AWS ホワイトペーパー

Amazon API Gateway と AWS Lambda を使用した AWS サーバーレスマルチティアアーキテクチャ

Amazon API Gateway と AWS Lambda を使用した AWS サーバーレスマルチティアアーキテクチャ : AWS ホワイトペーパー

Table of Contents

このホワイトペーパーは過去の参考用です。一部のコンテンツは古く、一部のリンクは使用できない場合があります。

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。

Amazon API Gateway と AWS Lambda を使用した AWS Serverless マルチ階層アーキテクチャ

公開日: 2021 年 10 月 20 日 (ドキュメントの改訂)

要約

このホワイトペーパーでは、Amazon Web Services (AWS) のイノベーションを使用して、多層アーキテクチャの設計方法を変更し、マイクロサービス、モバイルバックエンド、シングルページアプリケーションなどの一般的なパターンを実装する方法を説明します。アーキテクトとデベロッパーは、Amazon API Gateway やその他の のサービスを使用して AWS Lambda、多層アプリケーションの作成と管理に必要な開発と運用のサイクルを削減できます。

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。

Serverless Application Lens 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。

序章

多層アプリケーション (3 層、n 層など) は、数十年にわたって基盤となるアーキテクチャパターンであり、ユーザー向けアプリケーションの一般的なパターンであり続けています。多層アーキテクチャの説明に使用される言語は異なりますが、多層アプリケーションは通常、次のコンポーネントで構成されます。

- プレゼンテーション階層: ユーザーが直接操作するコンポーネント (ウェブページやモバイルアプリ UIs など) 。
- ロジック層: ユーザーアクションをアプリケーション機能 (CRUD データベースオペレーションやデータ処理など) に変換するために必要なコード。
- データ層: アプリケーションに関連するデータを保持するストレージメディア (データベース、オブジェクトストア、キャッシュ、ファイルシステムなど) 。

多層アーキテクチャパターンは、分離され、個別にスケーラブルなアプリケーションコンポーネントを個別に開発、管理、保守できる一般的なフレームワークを提供します (多くの場合、異なるチーム) 。

このパターンの結果として、ネットワーク (階層は別の階層とやり取りするためにネットワーク呼び出しを実行する必要があります) が階層間の境界として機能するため、多層アプリケーションを開発するには、多くの場合、多くの差別化されていないアプリケーションコンポーネントを作成する必要があります。これらのコンポーネントには、次のようなものがあります。

- 階層間の通信用のメッセージキューを定義するコード
- アプリケーションプログラミングインターフェイス (API) とデータモデルを定義するコード
- アプリケーションへの適切なアクセスを保証するセキュリティ関連のコード

これらの例はすべて「定型」コンポーネントと見なすことができ、多層アプリケーションでは必要ですが、実装がアプリケーションごとに大きく異なることはありません。

AWS は、サーバーレス多層アプリケーションの作成を可能にする多くのサービスを提供します。これにより、このようなアプリケーションを本番環境にデプロイするプロセスが大幅に簡素化され、従来のサーバー管理に関連するオーバーヘッドが排除されます。API を作成および管理するためのサービスである Amazon API Gateway、および任意のコード関数を実行するためのサービスAWS Lambdaである を組み合わせて、堅牢な多層アプリケーションの作成を簡素化できます。 APIs

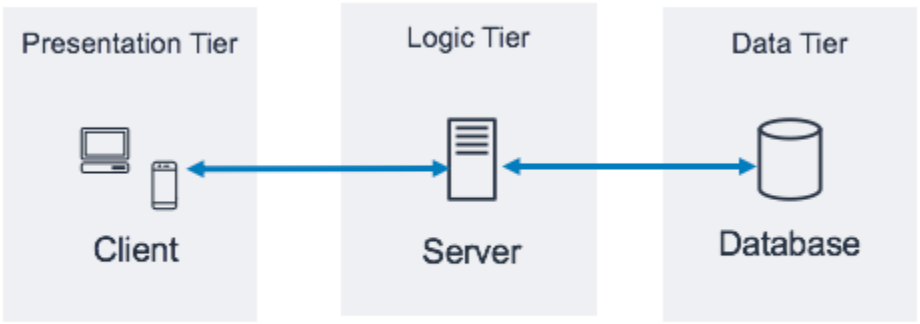
Amazon API Gateway と の統合 AWS Lambda により、ユーザー定義のコード関数を HTTPS リクエストから直接開始できます。リクエストボリュームに関係なく、API Gateway と Lambda の両方がアプリケーションのニーズを正確にサポートするように自動的にスケーリングされます (スケーラビリティ情報については、API Gateway Amazon API Gateway のクォータと重要な注意事項を参照してください)。これらの 2 つのサービスを組み合わせることで、アプリケーションにとって重要なコードのみを記述し、高可用性のための設計、クライアント SDKs の記述、サーバーとオペレーティングシステム (OS) の管理、スケーリング、クライアント認可メカニズムの実装など、多層アーキテクチャの実装に関する他のさまざまな差別化されていない側面に焦点を当てないようにする階層を作成できます。

API Gateway と Lambda を使用すると、サーバーレスロジック層を作成できます。アプリケーションの要件に応じて、AWS にはサーバーレスプレゼンテーション階層 (Amazon CloudFront や Amazon Simple Storage Service など) とデータ階層 (Amazon Aurora、Amazon DynamoDB など) を作成するオプションもあります。

このホワイトペーパーでは、多層アーキテクチャの最も一般的な例である 3 層ウェブアプリケーションに焦点を当てています。ただし、この多層パターンは、一般的な 3 層ウェブアプリケーションを大幅に超えて適用できます。

3 層アーキテクチャの概要

3 層アーキテクチャは、多層アーキテクチャの最も一般的な実装であり、単一のプレゼンテーション層、ロジック層、データ層で構成されます。次の図は、シンプルで汎用的な 3 層アプリケーションの例を示しています。



3 層アプリケーションのアーキテクチャパターン

一般的な 3 層アーキテクチャパターンの詳細については、多くの優れたオンラインリソースを参照してください。このホワイトペーパーでは、Amazon API Gateway と を使用したこのアーキテクチャの特定の実装パターンに焦点を当てています AWS Lambda。

サーバーレスロジック層

3 層アーキテクチャのロジック階層は、アプリケーションの頭脳を表します。ここで Amazon API Gateway と を使用すると、従来のサーバーベースの実装と比較して最も大きな影響を与える AWS Lambda 可能性があります。これらの 2 つのサービスの機能により、可用性、スケーラビリティ、安全性に優れたサーバーレスアプリケーションを構築できます。従来のモデルでは、アプリケーションに何千ものサーバーが必要になる場合がありますが、Amazon API Gateway を使用することで、AWS Lambda お客様はいかなる容量のサーバー管理にも責任を負いません。さらに、これらのマネージドサービスを一緒に使用することで、次の利点が得られます。

- AWS Lambda:
  - 選択、保護、パッチ適用、または管理する OS がない
  - 適切なサイズ、モニタリング、またはスケーリングを行うサーバーがない
  - 過剰プロビジョニングによるコストの削減
  - プロビジョニング不足によるパフォーマンスへのリスクを軽減

- Amazon API Gateway:
  - APIs をデプロイ、モニタリング、保護するメカニズムの簡素化
  - キャッシュとコンテンツ配信による API パフォーマンスの向上

AWS Lambda

AWS Lambda は、サーバーのプロビジョニング、管理、スケーリングを行わずに任意のコード関数を実行できるようにするコンピューティングサービスです。サポートされている言語には、Python、Ruby、Java、Go、.NET などがあります。Lambda 関数は、管理された分離されたコンテナで実行され、イベントソースと呼ばれる、 が使用可能 AWS にするいくつかのプログラムによるトリガーの 1 つであるイベントに応答して起動されます。サポートされている言語とイベントソースの詳細については、「Lambda のFAQs」を参照してください。

Lambda の多くの一般的なユースケースは、Amazon S3 に保存されているファイルの処理や Amazon Amazon Kinesisからのデータレコードのストリーミングなど、イベント駆動型のデータ処理ワークフローを中心に展開されています。Amazon API Gateway と組み合わせて使用すると、Lambda 関数は一般的なウェブサービスの機能を実行します。クライアント HTTPS リクエストに応答してコードを開始します。API Gateway はロジック階層のフロントドアとして機能し、アプリケーションコードを AWS Lambda 呼び出します。

ビジネスロジックはここにあり、サーバーは必要ありません

Lambda では、 イベントによって開始されたときに実行されるハンドラーと呼ばれるコード関数を記述する必要があります。API Gateway で Lambda を使用するには、API への HTTPS リクエストが発生したときにハンドラー関数を起動するように API Gateway を設定できます。サーバーレス多層アーキテクチャでは、APIs Gateway で作成する各 API は、必要なビジネスロジックを呼び出す Lambda 関数 (および内部のハンドラー) と統合されます。

AWS Lambda 関数を使用してロジック層を構成すると、アプリケーション機能 (API ごとに 1 つの Lambda 関数、または API メソッドごとに 1 つの Lambda 関数) を公開するために必要な粒度レベルを定義できます。Lambda 関数内では、ハンドラーは他の依存関係 (コード、ライブラリ、ネイティブバイナリ、外部ウェブサービスでアップロードした他のメソッドなど) や、他の Lambda 関数に連絡できます。

Lambda 関数を作成または更新するには、コードを zip ファイル内の Lambda デプロイパッケージとして Amazon S3 バケットにアップロードするか、すべての依存関係とともにコンテナイメージとしてパッケージ化する必要があります。関数は、AWS マネジメントコンソール、 の実行 AWS Command Line Interface (AWS CLI) 、インフラストラクチャの実行などのさまざまなデプロイ方法をコードテンプレートまたは CloudFormation、 AWS Serverless Application Model (AWS SAM) 、などのフレームワークとして使用できますAWS Cloud Development Kit (AWS CDK)。これらのメソッドのいずれかを使用して関数を作成するときは、デプロイパッケージ内のどのメソッドがリクエストハンドラーとして機能するかを指定します。同じデプロイパッケージを複数の Lambda 関数定義に再利用できます。各 Lambda 関数は、同じデプロイパッケージ内に一意のハンドラーを持つ場合があります。

Lambda セキュリティ

Lambda 関数を実行するには、 AWS Identity and Access Management (IAM) ポリシーで許可されているイベントまたはサービスによって呼び出される必要があります。IAM ポリシーを使用すると、定義した API Gateway リソースによって呼び出されない限り、まったく開始できない Lambda 関数を作成できます。このようなポリシーは、さまざまな AWS サービスでリソースベースのポリシーを使用して定義できます。

各 Lambda 関数は、Lambda 関数がデプロイされたときに割り当てられた IAM ロールを引き受けます。この IAM ロールは、Lambda 関数が操作できる他の AWS サービスとリソース (Amazon DynamoDB Amazon S3 など) を定義します。Lambda 関数では、これは実行ロールと呼ばれます。

機密情報を Lambda 関数内に保存しないでください。IAM は Lambda 実行ロールを介して AWS サービスへのアクセスを処理します。Lambda 関数内から他の認証情報 (データベース認証情報や

API キーなど) にアクセスする必要がある場合は、環境変数で AWS Key Management Service (AWS KMS) を使用するか、AWSSecrets Manager などのサービスを使用して、使用していないときにこの情報を安全に保つことができます。

大規模なパフォーマンス

Amazon Elastic Container Registry (Amazon ECR) または Amazon S3 にアップロードされた zip ファイルからコンテナイメージとしてプルされたコードは、AWS によって管理される分離された環境で実行されます。Lambda 関数をスケールする必要はありません。イベント通知が関数によって受信されるたびに、使用可能な容量がコンピューティングフリート内に AWS Lambda 配置され、定義したランタイム、メモリ、ディスク、タイムアウト設定でコードが実行されます。このパターンを使用すると、AWS は関数のコピーを必要な数だけ開始できます。

Lambda ベースのロジック階層は、常に顧客のニーズに適したサイズです。マネージドスケーリングと同時コード開始によるトラフィックの急増を、Lambda pay-per-useと組み合わせてすばやく吸収できるため、アイドル状態のコンピューティング容量に支払うことなく、常に顧客のリクエストを満たすことができます。

サーバーレスのデプロイと管理

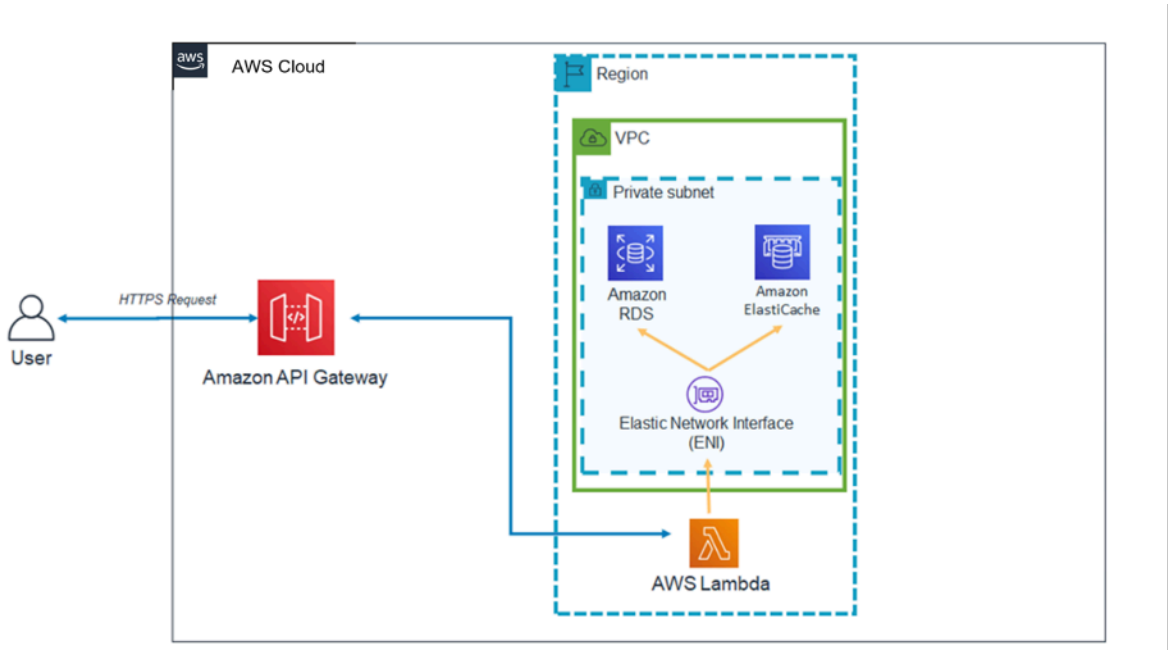
Lambda 関数のデプロイと管理に役立つように、以下を含むオープンソースフレームワークである AWS Serverless Application Model (AWS SAM) を使用します。

- AWS SAM テンプレート仕様 - 関数を定義し、それらの環境、アクセス許可、設定、イベントを記述して、アップロードとデプロイを簡素化するために使用される構文。
- AWS SAM CLI - SAM テンプレート構文の検証、関数のローカル呼び出し、Lambda 関数のデバッグ、パッケージ関数のデプロイを可能にするコマンド。

プログラミング言語を使用してクラウドインフラストラクチャを定義し AWS CDK、CloudFormation を通じてプロビジョニングするためのソフトウェア開発フレームワークである を使用することもできます。CDK はリソースを定義する必須の方法を提供しますが AWS 、 は宣言的な方法 AWS SAM を提供します。

通常、Lambda 関数をデプロイすると、割り当てられた IAM ロールで定義されたアクセス許可で呼び出され、インターネット向けエンドポイントに到達できます。ロジック層の中核として、 AWS Lambda はデータ層と直接統合されるコンポーネントです。データ層に機密性の高いビジネス情報またはユーザー情報が含まれている場合は、このデータ層が適切に分離されていることを確認することが重要です (プライベートサブネット内) 。

Lambda 関数がプライベートデータベースインスタンスのようにパブリックに公開できないリソースにアクセスできるようにする場合は、 AWS アカウントの Virtual Private Cloud (VPC) のプライベートサブネットに接続するように Lambda 関数を設定できます。関数を VPC に接続すると、Lambda は関数の VPC 設定内のサブネットごとに Elastic Network Interface を作成し、Elastic Network Interface を使用して内部リソースにプライベートにアクセスします。



VPC 内の Lambda アーキテクチャパターン

VPC で Lambda を使用すると、ビジネスロジックが依存するデータベースやその他のストレージメディアにインターネット経由でアクセスできなくなります。また、VPC は、インターネットからデータを操作する唯一の方法が、定義した APIs と、記述した Lambda コード関数を使用することです。

Amazon API Gateway

Amazon API Gateway は、開発者があらゆる規模で APIs を作成、公開、保守、モニタリング、保護できるようにするフルマネージドサービスです。

クライアント (プレゼンテーション層) は、標準の HTTPS リクエストを使用して APIs Gateway を介して公開される API と統合されます。APIs Gateway を介してサービス指向の多層アーキテクチャに公開される API の適用可能性は、個々のアプリケーション機能を分離し、REST エンドポイントを介してこの機能を公開する能力です。Amazon API Gateway には、ロジック層に強力な機能を追加できる特定の機能と品質があります。

との統合 AWS Lambda

Amazon API Gateway は、REST と HTTP の両方のタイプの APIs をサポートしています。API Gateway API は、リソースとメソッドで構成されます。リソースは、アプリケーションがリソースパス (など) を介してアクセスできる論理エンティティです`/tickets`。メソッドは、API リソース (など) に送信される API リクエストに対応しますGET `/tickets`。API Gateway を使用すると、Lambda 関数を使用して各メソッドをバックアップできます。つまり、API Gateway で公開されている HTTPS エンドポイントを介して API を呼び出すと、API Gateway は Lambda 関数を呼び出します。

API Gateway と Lambda 関数は、プロキシ統合と非プロキシ統合を使用して接続できます。

プロキシ統合

プロキシ統合では、クライアント HTTPS リクエスト全体がそのまま Lambda 関数に送信されます。API Gateway はクライアントリクエスト全体を Lambda ハンドラー関数のイベントパラメータとして渡し、Lambda 関数の出力はクライアントに直接返されます (ステータスコード、ヘッダーなどを含む) 。

非プロキシ統合

非プロキシ統合では、クライアントリクエストのパラメータ、ヘッダー、本文を Lambda ハンドラー関数のイベントパラメータに渡す方法を設定します。さらに、Lambda 出力をユーザーに変換する方法を設定します。

> **Note**
>
> API Gateway は、モック統合 (初期アプリケーション開発に有用) や S3 オブジェクトへの直接プロキシなど AWS Lambda、外部の追加のサーバーレスリソースにプロキシすることもできます。

リージョン間で安定した API パフォーマンス

Amazon API Gateway の各デプロイには、内部に Amazon CloudFront ディストリビューションが含まれます。CloudFront は、Amazon のエッジロケーションのグローバルネットワークを API を使用するクライアントの接続ポイントとして使用するコンテンツ配信サービスです。これにより、API の応答レイテンシーが短縮されます。世界中の複数のエッジロケーションを使用することで、Amazon

CloudFront は分散型サービス拒否 (DDoS) 攻撃シナリオに対抗する機能も提供します。詳細については、AWS Best Practices for DDoS Resiliency ホワイトペーパーを参照してください。

API Gateway を使用してレスポンスをオプションのインメモリキャッシュに保存することで、特定の API リクエストのパフォーマンスを向上させることができます。このアプローチは、繰り返される API リクエストのパフォーマンス上の利点を提供するだけでなく、Lambda 関数が呼び出される回数を減らし、全体的なコストを削減できます。

組み込み機能でイノベーションを奨励し、オーバーヘッドを削減する

新しいアプリケーションを構築するための開発コストは投資です。API Gateway を使用すると、特定の開発タスクに必要な時間を短縮し、合計開発コストを削減できるため、組織はより自由に実験やイノベーションを行うことができます。

初期アプリケーション開発フェーズでは、ロギングとメトリクス収集の実装は、新しいアプリケーションをより迅速に提供することを無視されることがよくあります。これにより、これらの機能を本番環境で実行されているアプリケーションにデプロイする際に、技術的な負債や運用上のリスクが発生する可能性があります。Amazon API Gateway は、API Gateway から raw データを収集して処理する Amazon CloudWatch とシームレスに統合され、API 実行をモニタリングするための読み取り可能なほぼリアルタイムのメトリクスになります。API Gateway は、設定可能なレポートによるアクセスログ記録と、デバッグのためのAWS X-Rayトレースもサポートしています。これらの各機能はコードを記述する必要はなく、コアビジネスロジックのリスクなしに本番環境で実行されているアプリケーションで調整できます。

アプリケーションの全体的な有効期間が不明な場合や、有効期間が短いことがわかっている場合があります。このようなアプリケーションを構築するためのビジネスケースの作成は、開始点に API Gateway が提供するマネージド機能がすでに含まれている場合や、API がリクエストの受信 APIs を開始した後にインフラストラクチャコストが発生する場合にのみ、簡単に行うことができます。詳細については、Amazon API Gateway の料金」を参照してください。

迅速に反復処理し、俊敏性を維持する

Amazon API Gateway と AWS Lambda を使用して API のロジック層を構築すると、API のデプロイとバージョン管理を簡素化することで、ユーザーベースの変化する需要にすばやく適応できます。

ステージデプロイ

API Gateway に API をデプロイするときは、デプロイを API Gateway ステージに関連付ける必要があります。各ステージは API のスナップショットであり、クライアントアプリケーションが呼び出

すことができます。この規則を使用すると、開発、テスト、ステージング、または本番ステージにアプリケーションを簡単にデプロイし、ステージ間でデプロイを移動できます。API をステージにデプロイするたびに、必要に応じて元に戻すことができる別のバージョンの API を作成します。これらの機能により、新しい機能が別の API バージョンとしてリリースされている間も、既存の機能とクライアントの依存関係をそのまま継続できます。

Lambda との分離された統合

API Gateway と Lambda 関数の API の統合は、API Gateway ステージ変数と Lambda 関数エイリアスを使用してデカップリングできます。これにより、API デプロイが簡素化され、高速化されます。API で Lambda 関数名またはエイリアスを直接設定する代わりに、Lambda 関数の特定のエイリアスを指すことができる API でステージ変数を設定できます。デプロイ中に、Lambda 関数エイリアスを指すようにステージ変数値を変更すると、API は特定のステージの Lambda エイリアスの背後にある Lambda 関数バージョンを実行します。

Canary リリースのデプロイ

Canary リリースは、テスト目的で API の新しいバージョンがデプロイされ、ベースバージョンが同じステージで通常のオペレーションの本番リリースとしてデプロイされたままであるソフトウェア開発戦略です。Canary リリースのデプロイでは、API トラフィックの合計が本番リリースと Canary リリースにランダムに分割され、比率が事前設定されています。API Gateway APIs は、Canary リリースデプロイ用に設定して、限られたユーザーセットで新機能をテストできます。

カスタムドメイン名

API Gateway が提供する URL の代わりに、直感的なビジネスフレンドリーな URL 名を API に提供できます。API Gateway には、APIs。カスタムドメイン名を使用すると、API のホスト名を設定し、複数レベルのベースパス (、`myservice/cat/v1`、 など`myservice/dog/v2`) を選択して、代替 URL `myservice`を API にマッピングできます。

API セキュリティの優先順位付け

すべてのアプリケーションは、承認されたクライアントのみが API リソースにアクセスできることを確認する必要があります。多層アプリケーションを設計する場合、Amazon API Gateway がロジック階層の保護に寄与するいくつかのさまざまな方法を活用できます。

トランジットセキュリティ

APIs へのすべてのリクエストは、転送中の暗号化を有効にするために HTTPS を介して行うことができます。

API Gateway は組み込みの SSL/TLS 証明書を提供します。パブリック API にカスタムドメイン名オプションを使用する場合は APIs 、AWS Certificate Manager を使用して独自の SSL/TLS 証明書を提供できます。API Gateway は、相互 TLS (mTLS) 認証もサポートしています。相互 TLS は API のセキュリティを強化し、クライアントスプーフィングや中間man-in-the攻撃などの攻撃からデータを保護するのに役立ちます。

API 認可

API の一部として作成する各リソースとメソッドの組み合わせには、 AWS Identity and Access Management (IAM) ポリシーで参照できる一意の Amazon リソースネーム (ARN) が付与されます。

API Gateway で API に認可を追加する一般的な方法は 3 つあります。

- IAM ロールとポリシー： クライアントは API アクセスに AWS 署名バージョン 4 (SigV4) 認可と IAM ポリシーを使用します。同じ認証情報を使用すると、必要に応じて他の AWS のサービスやリソース (Amazon S3 バケットや Amazon DynamoDB テーブルなど) へのアクセスを制限または許可できます。
- Amazon Cognito ユーザープール： クライアントは Amazon Cognito ユーザープールを介してサインインし、リクエストの承認ヘッダーに含まれるトークンを取得します。
- Lambda オーソライザー： ベアラートークン戦略 (OAuth や SAML など) を使用するか、リクエストパラメータを使用してユーザーを識別するカスタム認可スキームを実装する Lambda 関数を定義します。

アクセス制限

API Gateway は、API キーの生成と、これらのキーと設定可能な使用量プランの関連付けをサポートします。CloudWatch を使用して API キーの使用状況をモニタリングできます。

API Gateway は、API の各メソッドのスロットリング、レート制限、バーストレート制限をサポートしています。

プライベート API

API Gateway を使用すると、インターフェイス VPC エンドポイントを使用して、Amazon VPC の仮想プライベートクラウドからのみアクセスできるプライベート REST APIs を作成できます。これは、VPC で作成するエンドポイントネットワークインターフェイスです。

リソースポリシーを使用すると、AWS アカウント全体を含む、選択した VPCsおよび VPC エンドポイントから API へのアクセスを有効または拒否できます。各エンドポイントを使用して複数のプ

ライベート API にアクセスできます。AWS Direct Connect を使用して、オンプレミスネットワークから Amazon VPC に接続を確立し、その接続経由でプライベート API にアクセスすることもできます。

いずれの場合も、プライベート API へのトラフィックは安全な接続を使用し、Amazon のネットワーク外に出ることはありません — パブリックインターネットからは隔離されています。

AWS WAF を使用したファイアウォール保護

インターネット向け APIs は悪意のある攻撃に対して脆弱です。 AWS WAF は、このような攻撃から APIs を保護するのに役立つウェブアプリケーションファイアウォールです。SQL インジェクションやクロスサイトスクリプティング攻撃などの一般的なウェブエクスプロイトから APIs を保護します。API Gateway AWS WAFで を使用すると、APIsの保護に役立ちます。

データ層

を口ジック層 AWS Lambda として使用しても、データ層で使用できるデータストレージオプションは制限されません。Lambda 関数は、Lambda デプロイパッケージに適切なデータベースドライバーを含めて任意のデータストレージオプションに接続し、IAM ロールベースのアクセスまたは暗号化された認証情報 (AWS KMS または AWS Secrets Manager 経由) を使用します。

アプリケーションのデータストアの選択は、アプリケーションの要件に大きく依存します。AWS には、アプリケーションのデータ層を構成するために使用できるサーバーレスおよび非サーバーレスのデータストアが多数用意されています。

サーバーレスデータストレージオプション

Amazon S3 は、業界をリードするスケーラビリティ、データ可用性、セキュリティ、パフォーマンスを提供するオブジェクトストレージサービスです。

Amazon Aurora は、クラウド向けに構築された MySQL 互換および PostgreSQL 互換のリレーショナルデータベースであり、従来のエンタープライズデータベースのパフォーマンスと可用性を、オープンソースデータベースのシンプルさと費用対効果と組み合わせます。Aurora には、サーバーレスモデルと従来の使用モデルの両方が用意されています。

Amazon DynamoDB は、あらゆる規模で 1 桁ミリ秒のパフォーマンスを実現するキーバリューおよびドキュメントデータベースです。これは、フルマネージド、サーバーレス、マルチリージョン、マルチアクティブ、耐久性のあるデータベースで、セキュリティ、バックアップと復元、インターネット規模のアプリケーションのインメモリキャッシュが組み込まれています。

Amazon Timestream は、IoT および運用アプリケーション向けの高速でスケーラブルなフルマネージドの時系列データベースサービスです。リレーショナルデータベースのコストの 10 分の 1 で、1 日あたり数兆のイベントを簡単に保存および分析できます。IoT デバイス、IT システム、スマート産業用マシンの台頭により、時系列データ、つまり時間の経過に伴う変化を測定するデータは、最も急速に成長しているデータ型の 1 つです。

Amazon Quantum Ledger Database (Amazon QLDB) は、完全マネージド型台帳データベースであり、中央の信頼された機関が所有する透過的でイミュータブルな暗号化検証可能なトランザクションログを提供します。Amazon QLDB は、各アプリケーションデータの変更を追跡し、経時的な変更の完全で検証可能な履歴を維持します。

Amazon Keyspaces (Apache Cassandra 用) は、スケーラブルで可用性が高く、マネージド型の Apache Cassandra 互換データベースサービスです。Amazon Keyspaces では、現在使用している

AWS のと同じ Cassandra アプリケーションコードと開発者ツールを使用して、 で Cassandra ワークロードを実行できます。サーバーのプロビジョニング、パッチ適用、管理を行う必要はなく、ソフトウェアのインストール、保守、運用を行う必要もありません。Amazon Keyspaces はサーバーレスであるため、使用したリソースに対してのみ料金が発生し、サービスはアプリケーショントラフィックに応じてテーブルを自動的にスケールアップまたはスケールダウンできます。

Amazon Elastic File System (Amazon EFS) は、ストレージをプロビジョニングまたは管理することなくファイルデータを共有できる、シンプルでサーバーレス、set-and-forgetの伸縮自在なファイルシステムを提供します。AWS クラウドサービスやオンプレミスリソースで使用でき、アプリケーションを中断することなくオンデマンドでペタバイトにスケールするように構築されています。Amazon EFS を使用すると、ファイルを追加および削除するたびにファイルシステムを自動的に拡張および縮小できるため、増大に対応するために容量をプロビジョニングおよび管理する必要がなくなります。Amazon EFS は Lambda 関数でマウントできるため、APIs。

非サーバーレスデータストレージオプション

Amazon Relational Database Service (Amazon RDS) は、使用可能なエンジン (Amazon Aurora、PostgreSQL、MySQL、MariaDB、Oracle、Microsoft SQL Server) のいずれかを使用し、メモリ、パフォーマンス、または I/O 用に最適化された複数の異なるデータベースインスタンスタイプで実行することで、リレーショナルデータベースの設定、運用、スケーリングを容易にするマネージドウェブサービスです。

Amazon Redshift は、クラウド内のフルマネージド型のペタバイト規模のデータウェアハウスサービスです。

Amazon ElastiCache は、Redis または Memcached のフルマネージドデプロイです。一般的なオープンソース互換インメモリデータストアをシームレスにデプロイ、実行、スケーリングします。

Amazon Neptune は、高速で信頼性の高いフルマネージド型のグラフデータベースサービスであり、高度に接続されたデータセットで動作するアプリケーションの構築と実行を容易にします。Neptune は、プロパティグラフと W3C リソース記述フレームワーク (RDF) などの一般的なグラフモデルと、それぞれのクエリ言語をサポートしているため、高度に接続されたデータセットを効率的にナビゲートするクエリを簡単に構築できます。

Amazon DocumentDB (MongoDB 互換) は、MongoDB ワークロードをサポートする高速でスケーラブル、可用性が高く、フルマネージド型のドキュメントデータベースサービスです。

最後に、Amazon EC2 で個別に実行されているデータストアを多層アプリケーションのデータ層として使用することもできます。

プレゼンテーション階層

プレゼンテーション階層は、インターネット経由で公開されている API Gateway REST エンドポイントを介してロジック階層を操作する責任があります。HTTPS 対応クライアントまたはデバイスは、これらのエンドポイントと通信できるため、プレゼンテーション層にさまざまな形式 (デスクトップアプリケーション、モバイルアプリ、ウェブページ、IoT デバイスなど) を柔軟に取り込むことができます。要件に応じて、プレゼンテーション階層は次の AWS サーバーレスサービスを使用できます。

- Amazon Cognito - ユーザーサインアップ、サインイン、アクセスコントロールをウェブおよびモバイルアプリに迅速かつ効率的に追加できるサーバーレスユーザーアイデンティティおよびデータ同期サービス。Amazon Cognito は数百万のユーザーにスケールし、Facebook、Google、Amazon などのソーシャル ID プロバイダー、および SAML 2.0 を介したエンタープライズ ID プロバイダーとのサインインをサポートします。
- CloudFront を使用した Amazon S3 - ウェブサーバーをプロビジョニングすることなく、S3 バケットから直接、シングルページアプリケーションなどの静的ウェブサイトを提供できます。CloudFront をマネージドコンテンツ配信ネットワーク (CDN) として使用してパフォーマンスを向上させ、マネージド証明書またはカスタム証明書を使用して SSL/TL を有効にできます。

AWS Amplify は、フロントエンドのウェブデベロッパーやモバイルデベロッパーがスケーラブルなフルスタックアプリケーションを構築するのに役立つ、組み合わせて、または単独で使用できる一連のツールとサービスです AWS。Amplify は、静的ウェブアプリケーションをグローバルにデプロイおよびホストするためのフルマネージドサービスを提供します。Amazon の信頼性の高い CDN は、世界中の数百のプレゼンスポイントを持ち、アプリケーションのリリースサイクルを加速する組み込み CI/CD ワークフローを備えています。Amplify は、JavaScript、React、Angular、Vue、Next.js などの一般的なウェブフレームワークと、Android、iOS、React Native、Ionic、Flutter などのモバイルプラットフォームをサポートしています。ネットワーク設定とアプリケーション要件によっては、API Gateway APIsを Cross-Origin Resource Sharing (CORS) に準拠させる必要がある場合があります。CORS コンプライアンスにより、ウェブブラウザは静的ウェブページ内から直接 APIs を呼び出すことができます。

CloudFront を使用してウェブサイトをデプロイすると、アプリケーションに到達するための CloudFront ドメイン名が提供されます (例: `d2d47p2vcczkh2.cloudfront.net`)。Amazon Route 53 を使用してドメイン名を登録して CloudFront ディストリビューションに誘導したり、既に所有しているドメイン名を CloudFront ディストリビューションに誘導したりできます。これにより、ユーザーは使い慣れたドメイン名を使用してサイトにアクセスできます。Route 53 を使用して

カスタムドメイン名を API Gateway ディストリビューションに割り当てることもできます。これにより、ユーザーは使い慣れたドメイン名を使用して APIs を呼び出すことができます。

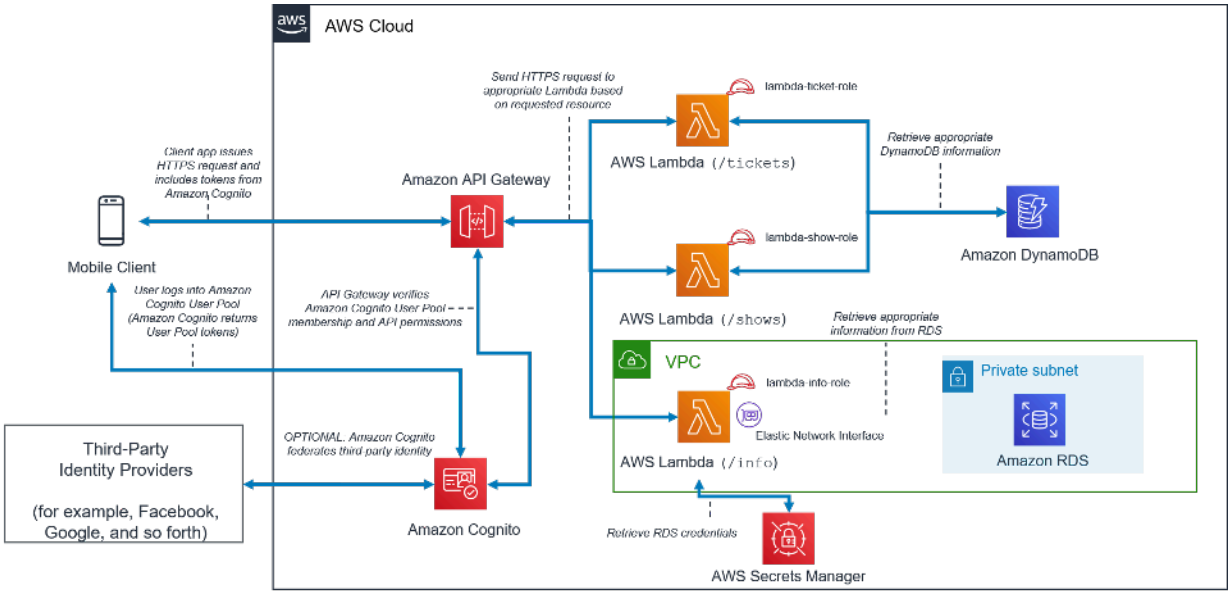
サンプルアーキテクチャパターン

API Gateway と を口ジック層 AWS Lambda として使用して、一般的なアーキテクチャパターンを実装できます。このホワイトペーパーには、 AWS Lambdaベースの口ジック層を活用する最も一般的なアーキテクチャパターンが含まれています。

- モバイルバックエンド - モバイルアプリケーションは API Gateway および Lambda と通信してアプリケーションデータにアクセスします。このパターンは、サーバーレス AWS リソースを使用してプレゼンテーション階層リソース (デスクトップクライアント、EC2 で実行されているウェブサーバーなど) をホストしない汎用 HTTPS クライアントに拡張できます。
- 単一ページアプリケーション - Amazon S3 および CloudFront でホストされている単一ページアプリケーションは、API Gateway と通信し AWS Lambda 、アプリケーションデータにアクセスします。
- ウェブアプリケーション – ウェブアプリケーションは、 と AWS Lambda API Gateway をビジネスロジックに使用する、イベント駆動型の汎用ウェブアプリケーションのバックエンドです。また、DynamoDB をデータベースとして使用し、Amazon Cognito をユーザー管理に使用します。すべての静的コンテンツは Amplify を使用してホストされます。

このホワイトペーパーでは、これら 2 つのパターンに加えて、一般的なマイクロサービスアーキテクチャへの Lambda と API Gateway の適用性について説明します。マイクロサービスアーキテクチャは、標準的な 3 層アーキテクチャではありませんが、アプリケーションコンポーネントを切り離し、相互に通信するステートレスで個々の機能ユニットとしてデプロイする一般的なパターンです。

モバイルバックエンド



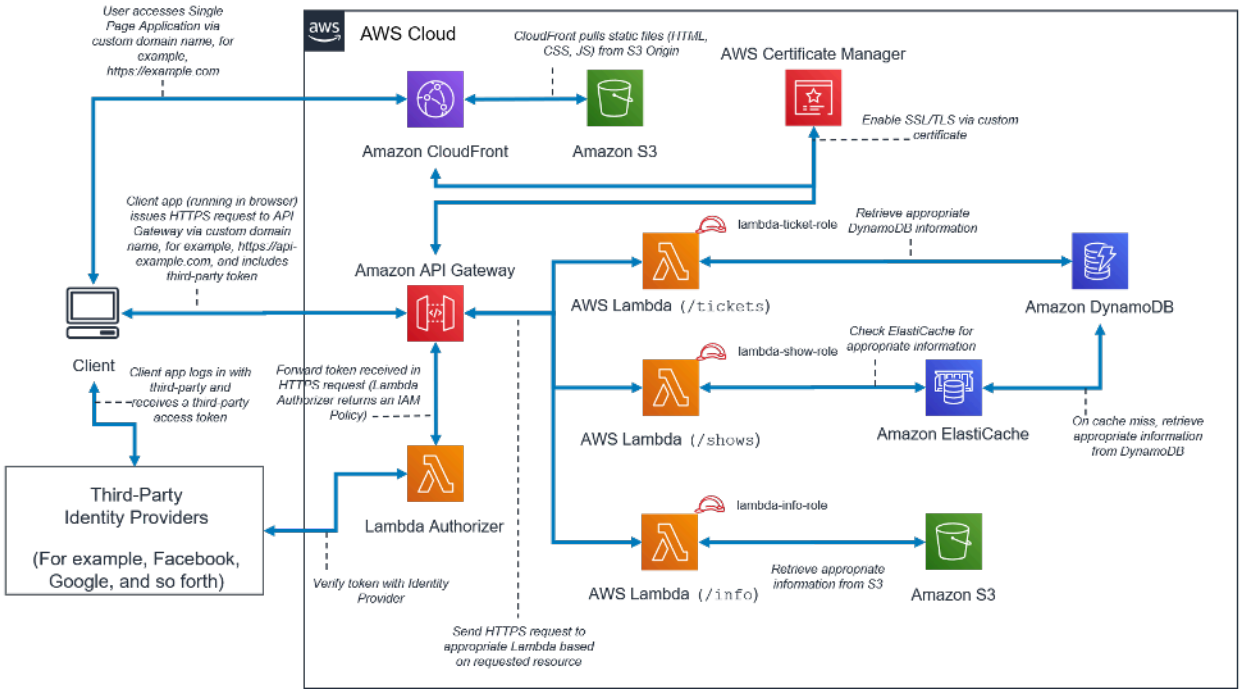
サーバーレスモバイルバックエンドのアーキテクチャパターン

表 1 - モバイルバックエンド階層コンポーネント

Tier	コンポーネント
プレゼンテーション	ユーザーデバイスで実行されているモバイルアプリケーション。
[Logic] (ロジック)	Amazon API Gateway と AWS Lambda。 このアーキテクチャは、3 つの公開サービス (`/tickets`、`/shows`、) を示しています`/info`。API Gateway エンドポイントは Amazon Cognito ユーザープールによって保護されます。この方法では、ユーザーは Amazon Cognito ユーザープールにサインインし (必要に応じてフェデレーティッドサードパーティーを使用)、API Gateway コールの認可に使用されるアクセストークンと ID トークンを受け取ります。

Tier	コンポーネント
	各 Lambda 関数には、適切なデータソースへのアクセスを提供する独自の Identity and Access Management (IAM) ロールが割り当てられます。
[データ]	DynamoDB は `/tickets`および `/shows`サービスに使用されます。 Amazon RDS は `/info`サービスに使用されます。この Lambda 関数は、 AWS Secrets Manager から Amazon RDS 認証情報を取得し、Elastic Network Interface を使用してプライベートサブネットにアクセスします。

単一ページアプリケーション

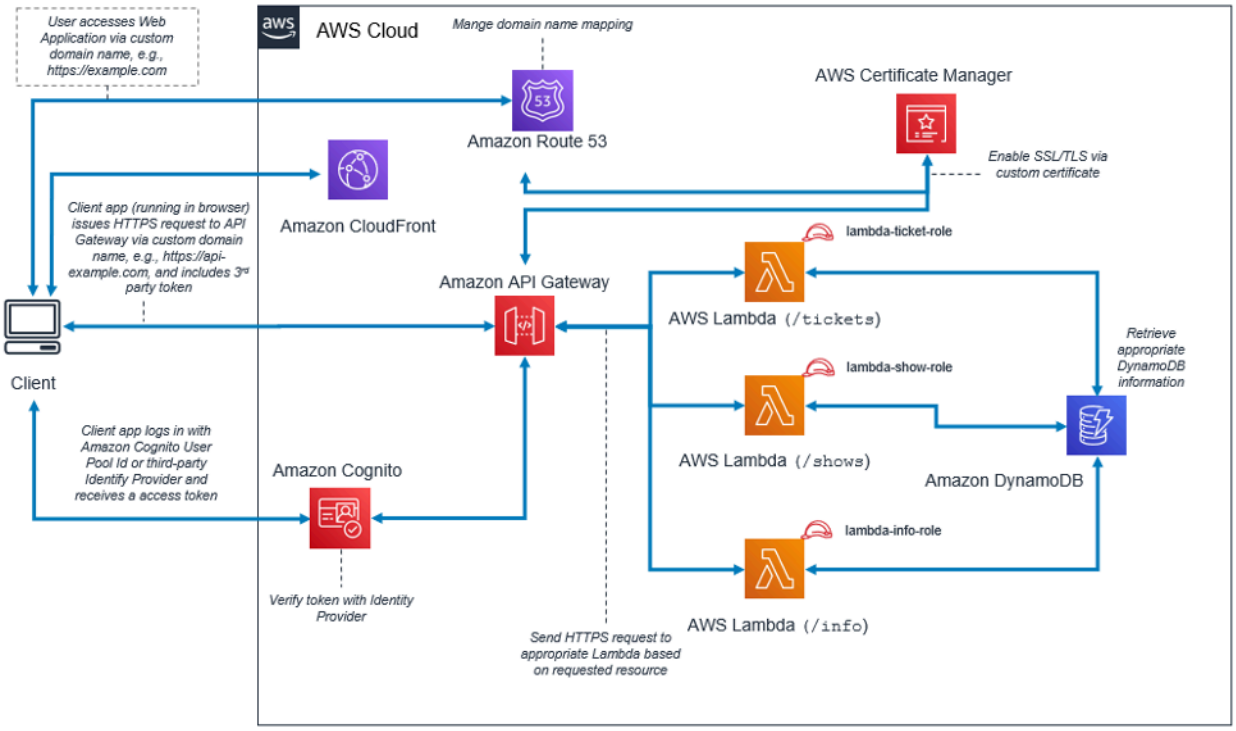


サーバーレスシングルページアプリケーションのアーキテクチャパターン

表 2 - 単一ページのアプリケーションコンポーネント

Tier	コンポーネント
プレゼンテーション	CloudFront によって配信される Amazon S3 でホストされる静的ウェブサイトコンテンツ。 AWS Certificate Manager では、カスタム SSL/TLS 証明書を使用できます。
[Logic] (ロジック)	API Gateway と AWS Lambda。 このアーキテクチャは、3 つの公開サービス (`/tickets`、`/shows`、) を示しています`/info`。API Gateway エンドポイントは Lambda オーソライザーによって保護されます。この方法では、ユーザーはサードパーティーの ID プロバイダーを介してサインインし、アクセストークンと ID トークンを取得します。これらのトークンは API Gateway 呼び出しに含まれ、Lambda オーソライザーはこれらのトークンを検証し、API 開始アクセス許可を含む IAM ポリシーを生成します。 各 Lambda 関数には、適切なデータソースへのアクセスを提供する独自の IAM ロールが割り当てられます。
[データ]	Amazon DynamoDB は、 `/tickets`および `/shows`サービスに使用されます。 Amazon ElastiCache は、データベースのパフォーマンスを向上させるために `/shows`サービスによって使用されます。キャッシュミスは DynamoDB に送信されます。 Amazon S3 は、 で使用される静的コンテンツをホストするために使用されます`/info service`。

ウェブアプリケーション



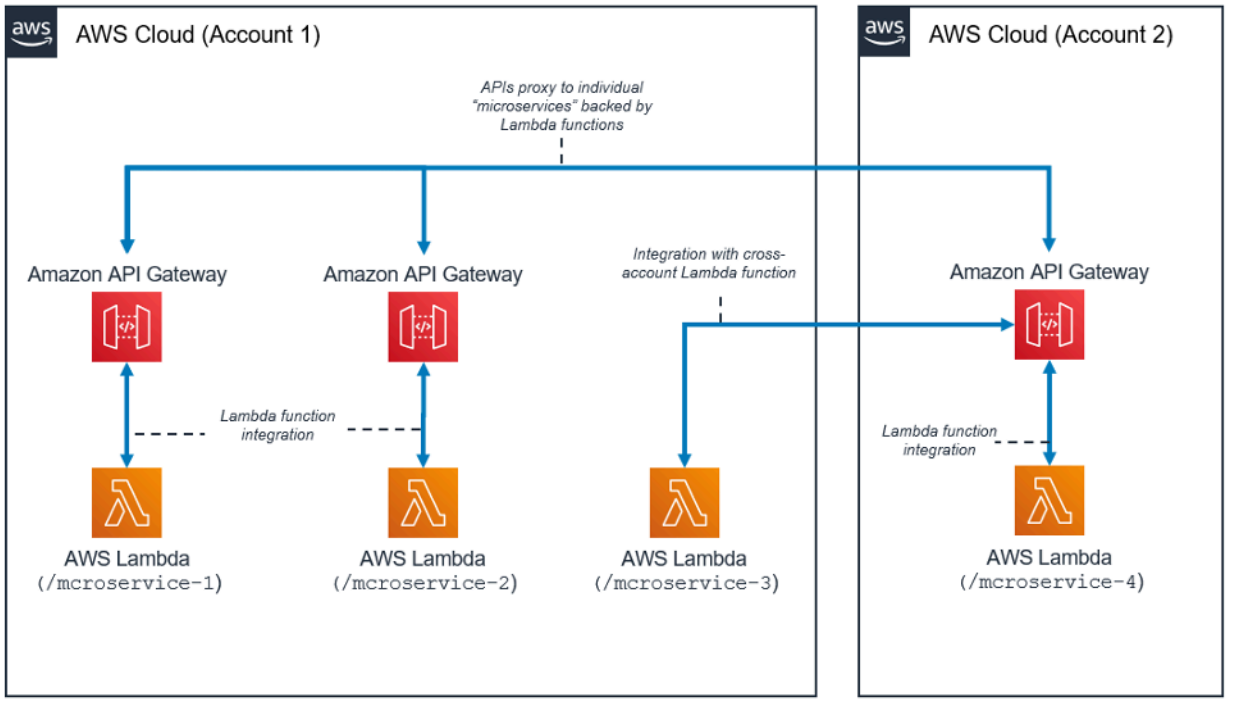
ウェブアプリケーションのアーキテクチャパターン

表 3 - ウェブアプリケーションコンポーネント

Tier	コンポーネント
プレゼンテーション	フロントエンドアプリケーションは、create-react-app などの React ユーティリティによって生成されるすべての静的コンテンツ (HTML、CSS、JavaScript、イメージ) です。Amazon CloudFront は、これらのすべてのオブジェクトをホストします。ウェブアプリケーションを使用すると、すべてのリソースがブラウザにダウンロードされ、そこから実行が開始されます。ウェブアプリケーションは、APIs を呼び出すバックエンドに接続します。
[Logic] (ロジック)	ロジックレイヤーは、API Gateway REST APIs。

Tier	コンポーネント
	このアーキテクチャは、複数の公開サービスを示しています。アプリケーションの異なる側面を処理する複数の異なる Lambda 関数があります。Lambda 関数は API Gateway の背後にあり、API URL パスを使用してアクセスできます。 ユーザー認証は、Amazon Cognito ユーザープールまたはフェデレーティッドユーザープロバイダーを使用して処理されます。API Gateway はAmazon Cognito とのすぐに使える統合を使用します。ユーザーが認証された後にのみ、クライアントは JSON Web Token (JWT) トークンを受け取り、API コールを行うときに使用する必要があります。 各 Lambda 関数には、適切なデータソースへのアクセスを提供する独自の IAM ロールが割り当てられます。
[データ]	この例では、DynamoDB がデータストレージに使用されますが、ユースケースと使用シナリオに応じて、他の専用の Amazon データベースまたはストレージサービスを使用できます。

Lambda を使用したマイクロサービス



Lambda を使用したマイクロサービスのアーキテクチャパターン

マイクロサービスアーキテクチャパターンは、一般的な 3 層アーキテクチャに縛られるわけではありませんが、この一般的なパターンは、サーバーレスリソースの使用から大きなメリットを得ることができます。

このアーキテクチャでは、各アプリケーションコンポーネントが分離され、個別にデプロイおよび運用されます。Amazon API Gateway で作成された API と、その後 によって起動される関数は AWS Lambda、マイクロサービスを構築するために必要なすべてです。チームはこれらのサービスを使用して、必要な粒度レベルまで環境を切り離してフラグメント化できます。

一般的に、マイクロサービス環境では、新しいマイクロサービスの作成に伴うオーバーヘッドの繰り返し、サーバーの密度と使用率の最適化に関する問題、複数のマイクロサービスの複数のバージョンを同時に実行する複雑さ、多くの個別のサービスと統合するためのクライアント側のコード要件の急増などの問題が発生する可能性があります。

サーバーレスリソースを使用してマイクロサービスを作成すると、これらの問題は解決しにくくなり、場合によっては単に消えます。サーバーレスマイクロサービスパターンは、後続の各マイクロサービスの作成の障壁を低くします (API Gateway では、既存の APIs、他のアカウントでの Lambda 関数の使用も許可されます) 。サーバー使用率の最適化は、このパターンには関係しなくなりまし

た。最後に、Amazon API Gateway はプログラムで生成されたクライアント SDKs を多くの一般的な言語で提供し、統合オーバーヘッドを削減します。

結論

多層アーキテクチャパターンは、メンテナンス、デカップリング、スケーリングが簡単なアプリケーションコンポーネントを作成するベストプラクティスを促進します。API Gateway によって統合が行われ、 内で計算が行われるロジック層を作成すると AWS Lambda、これらの目標を達成するための労力を削減しながら、これらの目標を実現できます。これらのサービスを組み合わせることで、クライアント用の HTTPS API フロントエンドと、一般的なサーバーベースのインフラストラクチャの管理に伴うオーバーヘッドを排除しながら、ビジネスロジックを適用するための安全な環境が提供されます。

寄稿者

本ドキュメントの寄稿者は次のとおりです。

- Andrew Baird、AWS ソリューションアーキテクト
- Bryant Bost、AWS ProServe コンサルタント
- AWS Mobile、Tech、Senior Product Manager、Stefano Buliani
- AWS Mobile、シニアプロダクトマネージャー、Vyom Nagrani
- AWS Mobile、シニアプロダクトマネージャー、Ajay Nair
- グローバルソリューションアーキテクト、Rahul Popat
- Brajendra Singh、シニアソリューションアーキテクト

詳細情報

詳細については、次を参照してください。

- AWS ホワイトペーパーとガイド

ドキュメントの改訂

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
マイナーな更新	バグ修正と、全体での多数のマイナーな変更。	2022 年 4 月 1 日
ホワイトペーパーの更新	新しいサービスの機能とパターンを更新しました。	2021 年 10 月 20 日
ホワイトペーパーの更新	新しいサービスの機能とパターンを更新しました。	2021 年 6 月 1 日
ホワイトペーパーの更新	新しいサービス機能を更新しました。	2019 年 9 月 25 日
初版発行	ホワイトペーパーの発行。	2015 年 11 月 1 日

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、(b) AWS の現行製品と慣行について説明しており、これらは予告なしに変更されることがあり、(c) AWS およびその関連会社、サプライヤー、またはライセンサーからの契約上の義務や保証をもたらすものではありません。AWS の製品やサービスは、明示または黙示を問わず、一切の保証、表明、条件なしに「現状のまま」提供されます。お客様に対する AWS の責任は AWS 契約によって規定されています。また、本文書は、AWS とお客様との間の契約に属するものではなく、また、当該契約が本文書によって修正されることもありません。