AWS サーバーレス多層アーキテクチャ

Amazon API Gateway と AWS Lambda の使用

初版*: 2015* 年 *11* 月

最終更新*: 2021* 年 *10* 月 *20* 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ており、(b) AWS の現行製品とプラクティスを表したものであり、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義務や確約を意味するものではありません。AWS の製品やサービスは、明示または暗示を問わず、いかなる保証、表明、条件を伴うことなく「現状のまま」提供されます。お客様に対する AWS の責任は、AWS 契約により規定されます。本書は、AWS とお客様の間で行われるいかなる契約の一部でもなく、そのような契約の内容を変更するものでもありません。

目次

要約

このホワイトペーパーは、多層アーキテクチャを設計したり、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したりする方法を変更するために、アマゾン ウェブ サービス (AWS) のイノベーションをどのように活用できるかを説明します。アーキテクトとデベロッパーは、Amazon API Gateway、AWS Lambda、その他のサービスを使用して、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低減できるようになります。

はじめに

多層アプリケーション (3 層、*n* 層など) は、何十年にもわたって基礎となってい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では人気の高いパターンとして残っています。多層アーキテクチャを説明するために使用される言葉はさまざまですが、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ています。

- **プレゼンテーション層** – ユーザーが直接やり取りするコンポーネント (ウェブページやモバイルアプリケーションの UI 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 (例: 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)。

多層アーキテクチャのパターンは、疎結合でスケーラブルなアプリケーションコンポーネントを (多くの場合、別々のチームが) 個別に開発、管理、保守できるようにするための一般的なフレームワークを提供します。

ネットワークが階層間の境界として機能するこのパターン (階層が別の階層とやり取りするためにはネットワーク呼び出しを行う必要がある) の結果として、多層アプリケーションの開発では、差別化につながらないアプリケーションコンポーネントを多数作成しなければならないことが多くあります。これらのコンポーネントには、次のものがあります。

- 階層間の通信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コード

これらの例はすべて、多層アプリケーションでは必要であっても、アプリケーションごとに実装が大きく変化しない「定型」コンポーネントと考えることができます。

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ており、そ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を大幅にシンプルして、従来のサーバー管理に関連するオーバーヘッドを排除します。API を作成、管理できる Amazon API Gateway と任意の関数コードを実行できる AWS Lambda を一緒に使用すると、堅牢な多層アプリケーションを簡単に作成できます。

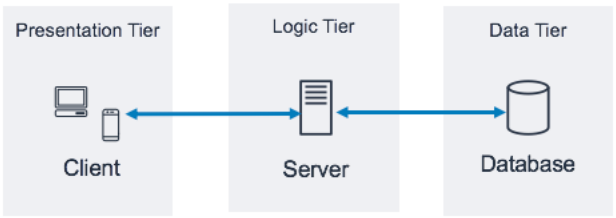
Amazon API Gateway と AWS Lambda の統合により、ユーザー定義の関数コードを HTTPS リクエストから直接開始できます。リクエストボリュームにかかわらず、Amazon API Gateway と AWS Lambda はどちらも、アプリケーションのニーズを正確にサポートするように自動的にスケールします (スケーラビリティ情報については、「Amazon API Gateway のクォータおよび重要な注意事項」を参照してください)。この 2 つのサービスを組み合わせることで、アプリケーションにとって重要なコードのみを記述する層を作成でき、高可用性を実現するためのアーキテクチャの設計、クライアント SDK の記述、サーバー/オペレーティングシステム (OS) の管理、スケーリング、クライアント認証メカニズムの実装など、多層アーキテクチャを実装するために必要な、差別化につながらないその他の作業を行う必要がなくなります。

Amazon API Gateway と AWS 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。AWS では、アプリケーションの要件に応じて、サーバーレスのプレゼンテーション層 (Amazon CloudFront と Amazon Simple Storage Service (Amazon S3))、およびデータ層 (Amazon Aurora、Amazon DynamoDB など) を作成するオプションも用意しています。

本書では、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合わせて説明します。もちろん、この多層パターンは、典型的な 3 層ウェブアプリケーション以外にも応用できます。

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については、詳しく学習できるリソースがオンラインに数多くあります。本書では、Amazon API Gateway と AWS Lambda を使用して 3 層アーキテクチャを実装する具体的なパターンについて説明します。

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層は、アプリケーションの頭脳に相当します。これは、従来のサーバーベースの実装と比較して、Amazon API Gateway と AWS Lambda の使用が最も大きな影響を与える可能性がある場所です。この 2 つのサービスの機能を組み合わせることで、高可用性とスケーラビリティを備えたセキュアな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数千台のサ

ーバーが必要となる場合がありますが、Amazon API Gateway と AWS Lambda を使用することで、容量にかかわらず、お客様によるサーバー管理は不要になります。さらに、この 2 つのマネージドサービスを組み合わせて使用することで、次のような利点も得られます。

- **AWS Lambda**
  - OS の選定、保護、パッチ適用、管理が不要
  - サーバーのサイジング、モニタリング、スケールが不要
  - 過剰なプロビジョニングによるコスト発生のリスクを低減
  - プロビジョニングの不足によるパフォーマンスへのリスクを低減
- **Amazon API Gateway**
  - API をデプロイ、モニタリング、保護するのためのシンプルなメカニズム
  - 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、サポートされている任意の言語 (Node.js、Python、Ruby、Java、Go、.NET。詳細については、「Lambda のよくある質問」を参照してください) であらゆる関数コードを実行できるコンピューティングサービスです。サーバーのプロビジョニング、管理、スケーリングは不要です。AWS Lambda 関数は、マネージド型の独立したコンテナで実行され、AWS が提供している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。これを、**イベントソース**と呼びます。すべてのイベントソースについては、「Lambda のよくある質問」を参照してください。

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

ビジネスロジック (配置のみ、サーバーは不要)

AWS Lambda では、イベントが開始されたときに実行するハンドラーと呼ばれる関数コードを記述する必要があります。Amazon API Gateway で AWS 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 Amazon API Gateway を設定できます。サーバーレスの多層アーキテクチャでは、Amazon API Gateway で作成した各 API は、必要なビジネスロジックを呼び出す AWS 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作成すると、アプリケーション機能を公開するための詳細度 (API ごとに 1 つの AWS Lambda 関数、または API メソッドごとに 1 つの AWS Lambda 関数) を定義できます。AWS Lambda 関数内では、ハンドラーは他の依存関係 (例えば、アップロードしたコード、ライブラリ、ネイティブバイナリ、そして外部ウェブサービス) や、他の AWS Lambda 関数にもアクセスできます。

AWS Lambda 関数を作成または更新するには、zip ファイルで AWS Lambda デプロイパッケージとして Amazon S3 バケットにコードを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コードをパッケージ化する必要があります。AWS Lambda 関数では、さまざまなデプロイ方法を使用できます。例えば、AWS マ

ネジメントコンソール、AWS コマンドラインインターフェイス (CLI) の実行のほか、Infrastructure as Code テンプレートや、AWS CloudFormation、AWS Serverless Application Model (AWS SAM)、AWS Cloud Development Kit (AWS CDK) などのフレームワークの実行などです。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に、デプロイパッケージ内でリクエストハンドラーとして機能するメソッドを指定します。デプロイパッケージ内に AWS Lambda 関数ごとに固有のハンドラーを用意することで、複数の AWS Lambda 関数の定義で同じデプロイパッケージを再利用できます。

AWS Lambda のセキュリティ

AWS 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、AWS Lambda 関数が呼び出される必要があります。AWS IAM ポリシーを使用すると、定義した Amazon API Gateway のリソースによって呼び出されない限り開始できない AWS 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 のサービスにわたるリソースベースのポリシーを使用して定義できます。

各 AWS Lambda 関数は、デプロイ時に割り当てられる IAM ロールを引き受けます。この IAM ロールは、AWS Lambda 関数がやり取りできる他の AWS のサービスとリソース (例: Amazon DynamoDB テーブル、Amazon S3) を定義します。AWS Lambda 関数のコンテキストでは、これを実行ロールと呼びます。

機密情報を AWS Lambda 関数内には保存しないでください。AWS IAM は AWS Lambda の実行ロールを通じて AWS のサービスへのアクセスを処理します。AWS 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、AWS Secrets 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) から、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独立した環境で実行されます。AWS Lambda 関数をスケールする必要はありません。関数がイベント通知を受け取るたびに、AWS Lambda はコンピューティングフリート内で利用可能な容量を見つけ、ユーザーが定義したランタイム、メモリ、ディスク、タイムアウトの設定を使用してコードを実行します。このパターンでは、AWS は関数のコピーを必要な数だけ開始できます。

AWS Lambda ベースのロジック層は、ニーズに合わせて常に適切なサイズになっています。マネージド型のスケーリングおよびコード開始の同時実行によるトラフィックの急増をすばやく吸収する機能と、AWS Lambda の従量制料金を組み合わせることで、アイドル状態のコンピューティング性能に対して支払いを行うことなく、常に顧客の要求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

AWS Lambda 関数のデプロイと管理を支援するには、次を含むオープンソースのフレームワークである AWS Serverless Application Model (AWS SAM) を使用します。

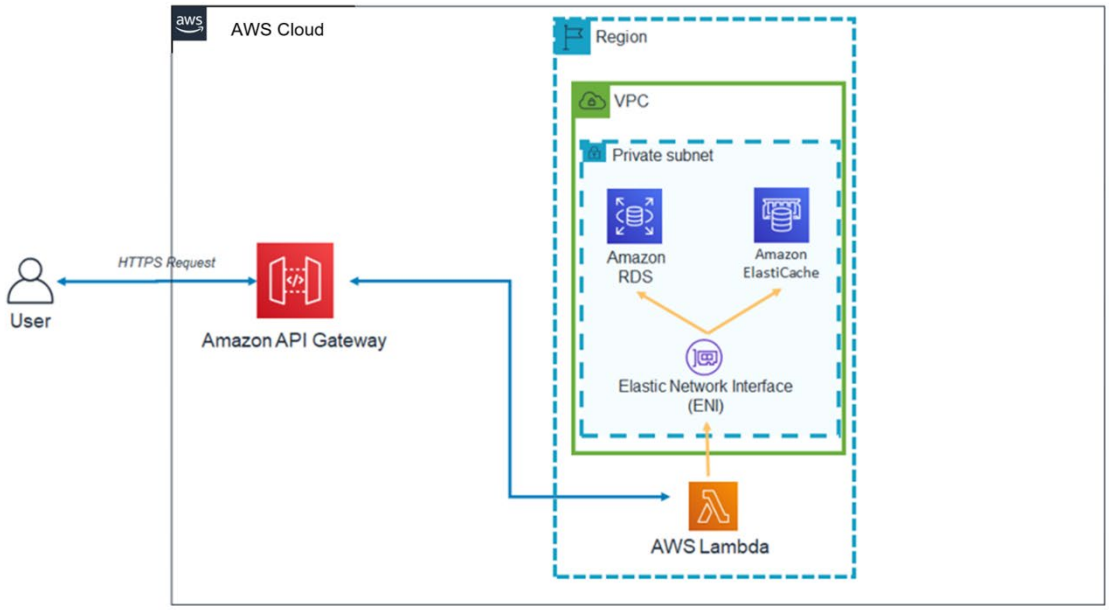
- **AWS SAM テンプレートの仕様** – 関数を定義し、アップロードとデプロイをシンプルにするための環境、許可、設定、イベントを記述するために使用される構文。

- **AWS SAM CLI** – AWS SAM テンプレート構文の検証、関数のローカル呼び出し、AWS Lambda 関数とデプロイパッケージ関数のデバッグを可能にするコマンド。

また、AWS CDK を使用することもできます。これは、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 CloudFormation を使用し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す。AWS CDK は AWS リソースを定義する命令型の方法を提供しますが、AWS SAM は宣言型の方法を提供します。

通常、AWS Lambda 関数をデプロイすると、割り当てられた IAM ロールによって定義された許可で呼び出され、インターネット向けエンドポイントにアクセスできます。ロジック層の中核をなす 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、このデータ層が (プライベートサブネットで) 適切に分離されていることを確認することが重要です。

プライベートデータベースインスタンスなど、パブリックに公開できないリソースに AWS Lambda 関数からアクセスする場合は、AWS アカウントの仮想プライベートクラウド (Amazon VPC) のプライベートサブネットに接続するように AWS Lambda 関数を設定できます。関数を Amazon VPC に接続すると、AWS Lambda は関数の Amazon 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。



Amazon VPC 内の AWS Lambda のアーキテクチャパターン

Amazon VPC で AWS Lambda を使用すると、ビジネスロジックで使用するデータベースやその他のストレージメディアをインターネットから遮断できます。また、インターネットからのデータの操作を、ユーザーが定義した API とユーザーが記述した AWS Lambda の関数コードからのリクエストに限定することもできます。

Amazon API Gateway

Amazon API Gateway は、デベロッパーが規模にかかわらず API の作成、パブリッシュ、メンテナンス、モニタリング、セキュリティ保護を行えるようにするための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 Amazon API Gateway 経由で公開された API と統合されます。Amazon API Gateway 経由でサービス指向の多層アーキテクチャに公開される API の適用範囲は、個々のアプリケーション機能を分離し、REST エンドポイントを通じてこの機能を公開

する機能です。Amazon API Gateway には、ロジック層に強力な機能を追加できる特有の機能と品質があります。

AWS Lambda との統合

Amazon API Gateway では、REST タイプと HTTP タイプの API の両方がサポートされています。Amazon API Gateway の 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 (`/tickets` など) を通じてアクセスできる論理エンティティです。メソッドは、API リソース (`GET /tickets` など) に送信される API リクエストに対応します。Amazon API Gateway では、AWS Lambda 関数を使用して各メソッドを後方に置くことができます。つまり、Amazon 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、Amazon API Gateway は AWS Lambda 関数を呼び出します。

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

プロキシの統合

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

非プロキシの統合

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 AWS Lambda ハンドラー関数の event パラメータに渡す方法を設定します。さらに、AWS Lambda のアウトプットをユーザーに変換して戻す方法を設定します。

> **注意**: Amazon API Gateway は、モック統合 (初期のアプリケーション開発に便利です) や Amazon S3 オブジェクトへの直接プロキシなど、AWS Lambda 外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 でデプロイする各 エッジ最適化 API は、内部で Amazon CloudFront ディストリビューションを経由するようになっています。Amazon CloudFront は、コンテンツ配信ウェブサービスで、API を使用するクライアントの接続ポイントとして、Amazon のグローバルネットワークのエッジロケーションを使用します。これにより、API のレスポンスのレイテンシーが減少します。また、Amazon CloudFront では、世界中にある複数のエッジロケーションを使用することで、分散型サービス妨害 (DDoS) 攻撃の対策となる機能を提供します。詳細については、「AWS Best Practices for DDoS Resiliency」のホワイトペーパーを参照してください。

Amazon API Gateway を使用してオプションのインメモリキャッシュにレスポンスを保存すると、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、繰り返しの 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、AWS 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加速し、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。Amazon 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、総開発コストを削減できるため、組織はより自由に実験とイノベーションを行うことができます。

アプリケーション開発の初期段階では、新しいアプリケーションをより迅速に配信するために、ログ記録とメトリクス収集の実装は無視されがちです。これにより、本番環境で実行されているアプリケーションにこれらの機能をデプロイする際に、技術的負債や運用上のリスクが生じる可能性があります。Amazon API Gateway は Amazon CloudWatch とシームレスに統合されます。Amazon CloudWatch は Amazon API Gateway から加工前の生データを収集し、API 実装のモニタリングのために、ほぼリアルタイムの読み取り可能なメトリクスに処理します。Amazon API Gateway では、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 によるトレースもサポートされています。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に対するリスクなしに、本番環境で実行され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ライフタイムがどの程度なのかはわからないことがあり、短期間であることがわかっている場合もあります。開始点に Amazon API Gateway が提供するマネージド機能が既に含まれ、API がリクエストの受信を開始した後にのみインフラストラクチャコストが発生する場合、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容易になる可能性があります。詳細については、「Amazon API Gateway の料金」を参照してください。

迅速に反復し、俊敏性を維持

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、ユーザー要件の変化に迅速に対応できます。

ステージのデプロイ

Amazon API Gateway で API をデプロイする場合、デプロイを Amazon API Gateway の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ために使用できるようになります。この規則を使用すると、アプリケーションを *dev*、*test*、*stage*、*prod* のステージに簡単にデプロイし、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 API の異なるバージョンを作成します。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が引き続き影響を受けることはなくなります。

AWS Lambda との統合の分離

Amazon API Gateway の API と AWS Lambda 関数の統合は、Amazon API Gateway のステージ変数と AWS Lambda 関数のエイリアスを使用して分離できます。これにより、API のデプロイがシンプルになり、高速化されます。API で AWS Lambda 関数名またはエイリアスを直接設定する代わりに、AWS Lambda 関数内の特定のエイリアスを指すことができる API のステージ変数を設定できます。デプロイ時に、AWS Lambda 関数のエイリアスを指すようにステージ変数の値を変更すると、API は特定のステージの AWS Lambda エイリアスの背後にある AWS Lambda 関数の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、テスト目的で新しいバージョンの API をデプロイし、ベースバージョンは同じステージでの通常のオペレーションの本番環境用リリースとしてデプロイされたままにな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、事前設定された比率で本番環境用リリースと Canary リリースにランダムに分離されます。Amazon API Gateway の API を Canary リリースのデプロイ用に設定して、制限されたユーザーセットで新しい機能をテストできます。

カスタムドメイン名

Amazon API Gateway によって提供される URL の代わりに、直感的でビジネスフレンドリーな URL 名を API に提供できます。Amazon API Gateway には、API のカスタムドメインを設定する機能が用意されています。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、マルチレベルのベースパス (`myservice`、`myservice/cat/v1`、`myservice/dog/v2` など) を選択して、代替 URL 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、許可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ように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 がロジック層の保護に役立つさまざまな方法を利用できます。

転送のセキュリティ

API へのリクエストはすべて HTTPS を経由して行い、伝送中にも暗号化されるようにします。

Amazon API Gateway には、組み込みの SSL/TLS 証明書が用意されています。パブリック API のカスタムドメイン名オプションを使用する場合は、AWS Certificate Manager を使用して独自の SSL/TLS 証明書を提供できます。Amazon API Gateway では、相互 TLS (mTLS) 認証もサポートされています。相互 TLS は API のセキュリティを強化し、クライアントのなりすましや中間者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 認証

API の一部として作成するリソースとメソッドの各組み合わせには、一意の Amazon リソースネーム (ARN) が付与されます。この ARN は、AWS Identity and Access Management (IAM) ポリシーにリストアップされます。

Amazon API Gateway で API に認証を追加するには、3 つの一般的な方法があります。

- **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) 認証と AWS IAM ポリシーを使用します。同じ認証情報を使用し、必要に応じて他の AWS のサービスやリソース (例: 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 ユーザープールを通じてサインインし、リクエストの認証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証スキームを実装する AWS Lambda 関数を定義します。

アクセス制限

Amazon API Gateway では、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との関連付けがサポートされています。Amazon CloudWatch で API キーの使用状況をモニタリングできます。

Amazon API Gateway では、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がサポートされています。

プライベート API

Amazon 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 API を作成できます。これは、Amazon 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、複数の AWS アカウントを対象に、Amazon VPC と VPC エンドポイントに対して API へのアクセスを有効にするか拒否できます。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。また、AWS Direct Connect を使用して、オンプレミスネットワークから Amazon VPC への接続を確立し、その接続を介してプライベート 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セキュアな接続を使用し、Amazon ネットワークを離れません。トラフィックはパブリックインターネットから分離されます。

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の API 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 API を保護する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攻撃から API を保護します。Amazon API Gateway で AWS WAF を使用すると、API を保護できます。

データ層

AWS Lambda をロジック層として使用しても、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。AWS Lambda 関数は、AWS Lambda の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ることで、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を (AWS KMS または AWS Secrets Manager を通じて) 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 では、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データストアを多数用意しています。

サーバーレスデータストレージのオプション

- 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、パフォーマンスを備えたオブジェクトストレージサービスです。
- Amazon Aurora は、MySQL および PostgreSQL と互換性のあるリレーショナルデータベースで、クラウドに合わせて構築されています。このデータベースは、オープンソースデータベースのシンプルさとコスト効率を備え、従来のエンタープライズ用データベースのパフォーマンスと可用性を併せ持っています。Amazon Aurora は、サーバーレス使用量モデルと従来の使用量モデルの両方を提供します。
- Amazon DynamoDB は、あらゆる規模で 10 ミリ秒未満のパフォーマンスを実現する key-value データベースおよびドキュメントデータベースです。これはマルチリージョンに対応した、耐久性のあるフルマネージド型のサーバーレスデータベースで、インターネットスケールのアプリケーションに対応した、組み込みのセキュリティ、バックアップと復元の機能、インメモリキャッシュを備えています。
- Amazon Timestream は、IoT および運用アプリケーションに適した、高速でスケーラブルなフルマネージド型の時系列データベースサービスです。1 日あたり数兆規模のイベントを、リレーショナルデータベースの 10 分の 1 のコストで簡単に保存および分析できます。IoT デバイスや IT システムの普及や、産業機器のスマート化により、時系列データ (時間の経過に伴うモノの変化を記録したデータ) は、急速に増加しているデータ型の 1 つです。
- Amazon Quantum Ledger Database (Amazon QLDB) はフルマネージド型の台帳データベースで、信頼された中央機関が所有する、透過的でイミュータブルであり、暗号的に検証可能なトランザクションログを備えています。Amazon

QLDB ではアプリケーションデータの全変更が追跡され、完全で検証可能な変更履歴が長期間維持されます。

- Amazon Keyspaces (for Apache Cassandra) は、スケーラブルで可用性の高い、Apache Cassandra 互換のマネージドデータベースサービスです。Amazon Keyspaces では、現在使用しているのと同じ Cassandra アプリケーションコードとデベロッパーツールを使用して、AWS で Cassandra ワークロードを実行できます。サーバーをプロビジョニング、パッチ適用、または管理する必要はなく、ソフトウェアをインストール、保守、または運用する必要もありません。Amazon Keyspaces はサーバーレスであるため、使用するリソースのみに料金を支払うだけで、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およびスケールダウンします。
- Amazon Elastic File System (Amazon EFS) は、シンプルでサーバーレスのセットアンドフォーゲットの伸縮自在なファイルシステムを提供します。これにより、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ます。AWS クラウドサービスおよび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まで拡張できるように構築されています。Amazon EFS により、ファイルを追加および削除するときにファイルシステムを自動的に拡大および縮小できます。これにより、拡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 AWS Lambda 関数からマウントすることが可能で、これによって API に価値のあるファイルストレージオプションを追加することができます。

非サーバーレスデータストレージのオプション

- Amazon Relational Database Service (Amazon RDS) は、複数のエンジン (Aurora、PostgreSQL、MySQL、MariaDB、Oracle、Microsoft SQL Server) を使用し、リレーショナルデータベースのセットアップ、運用、スケーリングを可能にするマネージドウェブサービスです。また、メモリ、パフォーマンス、I/O 用に最適化された、いくつかの異なるデータベースインスタンスタイプで実行可能です。
- Amazon Redshift は、クラウドにおけるフルマネージド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 のフルマネージド型のデプロイです。普及しているオープンソース互換のインメモリデータストアを、シームレスにデプロイ、運用、スケールできます。
- Amazon Neptune は、高速で信頼性に優れたフルマネージド型のグラフデータベースサービスで、関連性が高いデータセットを使って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。Amazon Neptune は、一般的なグラフモデル (プロパティグラフと W3C リソース記述フレームワーク (RDF)) 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) は高速で、スケーラブル、高い可用性を持ち、フルマネージドのドキュメントデータベースサービスで、MongoDB ワークロードをサポートします。
- 最後に、Amazon EC2 上で独立して実行されるデータストアを、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層

プレゼンテーション層は、インターネット経由で公開される Amazon API Gateway の REST エンドポイントを介してロジック層とのやり取りを処理します。HTTPS 対応クライアントまたはデバイスはいずれも、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はさまざまな形式 (デスクトップアプリケーション、モバイルアプリケーション、ウェブページ、IoT デバイスなど) を柔軟に利用できます。要件に応じて、プレゼンテーション層では以下の AWS サーバーレスサービスを使用できます。

- **Amazon Cognito** – サーバーレスのユーザー ID およびデータ同期サービスで、ユーザーのサインアップ、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ケーションに迅速かつ効率的に追加できます。Amazon Cognito は、数百万人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、SAML 2.0 を介したエンタープライズ ID プロバイダーによるサインインをサポートします。
- **Amazon S3 と Amazon CloudFront** – ウェブサーバーのプロビジョニングを必要とせずに、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 S3 バケットから直接提供できます。Amazon CloudFront をマネージドコンテンツ配信ネットワーク (CDN) として使用することで、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で SSL/TLS を有効にすることができます。

AWS Amplify は、それぞれを連携させたり個別で使用したりできる、ツールとサービスのセットです。これらの機能により、フロントエンドウェブおよびモバイルのデベロッパーが、AWS によるスケーラブルなフルスタックアプリケーションを構築できるようにします。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、信頼性の高い Amazon の CDN により、世界中に数百の POP (Point Of Presence) 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。AWS Amplify では、一般的なウェブフレームワーク (JavaScript、React、Angular、Vue、Next.js) やモバイルプラットフォーム (Android、iOS、React Native、Ionic、Flutter) がサポートされています。ネットワーキング設定とアプリケーションの要件によっては、Amazon API Gateway の API を Cross-Origin Resource Sharing (CORS) 準拠にすることが必要になる場合があります。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 API を直接呼び出すことができます。

Amazon 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アクセスするための Amazon CloudFront のドメイン名 (`d2d47p2vcczkh2.cloudfront.net` など) が提供されます。Amazon Route 53 を使用し、ドメイン名を登録して Amazon CloudFront ディストリビューションに振り分けることも、既に所有しているドメイン名を Amazon CloudFront ディストリビューションに振り分けることもできます。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サイトにアクセスできます。Amazon Route 53 を使用してカスタムドメイン名を Amazon API Gateway のディストリビューションに割り当てることもできます。このため、ユーザーは使い慣れたドメイン名を使用して API を呼び出すことができます。

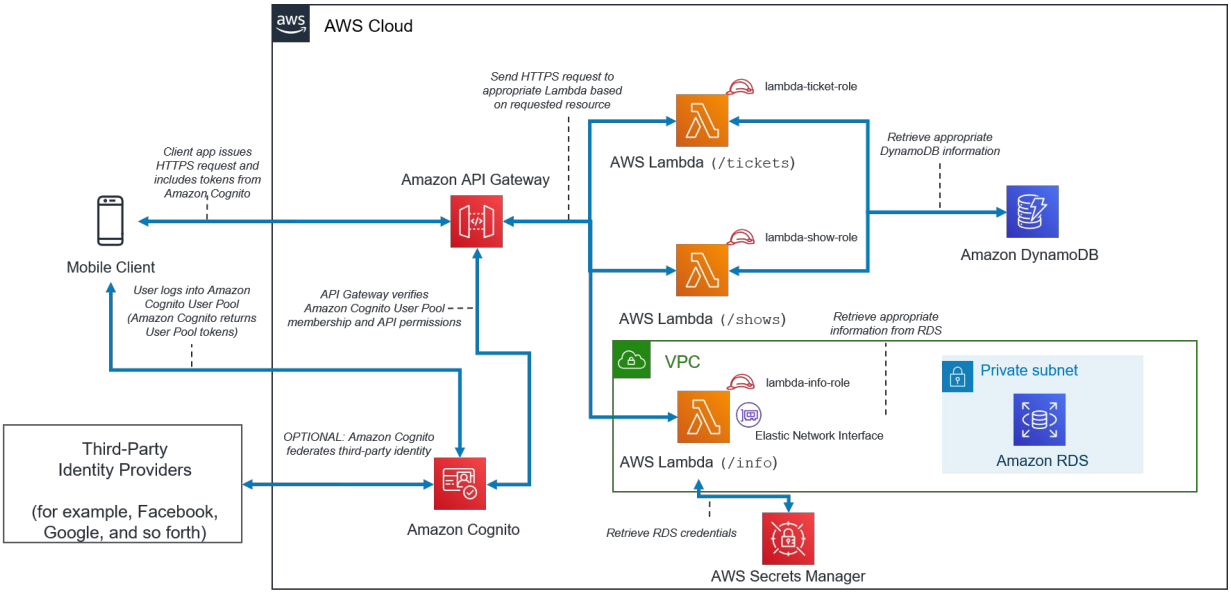
サンプルアーキテクチャパターン

Amazon API Gateway と AWS Lambda をロジック層として使用して、一般的なアーキテクチャパターンを実装できます。本書には、AWS Lambda ベースのロジック階層を使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 Amazon API Gateway および AWS 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、プレゼンテーション層のリソースをホストするサーバーレスの AWS リソースを使用しない汎用 HTTPS クライアント (デスクトップクライアント、Amazon EC2 で実行されているウェブサーバーなど) に拡張可能です。
- **シングルページアプリケーション** – Amazon S3 および Amazon CloudFront でホストされるシングルページアプリケーションは、Amazon API Gateway および AWS 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。
- **ウェブアプリケーション** – ウェブアプリケーションは、ビジネスロジックに AWS Lambda と Amazon API Gateway を使用する、汎用のイベント駆動型ウェブアプリケーションバックエンドです。また、データベースとして Amazon DynamoDB を使用し、ユーザー管理に Amazon Cognito を使用します。すべての静的コンテンツは AWS Amplify を使用してホストされます。

この 2 つのパターンに加えて、本書では、一般的なマイクロサービスアーキテクチャへの AWS Lambda と Amazon API Gateway の適用可能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の 3 層アーキテクチャではありませんが、疎結合なアプリケーションコンポーネントと、相互に通信するステートレスな個別の機能単位としてのデプロイが含まれる一般的なパターンです。

モバイルバックエンド



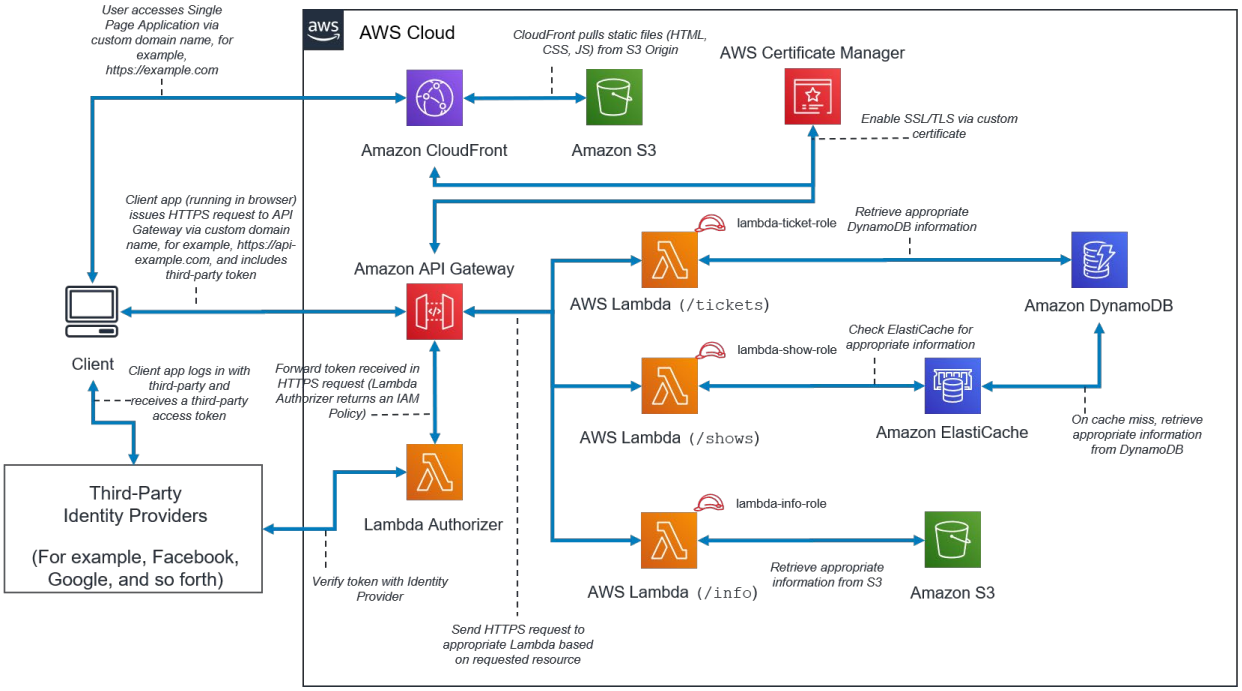
サーバーレスモバイルバックエンドのアーキテクチャパターン

表 *1* – モバイルバックエンド層のコンポーネント

階層	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。
ロジック	AWS Lambda と Amazon API Gateway。 このアーキテクチャは、3 つの公開されたサービス (`/tickets`、`/shows`、`/info`) を示します。Amazon 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を使用)、Amazon API Gateway の呼び出しを承認するため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。 各 AWS 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 AWS Identity and Access Management (IAM) ロールが割り当てられます。

階層	コンポーネント
データ	Amazon DynamoDB は `/tickets` サービスと `/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 AWS 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。

シングルページアプリケーション

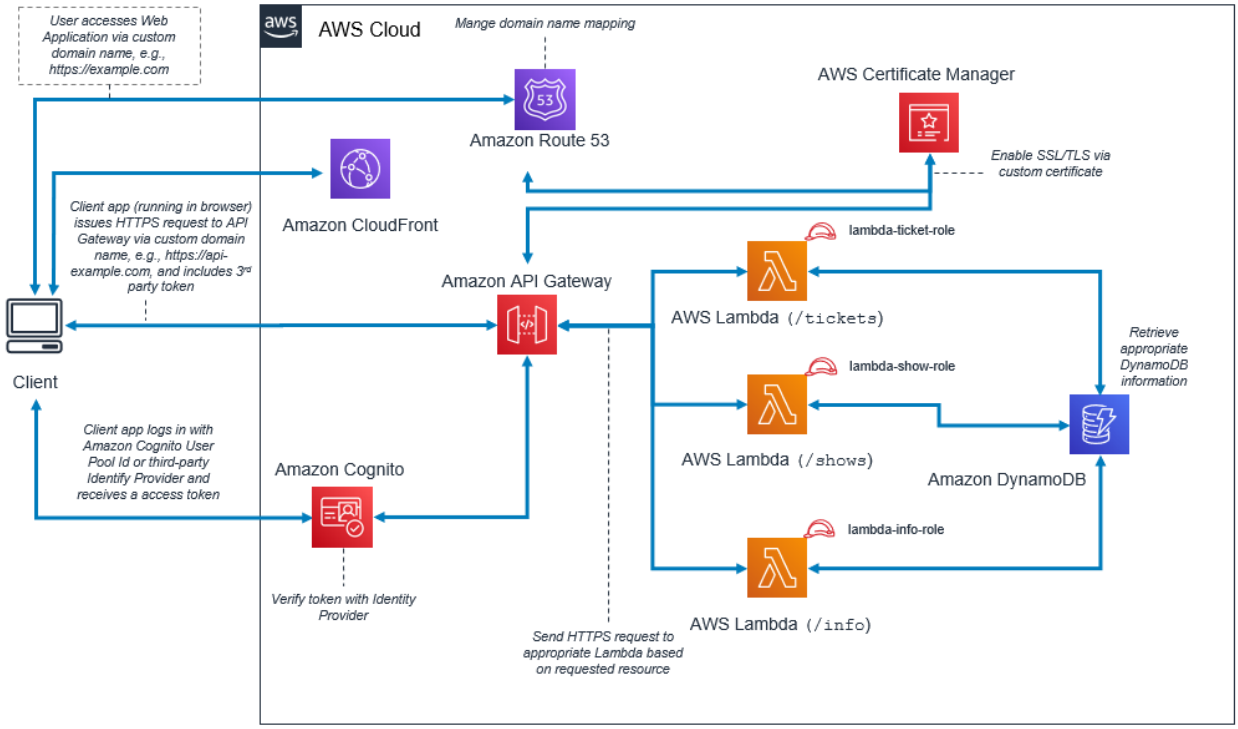


サーバーレスのシングルページアプリケーションのアーキテクチャパターン

表 2 – シングルページアプリケーションのコンポーネント

階層	コンポーネント
プレゼンテーション	静的ウェブサイトのコンテンツは Amazon S3 でホストされ、Amazon CloudFront によって配信されます。 AWS Certificate Manager では、カスタム SSL/TLS 証明書を使用できます。
ロジック	AWS Lambda と Amazon API Gateway。 このアーキテクチャは、3 つの公開されたサービス (`/tickets`、`/shows`、`/info`) を示します。Amazon API Gateway のエンドポイントは、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 ID プロバイダーを通じてサインインし、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 AWS API Gateway 呼び出しに含まれます。Lambda オーソライザーはこれらのトークンを検証し、API 開始許可を含む AWS IAM ポリシーを生成します。 各 AWS 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` サービスと `/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 Amazon DynamoDB に送信されます。 Amazon S3 は、`/info` サービス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。

ウェブアプリケーション



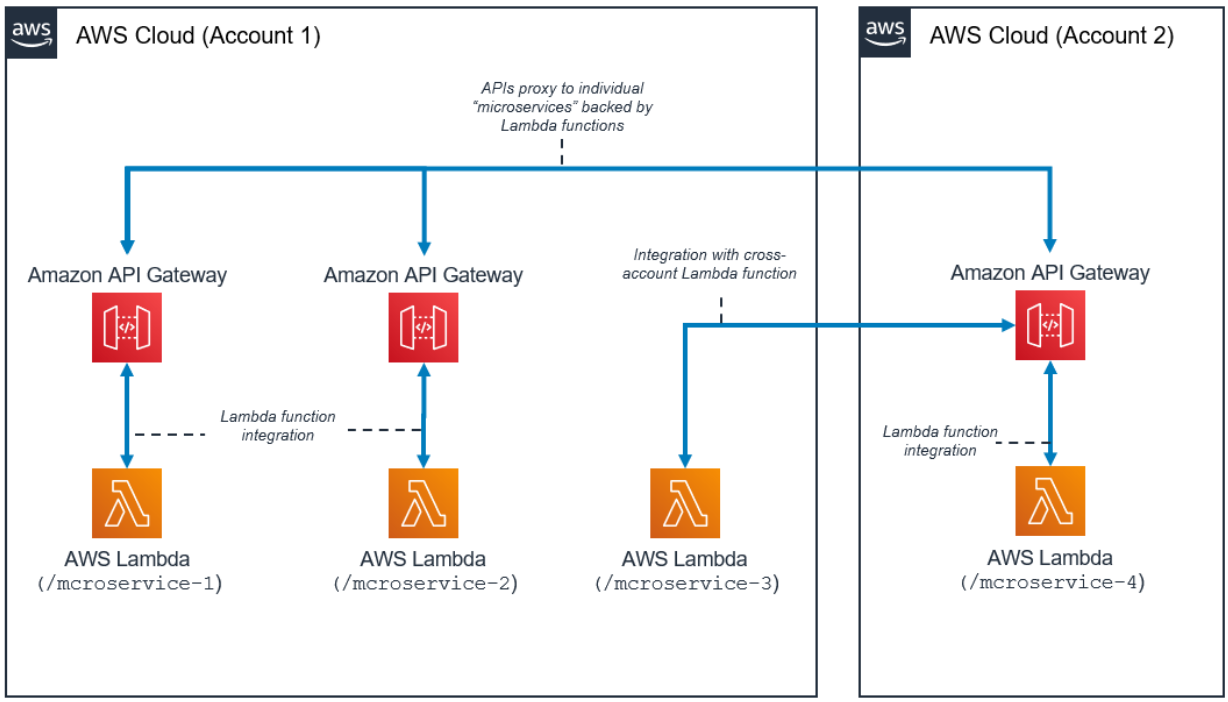
ウェブアプリケーションのアーキテクチャパターン

表 3 – ウェブアプリケーションのコンポーネント

階層	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すべて静的コンテンツ (HTML、CSS、JavaScript、イメージ) であり、create-react-app などの React ユーティリティによって生成されます。Amazon CloudFront は、これら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、API を呼び出すバックエンドに接続します。
ロジック	ロジック層は、Amazon API Gateway の REST API が前面に配置された AWS Lambda 関数を使用して構築されます。

階層	コンポーネント
	このアーキテクチャは、複数の公開されたサービスを示します。アプリケーションのさまざま側面を処理する、異なる複数の AWS Lambda 関数があります。AWS Lambda 関数は Amazon API Gateway の背後にあり、API の 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。Amazon API Gateway では、Amazon Cognito との統合をすぐに利用できます。ユーザーが認証された後でのみ、クライアントは JSON ウェブトークン (JWT) を受け取り、これを API コールの実行時に使用する必要があります。 各 AWS 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	この特定の例では、Amazon DynamoDB は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、Amazon の他の目的別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。

AWS Lambda を使用したマイクロサービス



AWS 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

マイクロサービスのアーキテクチャパターンは、一般的な 3 層アーキテクチャのみではありません。ただし、この一般的なパターンでは、サーバーレスリソースを使用することで大きなメリットを実現できます。

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、運用されます。マイクロサービスを構築するために必要なものは、Amazon API Gateway を使用して作成した API と AWS Lambda によって軌道される関数のみです。チームは、これらのサービスを活用して、十分に環境を疎結合化し、分離できます。

一般に、マイクロサービス環境では、新しいマイクロサービスを作成するたびに繰り返されるオーバーヘッド、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際の複雑さ、多数の個別のサービスと統合するためにクライアント側のコードの要件が増大する、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、このような問題を解決しやすくなります。場合によっては、完全に問題を解消できることもあります。サーバーレスマイクロサービスパターンを使用すると、それ以降の各マイクロサービスを作成する際の障壁が低くなります (Amazon API Gateway では、既存の API のクローン作成や、他のアカウントでの AWS Lambda 関数の使用も可能です)。このパターンでは、サーバー使用率の最適化を考慮する必要がなくなります。さらに、Amazon API Gateway では、多数の一般的な言語に対応したクライアント SDK をプログラムによって生成し、統合のオーバーヘッドを軽減できます。

まとめ

多層アーキテクチャパターンでは、メンテナンス、疎結合、スケールが容易なアプリケーションコンポーネントを作成するためのベストプラクティスに従うことを推奨します。Amazon API Gateway による統合や、AWS Lambda 内のコンピューティングが発生するようなロジック層を作成する場合、これらの目標の達成に必要な労力を減らしながら、目標を達成できます。これらのサービスは連携して、クライアント用の HTTPS API のフロントエンドと、ビジネスロジックを適用するセキュアな環境を提供します。それと同時に、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なくします。

寄稿者

本書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)
- Stefano Buliani (AWS Mobile、シニアプロダクトマネージャー、テクノロジー)
- Vyom Nagrani (AWS Mobile、シニアプロダクトマネージャー)
- Ajay Nair (AWS Mobile、シニアプロダクトマネージャー)
- Rahul Popat (グローバルソリューションアーキテクト)
- Brajendra Singh (シニアソリューションアーキテクト)

参考資料

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

日付	説明
2021 年 10 月 20 日	新しいサービスの機能とパターンについて更新されました。
2021 年 6 月 1 日	新しいサービスの機能とパターンについて更新されました。
2019 年 9 月 25 日	新しいサービス機能について更新されました。
2015 年 11 月 1 日	初版発行。