しくなり、個々のサービスの置き換えが妨げられるためです。

Step Functions は、エラー処理やシリアル化などのサービスオーケストレーションの複雑さを管理するワークフローエンジンを提供します。これにより、調整コードを追加することなく、アプリケーションのスケーリングと変更をすばやく行うことができます。Step Functions は AWS サーバーレスプラットフォームの一部であり、Lambda 関数、Amazon EC2、Amazon EKS、Amazon ECS、SageMaker AI AWS Glueなどをサポートしています。

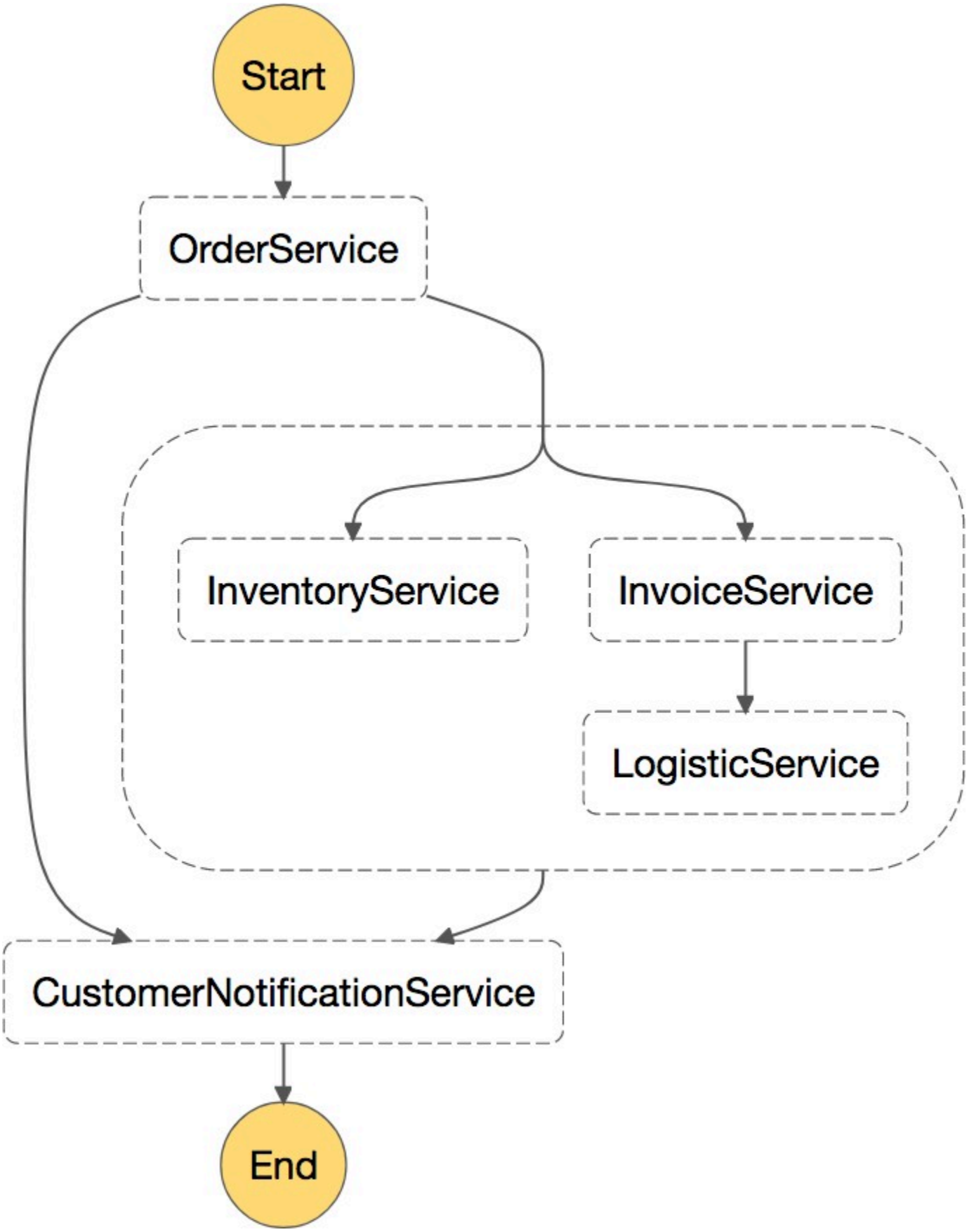
Start
OrderService
InventoryService
InvoiceService
LogisticService
CustomerNotificationService
End

図 9: によって呼び出される並列ステップとシーケンシャルステップを含むマイクロサービスワークフローの例 AWS Step Functions

Amazon Managed Workflows for Apache Airflow (Amazon MWAA) は Step Functions の代替手段です。オープンソースと移植性を優先する場合は、Amazon MWAA を使用する必要があります。Airflow には、新しい機能と統合を定期的に提供する大規模でアクティブなオープンソースコミュニティがあります。

オブザーバビリティ

マイクロサービスアーキテクチャは本質的に多くの分散コンポーネントで構成されているため、これらのコンポーネント全体のオブザーバビリティが重要になります。Amazon CloudWatch では、メトリクスの収集と追跡、ログファイルのモニタリング、 AWS 環境の変更への対応を行うことができます。アプリケーションとサービスによって生成された AWS リソースとカスタムメトリクスをモニタリングできます。

トピック

- モニタリング
- ログの一元化
- 分散トレース
- での口グ分析 AWS
- 分析のその他のオプション

モニタリング

CloudWatch は、リソース使用率、アプリケーションパフォーマンス、運用状態をシステム全体で可視化します。マイクロサービスアーキテクチャでは、開発者が収集するメトリクスを選択できるため、CloudWatch によるカスタムメトリクスのモニタリングが有益です。動的スケーリングは、これらのカスタムメトリクスに基づくこともできます。

CloudWatch Container Insights はこの機能を拡張し、CPU、メモリ、ディスク、ネットワークなどの多くのリソースのメトリクスを自動的に収集します。コンテナ関連の問題を診断し、解決を合理化するのに役立ちます。

Amazon EKS では、多くの場合、Prometheus が推奨されます。Prometheus は、包括的なモニタリングとアラート機能を提供するオープンソースプラットフォームです。通常、直感的なメトリクスの視覚化のために Grafana と結合されます。Amazon Managed Service for Prometheus (AMP) は、Prometheus と完全に互換性があるモニタリングサービスを提供し、コンテナ化されたアプリケーションを簡単に監視できます。さらに、Amazon Managed Grafana (AMG) はメトリクスの分析と視覚化を簡素化し、基盤となるインフラストラクチャを管理する必要がなくなります。

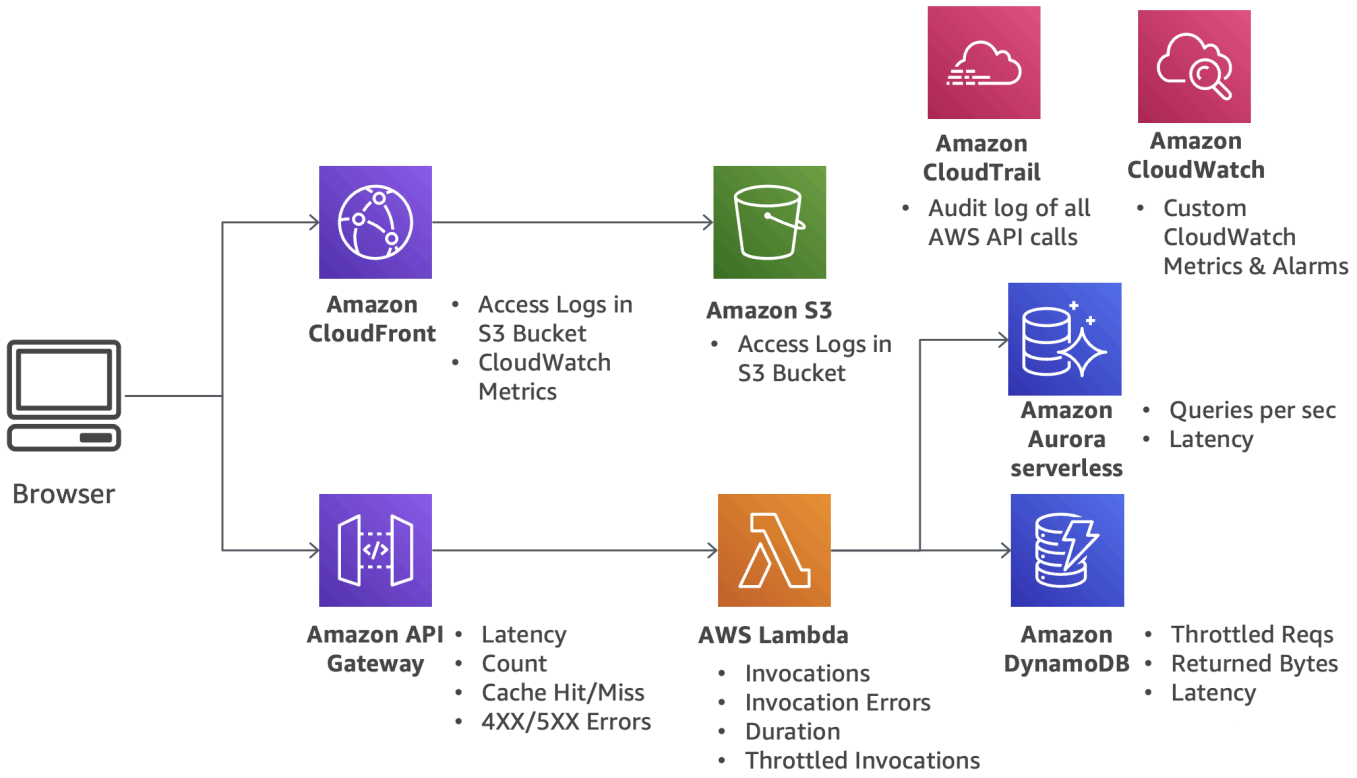

図 10: モニタリングコンポーネントを備えたサーバーレスアーキテクチャ

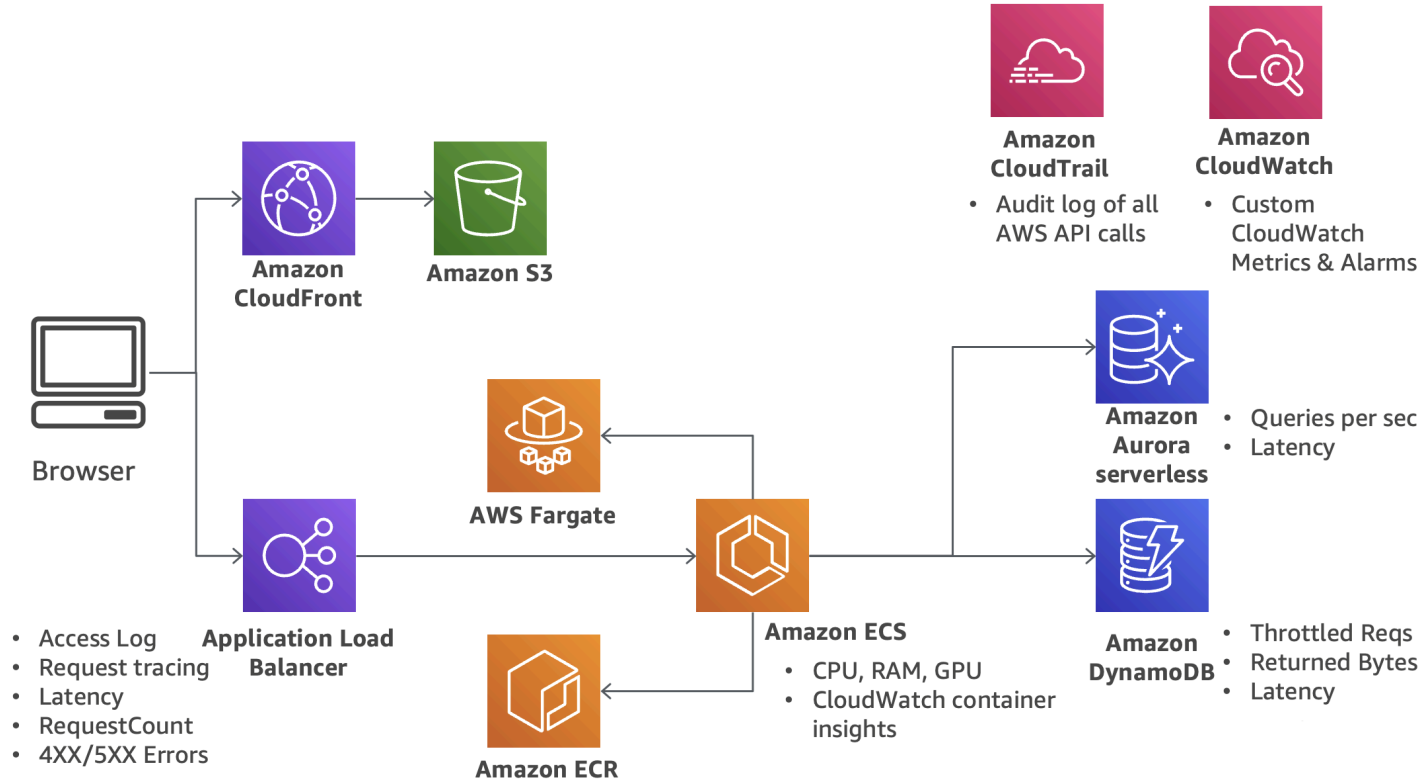

図 11: モニタリングコンポーネントを備えたコンテナベースのアーキテクチャ

ログの一元化

ログ記録は、問題を特定して解決するための鍵です。マイクロサービスを使用すると、より頻繁にリリースし、新機能を試すことができます。 は、ログファイルを一元化するために Amazon S3、CloudWatch Logs、Amazon OpenSearch Service などのサービス AWS を提供します。Amazon EC2 は CloudWatch にログを送信するためにデーモンを使用し、Lambda と Amazon ECS はログ出力をネイティブに送信します。Amazon EKS の場合、Fluent Bit または Fluentd を使用してログを CloudWatch に転送し、OpenSearch と Kibana を使用してレポートできます。CloudWatch OpenSearch ただし、フットプリントが小さく、パフォーマンス上の利点があるため、Fluentd よりも Fluent Bit が推奨されます。

図 12 は、さまざまな AWS サービスのログを Amazon S3 と CloudWatch に送信する方法を示しています。 CloudWatch これらの一元化されたログは、データの視覚化に Kibana を含む Amazon OpenSearch Service を使用してさらに分析できます。また、Amazon Athena は、Amazon S3 に保存されているログに対するアドホッククエリにも使用できます。

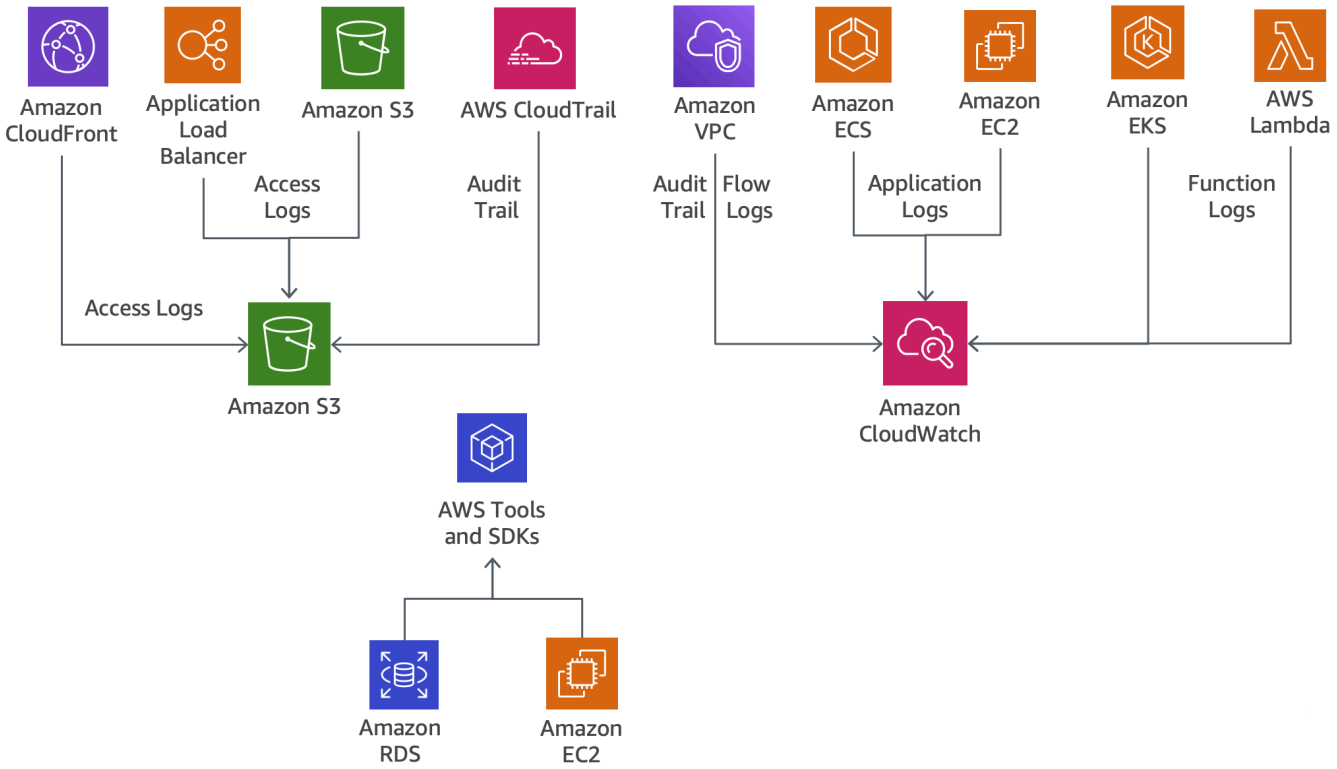

図 12: AWS サービスのログ記録機能

分散トレース

マイクロサービスは多くの場合、連携してリクエストを処理します。 は相関 ID AWS X-Ray を使用して、これらのサービス全体のリクエストを追跡します。 IDs X-Ray は、Amazon EC2、Amazon ECS、Lambda、Elastic Beanstalk と連携します。

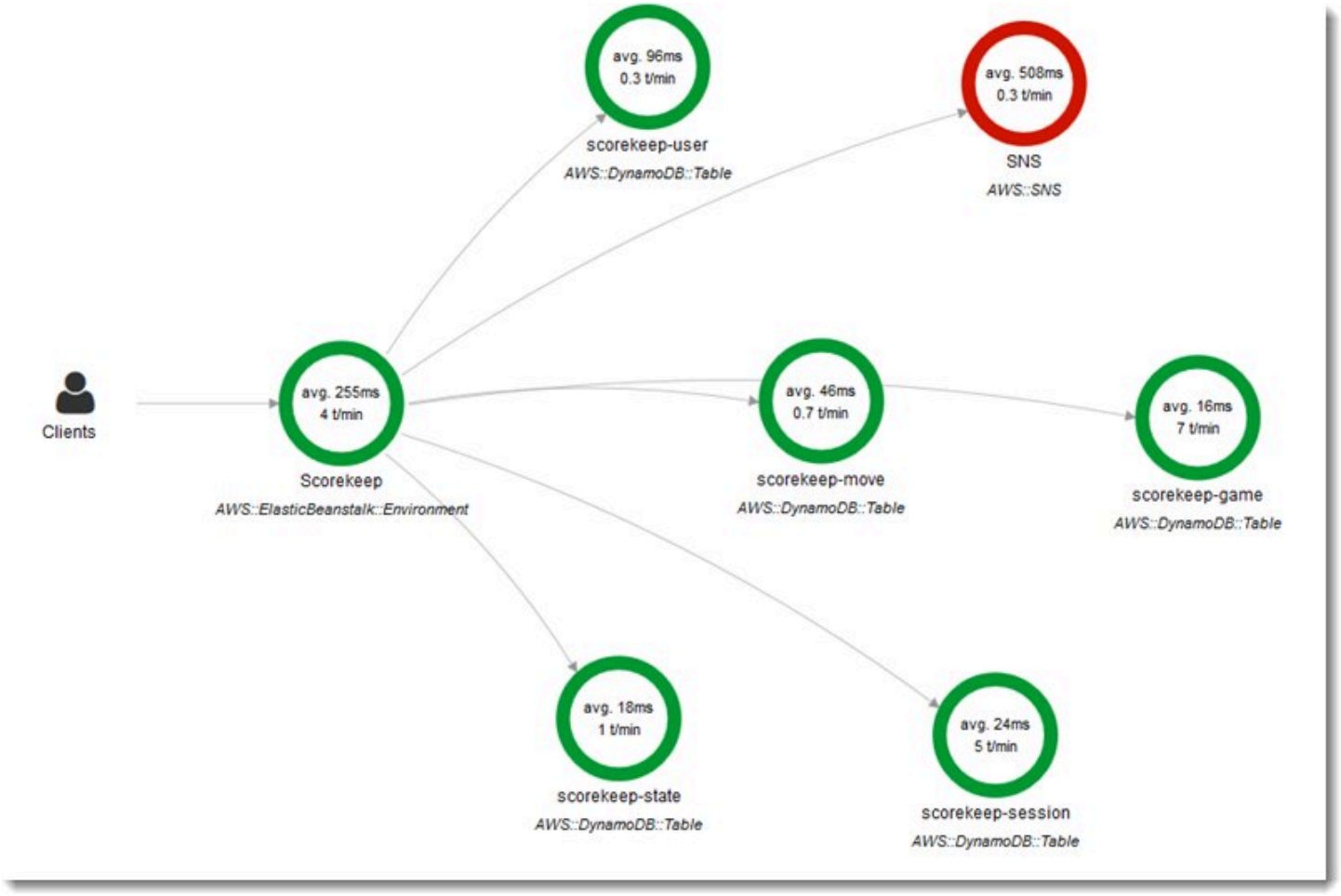

図 13: AWS X-Ray サービスマップ

AWS Distro for OpenTelemetry は OpenTelemetry プロジェクトの一部であり、分散トレースとメトリクスを収集するためのオープンソース APIs とエージェントを提供し、アプリケーションのモニタリングを向上させます。これにより、複数の AWS およびパートナーのモニタリングソリューションにメトリクスとトレースが送信されます。 AWS リソースからメタデータを収集することで、アプリケーションのパフォーマンスを基盤となるインフラストラクチャデータに合わせ、問題解決を加速します。さらに、さまざまな AWS サービスと互換性があり、オンプレミスで使用できます。

でのログ分析 AWS

Amazon CloudWatch Logs Insights を使用すると、リアルタイムのログ探索、分析、視覚化が可能になります。さらにログファイルを分析するには、Kibana を含む Amazon OpenSearch Service が強力なツールです。CloudWatch Logs は、ログエントリを OpenSearch Service にリアルタイムでストリーミングできます。OpenSearch とシームレスに統合された Kibana は、このデータを視覚化し、直感的な検索インターフェイスを提供します。

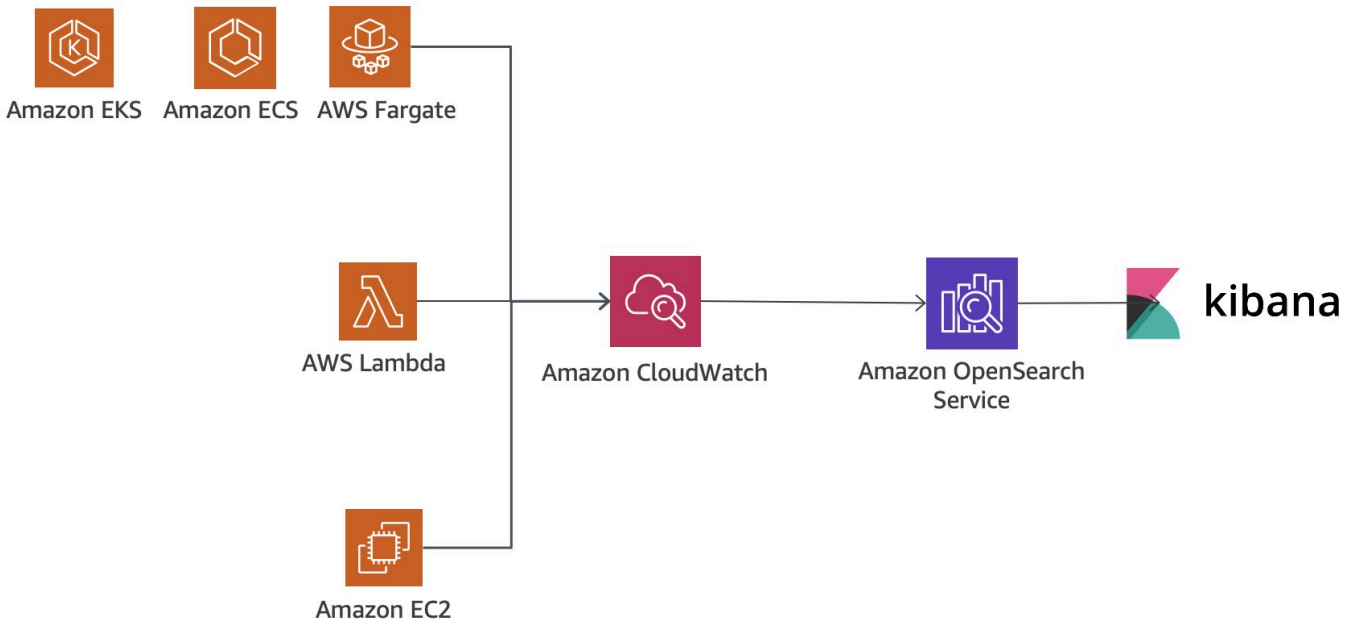

図 14: Amazon OpenSearch Service を使用したログ分析

分析のその他のオプション

詳細なログ分析のために、フルマネージドデータウェアハウスサービスである Amazon Redshift と、スケーラブルなビジネスインテリジェンスサービスである Quick Suite は、効果的なソリューションを提供します。QuickSight は、Redshift、RDS、Aurora、EMR、DynamoDB、Amazon S3、Kinesis などのさまざまな AWS データサービスへの簡単な接続を提供し、データアクセスを簡素化します。

CloudWatch Logs は、リアルタイムストリーミングデータを配信するサービスである Amazon Data Firehose にログエントリをストリーミングできます。次に、QuickSight は Redshift に保存されているデータを使用して、包括的な分析、レポート、視覚化を行います。

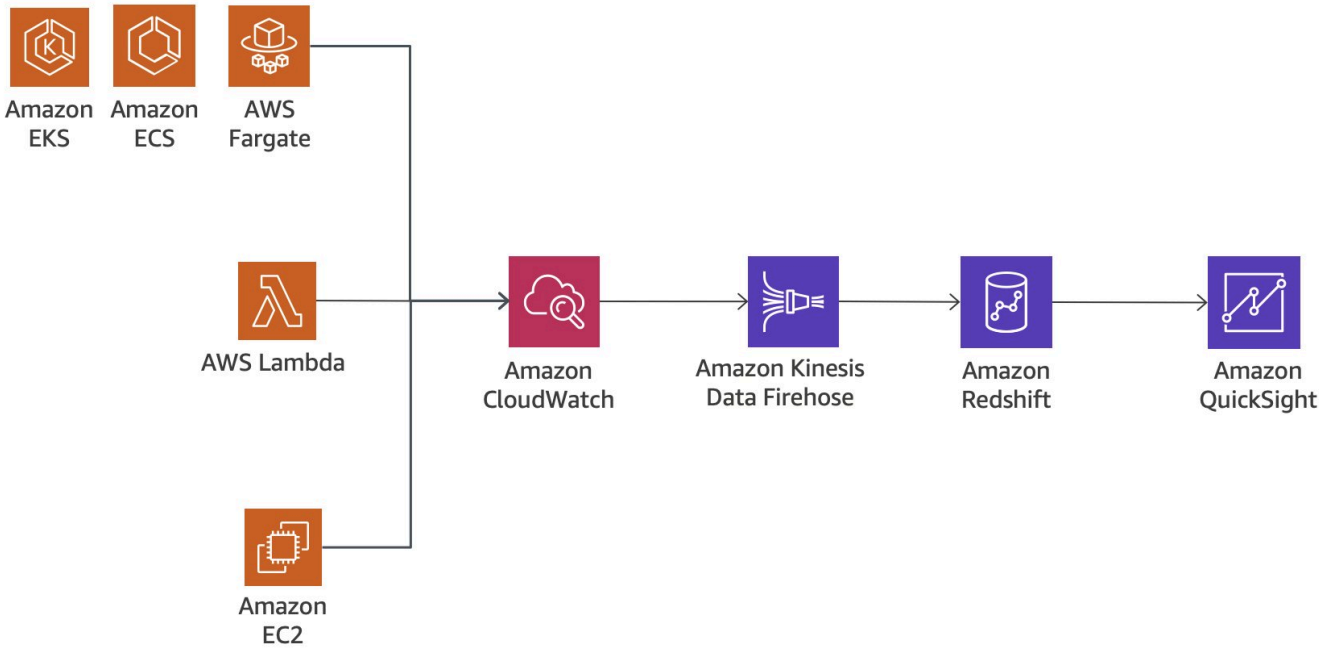

図 15: Amazon Redshift と Quick Suite を使用したログ分析

さらに、ログがオブジェクトストレージサービスである S3 バケットに保存されると、クラウドベースのビッグデータプラットフォームである Redshift や EMR などのサービスにデータをロードできるため、保存されたログデータの詳細な分析が可能になります。

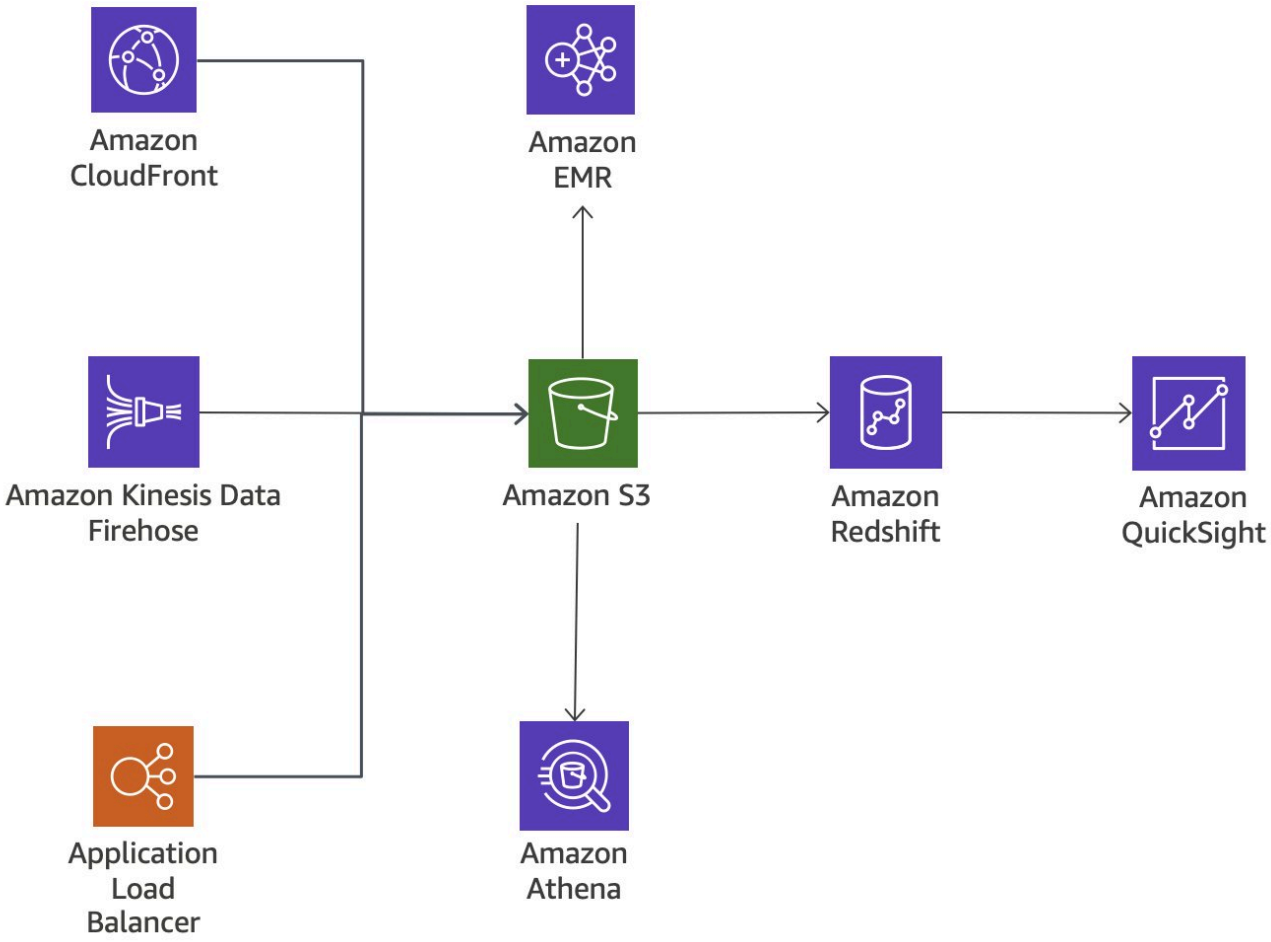

図 16: ログ分析の合理化: AWS サービスから QuickSight へ

マイクロサービス通信でのチャットの管理

チャット性とは、マイクロサービス間の過剰な通信を指します。これにより、ネットワークレイテンシーの増加による非効率が生じる可能性があります。適切に機能するシステムのために、チャットを効果的に管理することが不可欠です。

チャットを管理するための主要なツールには、REST APIs、HTTP APIs、gRPC APIs。REST APIs は、API キー、クライアントごとのスロットリング、リクエストの検証、 AWS WAF 統合、プライベート API エンドポイントなど、さまざまな高度な機能を提供します。HTTP APIsは最小限の機能で設計されているため、低価格で利用できます。このトピックと、REST APIs および HTTP APIs「REST APIs と HTTP APIs」を参照してください。

多くの場合、マイクロサービスは広く使用されているため、通信に HTTP 経由の REST を使用します。ただし、大量の状況では、REST のオーバーヘッドによってパフォーマンスの問題が発生する可能性があります。これは、通信が新しいリクエストごとに必要な TCP ハンドシェイクを使用するためです。このような場合は、gRPC API の方が適しています。gRPC は、単一の TCP 接続で複数のリクエストを許可するため、レイテンシーを短縮します。gRPC は双方向ストリーミングもサポートしているため、クライアントとサーバーは同時にメッセージを送受信できます。これにより、特に大規模またはリアルタイムのデータ転送の場合、通信がより効率的になります。

適切な API タイプを選択してもチャットが続く場合は、マイクロサービスアーキテクチャの再評価が必要になる場合があります。サービスを統合したり、ドメインモデルを改訂したりすると、喧嘩が軽減され、効率が向上します。

プロトコルとキャッシュの使用

マイクロサービスでは、通信に gRPC や REST などのプロトコルがよく使用されます (「」の前のセクションを参照) コミュニケーションメカニズム。gRPC では転送に HTTP/2 が使用され、REST では通常 HTTP/1.1 が使用されます。gRPC ではシリアル化にプロトコルバッファが使用され、REST では通常 JSON または XML が使用されます。レイテンシーと通信オーバーヘッドを減らすために、キャッシュを適用できます。Amazon ElastiCache や API Gateway のキャッシュレイヤーなどのサービスは、マイクロサービス間の呼び出し数を減らすのに役立ちます。

監査

マイクロサービスアーキテクチャでは、すべての サービスにわたるユーザーアクションを可視化することが重要です。 は AWS CloudTrail、 で行われたすべての API コールをログに記録する や AWS、アプリケーションログをキャプチャするために使用される AWS CloudWatch などのツール AWS を提供します。これにより、マイクロサービス全体の変更を追跡し、動作を分析できます。Amazon EventBridge は、システムの変更に迅速に対応し、適切なユーザーに通知したり、問題を解決するためのワークフローを自動的に開始したりできます。

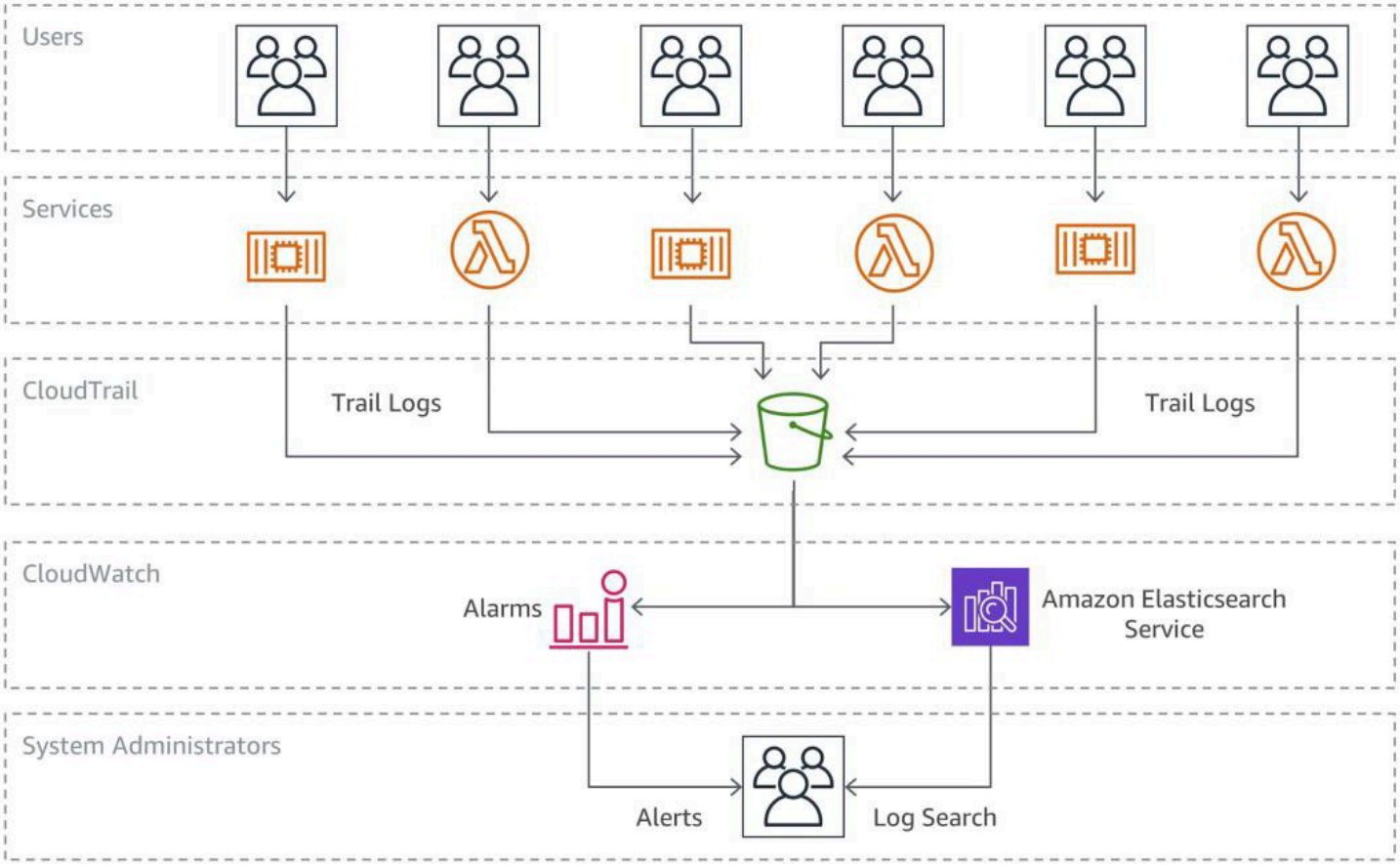

図 17: マイクロサービス全体の監査と修復

リソースインベントリと変更管理

急速に進化するインフラストラクチャ設定を持つアジャイル開発環境では、自動監査と制御が不可欠です。マイクロサービス全体でこれらの変更をモニタリングするためのマネージド型アプローチ AWS Config ルール を提供します。これにより、ポリシー違反を自動的に検出、追跡、アラートを送信する特定のセキュリティポリシーの定義が可能になります。

例えば、マイクロサービスの API Gateway 設定が HTTPS リクエストだけでなくインバウンド HTTP トラフィックを受け入れるように変更された場合、事前定義された AWS Config ルールがこのセキュリティ違反を検出できます。監査の変更を記録し、SNS 通知をトリガーして、準拠状態を復元します。

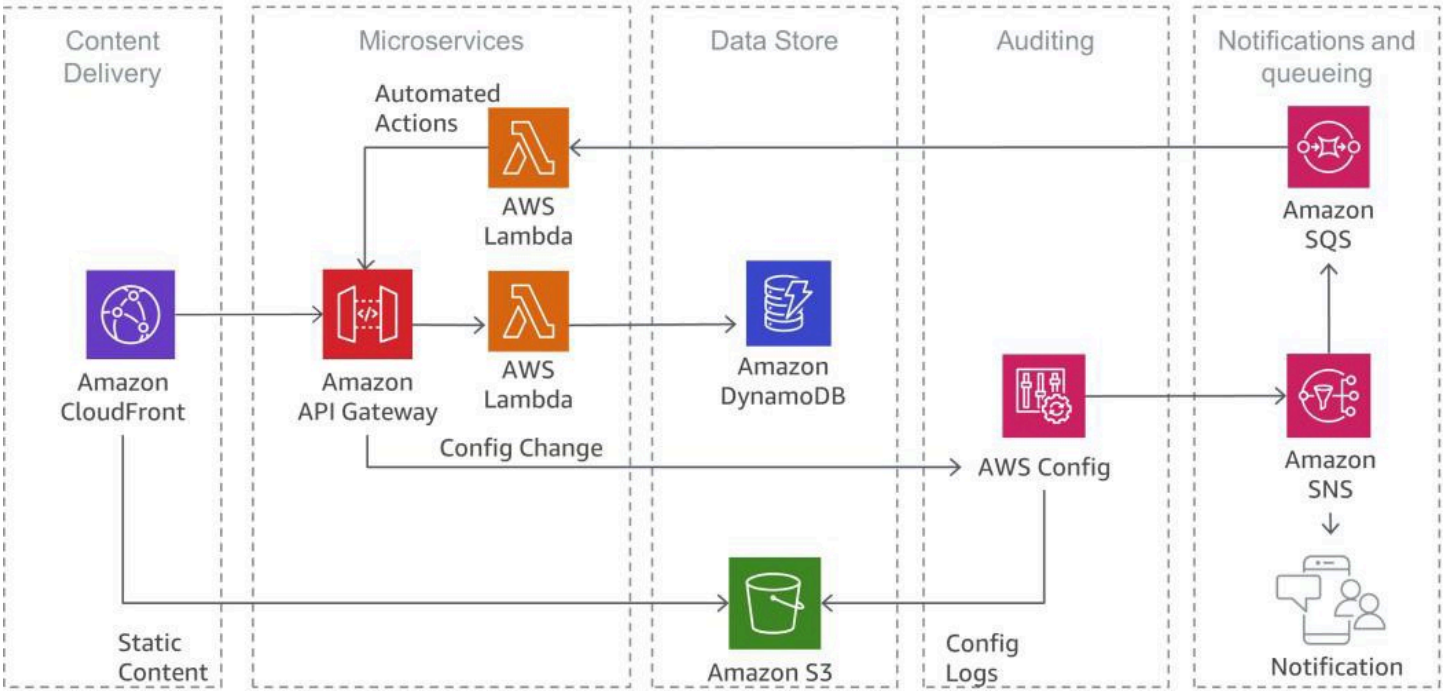

図 18: を使用したセキュリティ違反の検出 AWS Config

結論

マイクロサービスアーキテクチャは、従来のモノリシックシステムに代わる汎用性の高い設計アプローチであり、アプリケーションのスケーリング、開発速度の向上、組織成長の促進に役立ちます。その適応性により、コンテナ、サーバーレスアプローチ、または 2 つの組み合わせを使用して実装し、特定のニーズに合わせて調整できます。

ただし、これは one-size-fits-allソリューションではありません。各ユースケースでは、アーキテクチャの複雑さと運用上の需要が増加する可能性があるため、細心の注意を払った評価が必要です。しかし、戦略的にアプローチすると、マイクロサービスの利点はこれらの課題を大幅に上回ります。重要なのは、特にオブザーバビリティ、セキュリティ、変更管理の分野において、プロアクティブプランニングです。

また、マイクロサービス以外にも、検索拡張生成 (RAG) などの生成 AI アーキテクチャなど、まったく異なるアーキテクチャフレームワークがあり、ニーズに最適な幅広いオプションを提供することに注意してください。

AWSは、堅牢なマネージドサービススイートにより、チームが効率的なマイクロサービスアーキテクチャを構築し、複雑さを効果的に最小限に抑えることができます。このホワイトペーパーは、関連する AWS サービスとキーパターンの実装をガイドすることを目的としています。目標は、マイクロサービスの能力を活用するための知識を身に付け、マイクロサービスの利点を活用し AWS、アプリケーション開発ジャーニーを変革できるようにすることです。

寄稿者

このドキュメントには、次の個人および組織が貢献しました。

- Amazon Web Services、ソリューションアーキテクチャ、Sascha Möllering
- Amazon Web Services、ソリューションアーキテクチャ、Müller
- Matthias Jung、Amazon Web Services、ソリューションアーキテクチャ
- アマゾン ウェブ サービス、 ソリューションアーキテクチャ、Private Dalbhanjan
- アマゾン ウェブ サービス、 ソリューションアーキテクチャ、David Chapman
- アマゾン ウェブ サービス、ソリューションアーキテクチャ、Christoph Kassen
- Amazon Web Services、ソリューションアーキテクチャ、Umair Ishaq
- Rajiv Kumar、Amazon Web Services、ソリューションアーキテクチャ
- Amazon Web Services、ソリューションアーキテクチャ、Ramesh Dwarakanath
- Amazon Web Services、ソリューションアーキテクチャ、Andrew Watkins
- Amazon Web Services、ソリューションアーキテクチャ、Yann Stoneman
- Mainak Chaudhuri、Amazon Web Services、ソリューションアーキテクチャ
- Gaurav Acharya、Amazon Web Services、ソリューションアーキテクチャ

ドキュメント履歴

このホワイトペーパーの更新に関する通知を受け取るには、RSS フィードにサブスクライブしてください。

変更	説明	日付
メジャーな更新	AWS Customer Carbon Footprint Tool、Amazon EventBridge、 (GraphQL) 、 AWS AppSync AWS Lambda Layers、Lambda SnapStart、大規模言語モデル (LLMs)、Amazon Managed Streaming for Apache Kafka (MSK)、Amazon Managed Workflows for Apache Airflow (MWAA)、Amazon VPC Lattice、 AWS AppConfig に関する情報を追加しました。コストの最適化と持続可能性に関する別のセクションを追加しました。	2023 年 7 月 31 日
マイナーな更新	Well-Architected を抽象化に追加しました。	2022 年 4 月 13 日
ホワイトペーパーの更新	Amazon EventBridge、AWS OpenTelemetry、AMP、AMG、Container Insights の統合、マイナーテキストの変更。	2021 年 11 月 9 日
マイナーな更新	ページレイアウトの調整	2021 年 4 月 30 日
マイナーな更新	軽微なテキストの変更。	2019 年 8 月 1 日

ホワイトペーパーの更新	Amazon EKS、AWS Fargate、Amazon MQ、AWS PrivateLink、AWS App Mesh、AWS Cloud Map の統合	2019 年 6 月 1 日
ホワイトペーパーの更新	AWS Step Functions、AWS X-Ray、および ECS イベントストリームの統合。	2017 年 9 月 1 日
初版発行	AWS でのマイクロサービスの実装が公開されました。	2016 年 12 月 1 日

> **Note**
>
> RSS の更新を購読するには、使用しているブラウザで RSS プラグインを有効にする必要があります。

注意

お客様は、本書に記載されている情報を独自に評価する責任を負うものとします。本書は、(a) 情報提供のみを目的としており、(b) 通知なしに変更される可能性がある現在の AWS 製品提供および慣行を表し、(c) AWS およびその関連会社、サプライヤー、または許諾者からのいかなる約束または保証も生じません。 AWS 製品またはサービスは、明示または黙示を問わず、いかなる種類の保証、表明、または条件も伴わず、現状有姿のまま提供されます。顧客 AWS に対する の責任は AWS 契約によって管理され、本書は AWS と顧客との間の契約の一部でも変更もされません。

AWS 用語集

最新の AWS 用語については、「 AWS の用語集 リファレンス」のAWS 「 用語集」を参照してください。

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。