AWS ホワイトペーパー

AWS によるサーバーレス多層アーキテクチャ: Amazon API Gateway と AWS Lambda の使用

AWS によるサーバーレス多層アーキテクチャ: Amazon API Gateway と AWS Lambda の使用 : AWS ホワイトペーパー

Table of Contents

AWS によるサーバーレス多層アーキテクチャ: Amazon API Gateway と AWS Lambda の使用

公開日: 2021 年 10 月 20 日 (改訂履歴)

要約

このホワイトペーパーでは、多層アーキテクチャを設計し、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解説します。アーキテクトとデベロッパーは、Amazon API Gateway、AWS Lambda、および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オペレーションのサイクルを短縮できます。

はじめに

多層アプリケーション (3 層、 n 層など) は、何十年もの間、基礎を成すアーキテクチャパターンとして使用されてきました。ユーザー向けアプリケーションのパターンとしては、依然として一般的です。多層アーキテクチャの記述に使用される言語はさまざまですが、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ます。

- プレゼンテーション層: 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 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に変換するために必要なコード (CRUD データベース操作やデータ処理など)。
- 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、疎結合化され別々にスケーリングできるアプリケーションコンポーネントを (多くの場合は別々のチームが) 個別に開発、管理、保守するための一般的なフレームワークを提供します。

このパターンではネットワーク (層と層とのやり取りのためにネットワーク呼び出しを行う必要がある) が層と層の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、多数の画一的なアプリケーションコンポーネントの作成が必要になることが少なくありません。これには、次のようなコンポーネントが該当します。

- 層と層の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ムインターフェイス (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ます。多層アプリケーションに必要なものですが、アプリケーションごとの実装に大きな違いがないコンポーネントです。

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数のサービスを提供しています。これらにより、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伴う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 と、任意のコード関数を実行するためのサービスである 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。

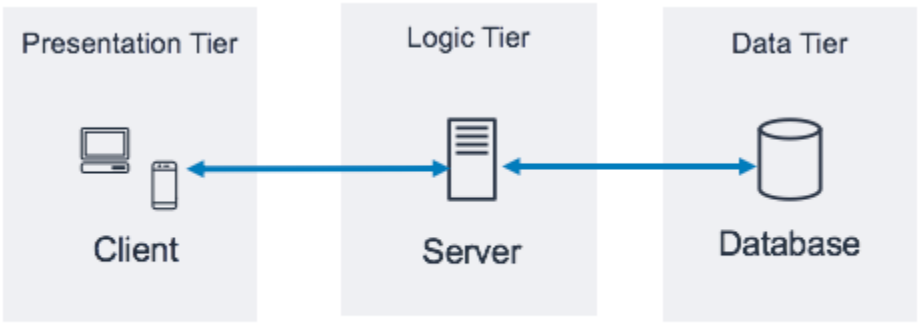
Amazon API Gateway と AWS Lambda との統合により、HTTPS リクエストを通じてユーザー定義のコード関数を直接開始できます。リクエストの量に関係なく、API Gateway と Lambda はどちらも自動的にスケーリングされ、アプリケーションのニーズを正確にサポートします (スケーラビリティに関する情報については、「Amazon API Gateway のクォータと重要な注意点」を参照してください)。これら 2 つのサービスを組み合わせると、アプリケーションにとって重要なコードのみを記述することに集中するための層を作成できます。高可用性を実現するためのアーキテクチャの設計、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の実装に必要となる、他の画一的な作業を行う必要がなくなります。

API Gateway と Lambda により、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 や Amazon Simple Storage Service の使用など) 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されています。

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、1 つ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、詳しく学習できるオンラインリソースが数多くあります。このホワイトペーパーでは、Amazon API Gateway と AWS Lambda を使用してこの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。ここは Amazon API Gateway を使用する場所であり、従来のサーバーベースの実装と比較して、AWS Lambda はさらに大きな効果をもたらす可能性があります。2 つのサービスの機能を組み合わせることで、可用性、拡張性、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。従来のモデルであれば数千台のサーバーを要するアプリケーションであっても、Amazon API Gateway と AWS Lambda を使用すれば、容量を問わずお客様によるサーバー管理が不要になります。さらに、これらのマネージドサービスを組み合わせて使用することで、以下のようなメリットも得られます。

- AWS Lambda:
  - OS の選択、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、スケーリングが不要
  - 過剰プロビジョニングによるコスト発生のリスクを軽減
  - 過少プロビジョニングによるパフォーマンス不足のリスクを軽減

- Amazon API Gateway:
  - API のデプロイ、監視、保護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、サポートされている言語 (Node.js、Python、Ruby、Java、Go、.NET、詳細については「Lambda よくある質問」を参照) で任意のコード関数を実行するためのコンピューティングサービスです。Lambda 関数は隔離されたマネージドコンテナで実行され、イベントに応答して起動されます。このイベントは、AWS によって利用可能となりイベントソースと呼ばれる、プログラムによるトリガーの 1 つです。 すべてのイベントソースについては、「Lambda よくある質問」を参照してください。

AWS Lambda のポピュラーなユースケースの多くは、Simple Storage Service (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、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。Lambda 関数を Amazon API Gateway と組み合わせて使用すると、一般的なウェブサービスの機能を実行できます。Lambda 関数はクライアントの HTTPS リクエストに応答してコードを開始し、API Gateway がロジック層のフロントドアとしての役割を果たして、AWS Lambda がアプリケーションコードを呼び出します。

ビジネスロジックの配置先: サーバーは不要

Lambda では、イベントに応答して実行されるハンドラーというコード関数の記述が必要になります。API Gateway と共に Lambda を使用するには、API に対する 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 を設定します。サーバーレスの多層アーキテクチャでは、必要なビジネスロジックを呼び出す Lambda 関数 (およびその中のハンドラー) と、API Gateway で作成した各 API を統合します。

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を公開するための必要な粒度 (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) を定義できます。Lambda 関数内で、ハンドラーは他の依存関係 (たとえば、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と共にアップロードした他のメソッド) や、他の Lambda 関数にもアクセスできます。

Lambda 関数を作成または更新するには、コードを Zip ファイルの Lambda デプロイパッケージとして Simple Storage Service (Amazon S3) バケットにアップロードするか、コードをすべての依存関係と共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。これらの関数では、AWS マネジメントコンソールの使用や AWS Command Line Interface (AWS CLI) の実行の他、Infrastructure as Code のテンプレートまたはフレームワーク (AWS CloudFormation、AWS Serverless Application Model (AWS SAM)、AWS Cloud Development Kit (AWS CDK) など) の実行といった、さまざまなデプロイ方法を使用できます。これらの方法を使用して関数を作成する際には、デプロイパッケージ内でどの方法をリクエストハンドラーとして使用するかを指定します。複数の Lambda 関数定義で同じデプロイパッケージを再利用することもできます。その場合は、同じデプロイパッケージ内に Lambda 関数ごとに固有のハンドラーを用意します。

Lambda のセキュリティ

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たイベントまたはサービスによってその Lambda 関数を呼び出す必要があります。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開始できない Lambda 関数を作成することもできます。このようなポリシーは、さまざまな AWS サービスにまたが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 Lambda 関数は、その Lambda 関数のデプロイ時に割り当てられた IAM ロールを引き受けます。この IAM ロールには、Lambda 関数がやり取りできる他の AWS サービスやリソース (Amazon DynamoDB、Simple Storage Service (Amazon S3) など) を定義します。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。

機密情報は Lambda 関数内に保存しないでください。IAM は Lambda 実行ロールを通じて 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、環境変数と共に 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、この情報を使用時以外も安全に保管します。

スケールに応じたパフォーマンス

Amazon Elastic Container Registry (Amazon ECR) から、または Simple Storage Service (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取り出されたコードは、AWS が管理する隔離された環境で実行されます。Lambda 関数をスケーリング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 ではコンピューティングフリート内で使用可能なキャパシティが特定され、定義したたランタイム、メモリ、ディスク、タイムアウトの設定でコードが実行されます。このパターンにより、AWS は必要な数だけ関数のコピーを開始できます。

Lambda ベースのロジック層は、お客様のニーズに合わせて常に適切なサイズに調整されます。マネージドスケーリングおよび同時コード開始機能に Lambda の従量課金制を組み合わせることで、トラフィックの急増を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性能に対して料金を支払うことなく、それと同時に、常にお客様の要求に応えることができます。

サーバーレスデプロイと管理

Lambda 関数のデプロイと管理を支援するには、オープンソースフレームワークである AWS サーバーレスアプリケーションモデル (AWS SAM) を使用します。これには、以下が含まれています。

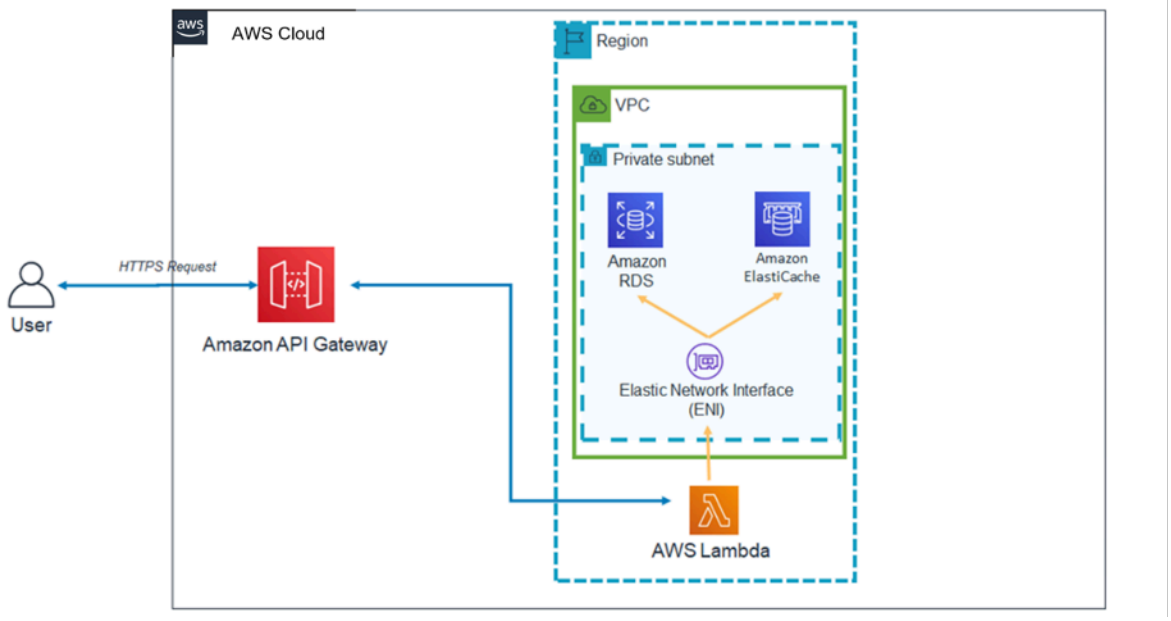
- AWS SAM テンプレート仕様 - アップロードとデプロイを簡素化するために、関数を定義し、その環境、アクセス許可、設定、イベントを記述するために使用される構文。
- AWS SAM CLI - SAM テンプレート構文の検証、ローカルでの関数の呼び出し、Lambda 関数のデバッグ、およびパッケージ関数のデプロイを可能にするコマンド。

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、CloudFormation を通じてプロビジョニングを行うためのソフトウェア開発フレームワークである AWS CDK を使用することもできます。CDK では AWS リソースを定義するための命令型の方法が提供されますが、AWS SAM では宣言型の方法が提供されます。

通常、デプロイした Lambda 関数は、割り当てられた IAM ロールによって定義されているアクセス許可で呼び出され、インターネット接続エンドポイントに到達できます。ロジック層の中核である 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 (プライベートサブネット内で) 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 Lambda 関数からアクセスする必要がある場合は、AWS アカウントの仮想プライベートクラウド (VPC) 内のプライベートサブネットに接続するように Lambda 関数を設定します。関数を VPC に接続すると、Lambda は関数の VPC 設定のサブネットごとに Elastic Network Interface を作成します。Elastic Network Interface は、内部リソースへのプライベートアクセスに使用されます。



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、VPC では、インターネットからデータを操作する手段を、ユーザーが定義した API とユーザーが記述した Lambda コード関数からのリクエストのみに限定できます。

Amazon API Gateway

Amazon API Gateway は、デベロッパーがあらゆる規模で API を作成、公開、保守、モニタリング、保護でき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 API Gateway 経由で公開される API と統合されます。API Gateway を通じて公開された API をサービス指向の多層アーキテクチャに適用できるということは、アプリケーション機能の個々の部分を分離し、この機

能を REST エンドポイントを通じて公開できることを意味します。Amazon API Gateway には、ロジック層に強力な能力を追加できる固有の機能と能力があります。

AWS Lambda との統合

Amazon API Gateway は、REST タイプと HTTP タイプの API の両方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。リソースとは、リソースパスを介してアプリケーションがアクセスできる論理エンティティです (例: `/tickets`)。メソッドは、API リソース (例: `GET /tickets`) に送信される API リクエストに対応します。API Gateway では、各メソッドを Lambda 関数で強化できます。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 によって Lambda 関数が呼び出されます。

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、クライアントの 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 (ステータスコード、ヘッダーなどを含む) がクライアントに直接返されます。

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ユーザーが設定します。また、Lambda 出力をユーザーに返す方法もユーザーが設定します。

> **Note**
>
> API Gateway は、モック統合 (初期のアプリケーション開発に便利) や S3 オブジェクトへのダイレクトプロキシなど、AWS Lambda 外部の追加サーバーレスリソースへのプロキシとして使用することもできます。

リージョンを越えて安定した API パフォーマンスを提供

Amazon API Gateway の各デプロイには、Amazon CloudFront ディストリビューションが含まれています。CloudFront は、API を使用するクライアントの接続ポイントとして Amazon のエッジロケーションのグローバルネットワークを使用するコンテンツ配信サービスです。これにより、API のレスポンスレイテンシーを短縮できます。Amazon CloudFront 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も提供しています。詳細については、ホワイトペーパー「DDoS 攻撃に対するレジリエンスを獲得するための AWS のベストプラクティス」を参照してください。

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。このアプローチにより、反復的な API リクエストに対してパフォーマンス上のメリットが得られるだけでなく、Lambda 関数が呼び出される回数が減り、全体的なコストを削減できます。

組み込み機能によるイノベーションの促進とオーバーヘッドの削減

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、一種の投資です。API Gateway を使用すると、特定の開発タスクに要する時間が短縮され、総開発コストが削減されるため、組織はより自由にさまざまな試みや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提供するために、ロギングとメトリクス収集の実装が軽視されることがよくあります。このため、本番環境で実行中のアプリケーションにこれらの機能をデプロイすると、技術的な負債やオペレーション上のリスクにつなが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。これにより、API Gateway から raw データを収集し、リアルタイムに近い読み取り可能なメトリクスに加工して、API 実行のモニタリングに使用できます。API Gateway は、設定可能なレポートによるアクセスロギングと、デバッグ用の 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の記述が不要であり、中核的なビジネスロジックにリスクを生じることなく、本番環境で実行するアプリケーション用に調整できます。

アプリケーションの全体的な存続期間が不明である場合も、短期間であることがわかっている場合もあります。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は、出発点に API Gateway が提供するマネージド機能が既に含まれていて、API によるリクエストの受信が始まるまでインフラストラクチャコストが発生しないのであれば、より簡単に作成できます。詳細については、Amazon API Gateway の料金を参照してください。

迅速なイテレーションと俊敏性の維持

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し、変化するユーザーベースの要求にすばやく適応できます。

ステージデプロイ

API Gateway に API をデプロイする場合は、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から呼び出すことが

できます。この方法を使用することで、簡単に開発、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デプロイし、ステージ間でデプロイを移動することができます。API をステージにデプロイするたびに、別バージョンの API が作成されます。これは、必要に応じて元に戻すことができます。これらの機能により、新しい機能を個別の API バージョンとしてリリースしながら、既存の機能やクライアントの依存関係をそのまま継続できます。

Lambda との統合の疎結合化

API Gateway の API と Lambda 関数の統合は、API Gateway ステージ変数と Lambda 関数エイリアスを使用して疎結合化できます。これにより、API デプロイを簡素化および高速化できます。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 関数内の特定のエイリアスを指すステージ変数を API で設定できます。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の値を変更すると、API では特定のステージで Lambda エイリアスに対応する Lambda 関数バージョンが実行されます。

Canary リリースのデプロイ

Canary リリースは、新しいバージョンの API をテスト目的でデプロイするソフトウェア開発戦略であり、ベースバージョンについては、同じステージで通常のオペレーション用の本稼働リリースとしてデプロイされた状態が維持されます。Canary リリースのデプロイでは、API の合計トラフィックがランダムに分割され、事前に設定された比率で本稼働リリースと Canary リリースに送られます。API Gateway の API を Canary リリースのデプロイ用に設定すると、限られたユーザー数で新機能をテストできます。

カスタムドメイン名

API Gateway によって提供される URL の代わりに、直感的でビジネスに適した URL 名を API に提供できます。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、マルチレベルの基本パス (`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で、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際には、ロジック層を保護するために Amazon API Gateway に用意されているさまざまな方法を利用できます。

トランジットセキュリティ

API 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 経由で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。公開用 API に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。API ゲートウェイは相互 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ために役立ちます。

API 認証

API の一部として作成した各リソース/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 に認可を追加するには、次の 3 つの一般的な方法を使用できます。

- 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 IAM ポリシーを使用します。同じ認証情報で、必要に応じて他の AWS サービスやリソース (Simple Storage Service (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: カスタム認証スキームを実装する Lambda 関数を定義します。カスタム認証スキームでは、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します。

アクセス制限

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プランとの関連付けがサポートされています。API キーの使用状況は CloudWatch でモニタリングできます。

API Gateway では、API の各メソッドに対するスロットリング、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

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、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、選択した VPC および VPC エンドポイント (複数の AWS アカウントを含む) から API へのアクセスを許可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 へのトラフィックには安全な接続が使用されます。また、パブリックインターネットから隔離され、Amazon ネットワークから外に出ることはありません。

AWS WAF を使用したファイアウォール保護

インターネットに接続する API には、悪意のある攻撃に対する脆弱性があります。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エクスプロイトから、API を保護します。AWS WAF を API Gateway と共に使用すると、API の保護に役立てることができます。

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。Lambda 関数では、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ることにより、どのデータストレージオプションにも接続でき、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) を使用できます。

アプリケーションに適した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。

サーバーレスのデータストレージオプション

Simple Storage Service (Amazon S3) は、業界最高水準のスケーラビリティ、データ可用性、セキュリティ、および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、MySQL および PostgreSQL との互換性があるリレーショナルデータベースであり、クラウド用に構築されています。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。Aurora には、サーバーレス型と従来型の両方の使用モデルが用意されています。

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/ドキュメントデータベースです。マルチリージョンとマルチマスターに対応し、耐久性に優れたフルマネージド型サーバーレスデータベースです。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 日あたり数兆件規模のイベントを、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 デバイスと 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タイプの 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す。信頼された中央機関が所有する、透過的かつ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提供します。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 (Apache Cassandra 用) 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、現在使用しているものと同じ 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。サーバーのプロビジョニング、パッチ適用、管理は必要なく、ソフトウェアのインストール、メンテナンス、オペレーションも必要ありません。Amazon Keyspaces はサーバーレスであるため、料金の支払い対象は使用したリソースのみです。アプリケーショントラフィックに応じて、テーブルのスケールアップおよびスケールダウンがサービスによって自動的に行われます。

Amazon Elastic File System (Amazon EFS) は、シンプルでサーバーレスの伸縮自在な全自動ファイルシステムを提供します。これにより、ストレージのプロビジョニングや管理を行う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せずにオンデマンドでペタバイト単位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 Lambda 関数でマウントできるため、API で使用可能なファイルストレージオプションになります。

非サーバーレスの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 用に最適化されたさまざまなタイプのデータベースインスタンス上で実行することで、リレーショナルデータベースのセットアップ、運用、スケーリングを容易にするマネージド型ウェブサービスです。

Amazon Redshift は、フルマネージド型でペタバイトスケールのクラウド内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。広く使われているオープンソース互換インメモリデータストアのデプロイ、実行、スケール変更をシームレスに実行できます。

Amazon Neptune は、高速かつ信頼性の高いフルマネージド型グラフデータベースサービスです。このサービスでは、高度に接続されたデータセットと連携するアプリケーションを簡単に構築および実行できます。Neptune では、プロパティグラフや W3C Resource Description Framework (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がサポートされており、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、高速かつスケーラブルで、高可用性を備えたフルマネージド型ドキュメントデータベースサービスであり、MongoDB のワークロードに対応しています。

最後に、Amazon EC2 で独立して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層

プレゼンテーション層は、インターネット上に公開された API Gateway の REST エンドポイントを介してロジック層とやり取りする役割を担います。HTTPS 対応のクライアントまたはデバイスはすべて、これらのエンドポイントと通信できるため、さまざまな形態 (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) を柔軟にプレゼンテーション層とすることができます。要件に応じて、プレゼンテーション層では次の AWS サーバーレス製品を使用できます。HTTPS 対応のクライアントまたはデバイスはすべて、これらのエンドポイントと通信できるため、さまざまな形態 (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) を柔軟にプレゼンテーション層とすることができます。要件に応じて、プレゼンテーション層では次の AWS サーバーレス製品を使用できます。

- 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迅速かつ効率的にウェブアプリやモバイルアプリケーションに追加できる、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す。Amazon Cognito は、数百万人規模のユーザー数に対応し、Facebook、Google、Amazon などのソーシャル ID プロバイダーや、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 with CloudFront - 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。ウェブサーバーのプロビジョニングは必要ありません。CloudFront をマネージド型コンテンツ配信ネットワーク (CDN) として使用すると、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 を有効にすることができます。

AWS Amplify はツールとサービスのセットです。それぞれを連携することも、個別で使用することもでき、フロントエンドのウェブ/モバイルデベロッパーが、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するために役立ちます。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。アプリケーションは、世界中に数百の POP (Point Of Presence) を有し、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た、信頼性の高い Amazon の CDN を介して配信されます。Amplify では、一般的なウェブフレームワーク (JavaScript、React、Angular、Vue、Next.js など) やモバイルプラットフォーム (Android、iOS、React Native、Ionic、Flutter など) がサポートされています。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 API をクロスオリジンリソース共有 (CORS) 準拠にする必要がある場合があります。CORS への準拠により、ウェブブラウザは静的なウェブページ内から API を直接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アクセスするための CloudFront ドメイン名 (例: `d2d47p2vcczkh2.cloudfront.net`) が提供されます。Amazon

Route 53 を使用すると、ドメイン名を登録して CloudFront ディストリビューションに転送することも、既にお持ちのドメイン名から CloudFront ディストリビューションに転送することもできます。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対象のサイトにアクセスできます。Route 53 を使用して API Gateway ディストリビューションにカスタムドメイン名を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 API を呼び出すことができます。

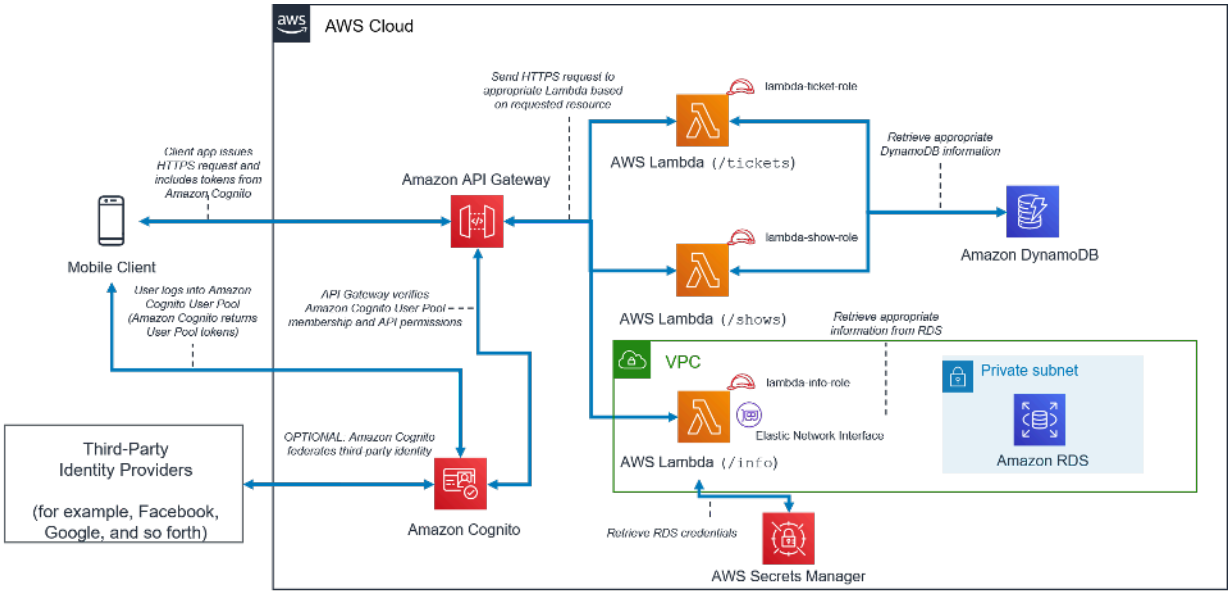
サンプルアーキテクチャパターン

API Gateway と AWS Lambda をロジック層として使用すると、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では、AWS Lambda ベースのロジック層を活用した最も一般的なアーキテクチャパターンをご紹介します。

- モバイルバックエンド – モバイルアプリケーションが 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、プレゼンテーション層のリソース (デスクトップクライアント、EC2 で実行されるウェブサーバーなど) をホストするためにサーバーレス AWS リソースを使用しない一般的な HTTPS クライアントに拡張できます。
- シングルページアプリケーション – Simple Storage Service (Amazon S3) と CloudFront でホストされるシングページアプリケーションは、API Gateway および AWS Lambda と通信し、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 API Gateway と共に AWS Lambda を使用するイベント駆動型の汎用的なウェブアプリケーションバックエンドです。データベースとして Amazon DynamoDB、ユーザー管理用に Amazon Cognito も使用されます。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 の適用性についても説明します。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、よく使用されるパターンです。アプリケーションコンポーネントを疎結合化し、相互に通信する別々のステートレスな機能単位として、これらのコンポーネントをデプロイします。

モバイルバックエンド



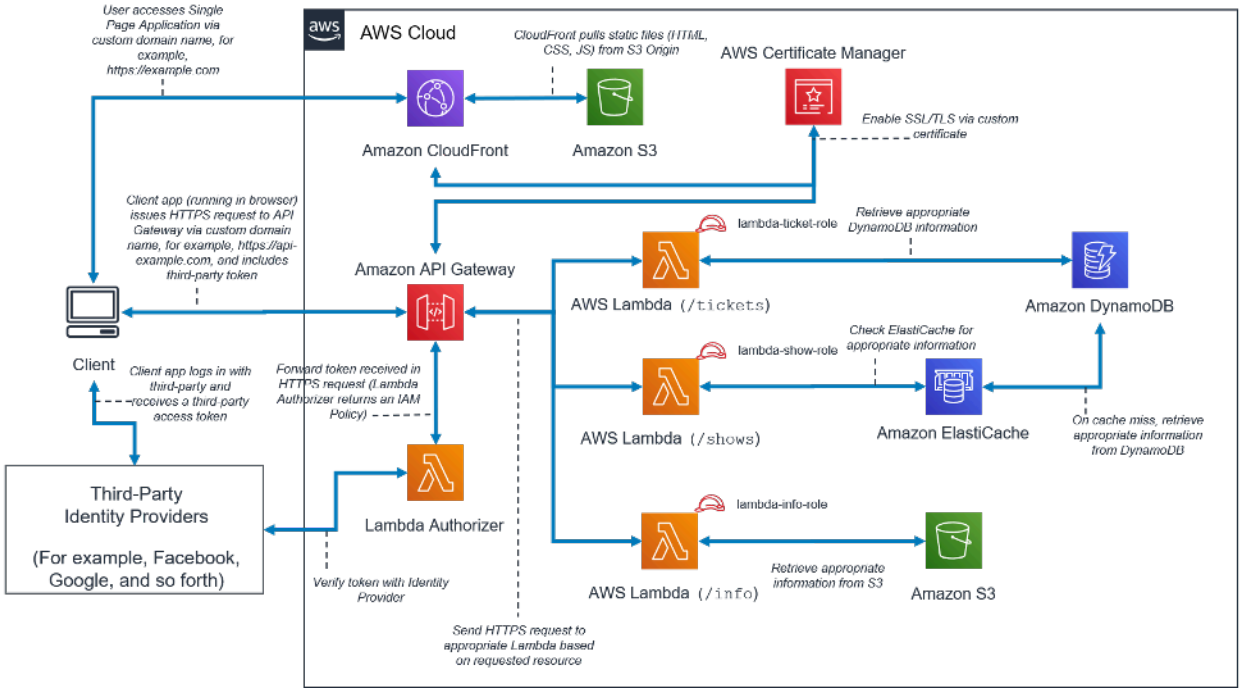
サーバーレスモバイルバックエンドのアーキテクチャパターン

表 1 - モバイルバックエンド層のコンポーネント

層	コンポーネント
プレゼンテーション層	ユーザーデバイスで実行されるモバイルアプリケーション。
ロジック層	Amazon API Gateway と AWS Lambda。 このアーキテクチャでは、3 つの公開されたサービス (`/tickets`、`/shows`、`/info`) が示されています。API Gateway エンドポイントは、Amazon Cognito ユーザープール によって保護されています。この方法では、ユーザーが 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 呼び出しの承認に使用されるアクセスと ID トークンを受け取ります。

層	コンポーネント
	各 Lambda 関数には独自の Identity and Access Management (IAM) ロールが割り当てられ、適切なデータソースへのアクセスを提供します。
データ層	DynamoDB は、`/tickets` および `/shows` の各サービスに使用されます。 `/info` サービスには Amazon RDS が使用されます。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。

シングルページアプリケーション



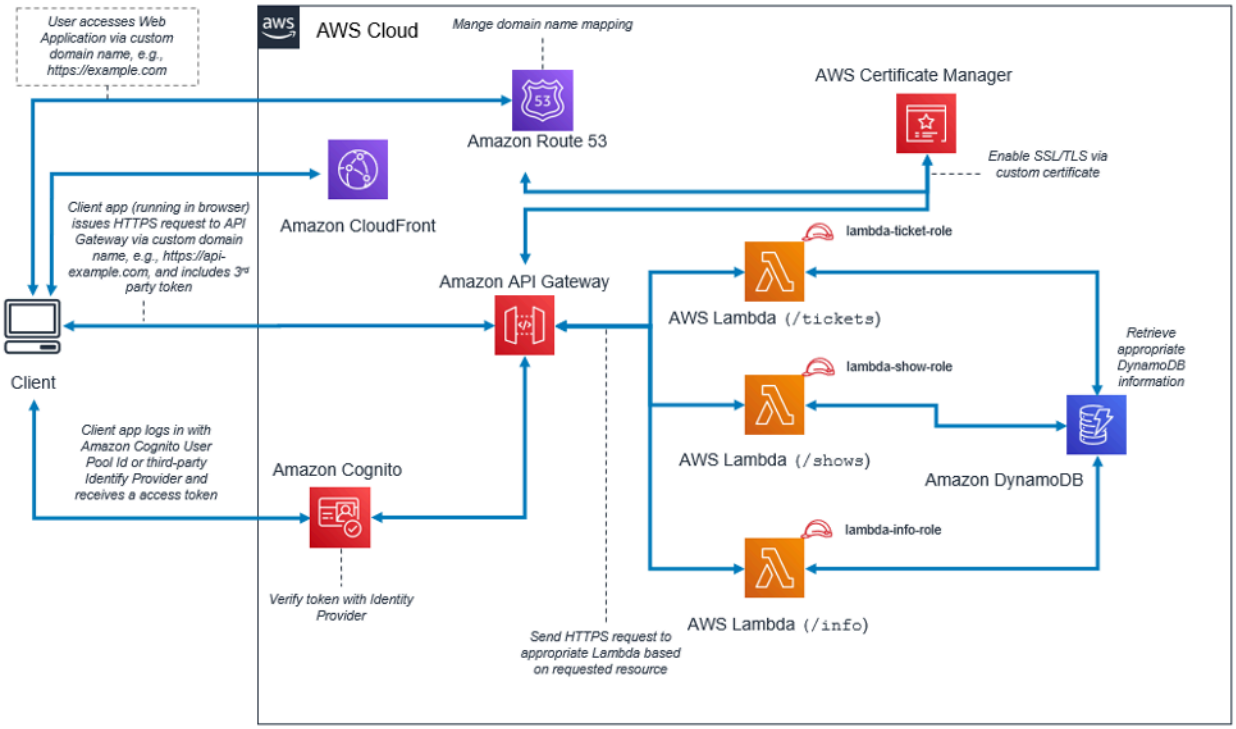
サーバーレスシングルページアプリケーションのアーキテクチャパターン

表 2 - シングルページアプリケーションのコンポーネント

層	コンポーネント
プレゼンテーション層	Simple Storage Service (Amazon S3) でホストされ、CloudFront によって配布される静的ウェブサイトコンテンツ。 AWS Certificate Manager では、カスタムの SSL/TLS 証明書を使用できます。
ロジック層	API Gateway と AWS Lambda。 このアーキテクチャでは、3 つの公開されたサービス (`/tickets`、`/shows`、`/info`) が示されています。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 プロバイダーを通じてサインインし、アクセストークンおよび ID トークンを取得します。これらのトークンは API Gateway 呼び出しに含まれており、Lambda オーソライザーはこれらのトークンを検証して、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 関数には独自の IAM ロールが割り当てられ、適切なデータソースへのアクセスを提供します。
データ層	Amazon DynamoDB は、`/tickets` および `/shows` の各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。

層	コンポーネント
	Simple Storage Service (Amazon S3) は、`/info service` が使用す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



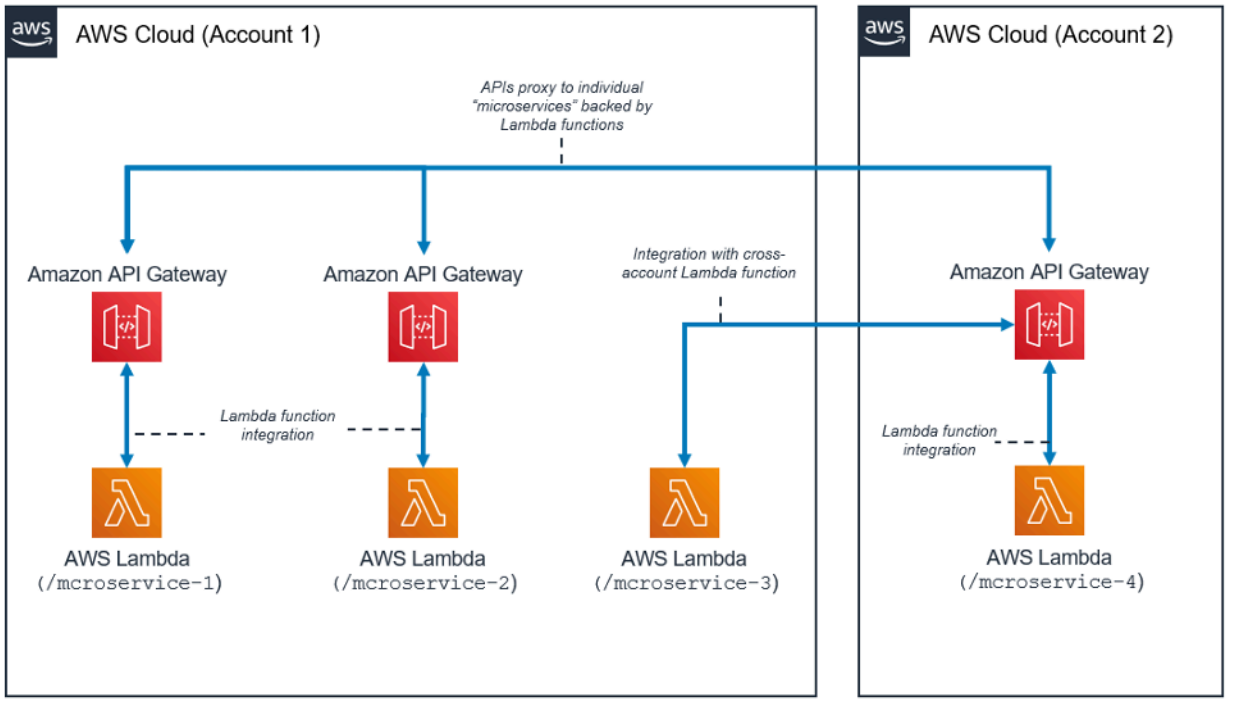
ウェブアプリケーションのアーキテクチャパターン

表 3 - ウェブアプリケーションのコンポーネント

層	コンポーネント
プレゼンテーション層	フロントエンドアプリケーションはすべて、create-react-app などの React ユーティリティによって生成される静的コンテンツ (HTML、CSS、JavaScript、画像) です。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

層	コンポーネント
	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されます。
ロジック層	ロジック層は、API Gateway REST API を代表する Lambda 関数を使用して構築されます。 このアーキテクチャでは、複数の公開されたサービスが示されています。複数の異なる Lambda 関数があり、それぞれがアプリケーションの異なる側面を処理します。Lambda 関数は API Gateway の背後にあり、これには 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 では、Amazon Cognito とのアウトオブボックス統合が使用されます。ユーザーが認証された場合のみ、クライアントが JSON Web Token (JWT) トークンを受け取ります。このトークンは API コールで使用されます。 各 Lambda 関数には独自の IAM ロールが割り当てられ、適切なデータソースへのアクセスを提供します。
データ層	この例では、DynamoDB をデータストレージに使用していますが、ユースケースや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することもできます。

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限定されない一般的なパターンです。このパターンでは、サーバーレスリソースの使用により、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疎結合化され、別々にデプロイおよび運用されます。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、続いて AWS Lambda によって起動される関数のみです。これらのサービスを使用すれば、環境の疎結合化を行い、必要な粒度に分離することができます。

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し生じるオーバーヘッド、サーバーの密度/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。

このような問題は、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、解決しやすくなります。場合によっては、完全に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、追加のマイクロサービスを作成しやすくなります (API Gateway では既存の API のクローンを作成することも、他のアカウントで Lambda 関数を使用することもできます)。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、管理、疎結合化、スケーリング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ています。API Gateway による統合が生じ、コンピューティングが AWS Lambda 内で行われるようなロジック層を作成する場合、この目標の達成に必要な作業量を減らしながら、目標の達成に向かうことができます。これらのサービスを組み合わせることで、クライアントに HTTPS API フロントエンドを提供し、ビジネスロジックを適用する安全な環境を実現できます。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で生じるオーバーヘッドは不要になります。

寄稿者

本書の作成における寄稿者

- AWS ソリューションアーキテクト、Andrew Baird
- AWS ProServe コンサルタント、Bryant Bost
- AWS Mobile、テクノロジー、シニアプロダクトマネージャー、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
- シニアソリューションアーキテクト、Brajendra Singh

その他の資料

詳細については、以下を参照してください。

- AWS ホワイトペーパーとガイド

改訂履歴

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 フィードをサブスクライブしてください。

update-history-change	update-history-description	update-history-date
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に合わせて更新しました。	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に合わせて更新しました。	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に合わせて更新しました。	2019 年 9 月 25 日
初版公開	ホワイトペーパーを公開しました。	2015 年 11 月 1 日

注意

お客様は、この文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、明示または暗示を問わず、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 契約により規定されます。本書は、AWS とお客様の間で締結されるいかなる契約の一部でもなく、その内容を修正するものでもありません。