AWS サーバレス多層アーキテクチャ

Amazon API Gateway と AWS Lambda の使用

2015 年 *11* 月

注意

本書は、情報提供の目的のみのために提供されるものです。本書の発行時点における AWS の現行製品と慣行を表したものであり、それらは予告なく変更されることがあります。お客様は本書の情報および AWS 製品の使用について独自に評価する責任を負うものとします。これらの情報は、明示または黙示を問わずいかなる保証も伴うことなく、「現状のまま」提供されるものです。本書のいかなる内容も、AWS、その関係者、サプライヤー、またはライセンサーからの保証、表明、契約的責任、条件や確約を意味するものではありません。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では、アマゾン ウェブ サービス (AWS) のイノベーションによって、マイクロサービス、モバイルバックエンド、公開ウェブサイトなどの革新的技術方法を変更する方法などの一般的なパターンに関する多層アーキテクチャの設計がどのように変化するかを説明します。アーキテクトおよび開発者は、Amazon API Gateway と AWS Lambda などの実装パターンを使用できるようになりました。これにより、多層アプリケーションの作成と運用管理に必要な開発と運用のサイクルを短縮することができます。

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、基礎となるアーキテクチャパターンでした。多層パターンは、アプリケーションコンポーネントを分離および拡張でき、(多くの場合、別々のチームで) 個別に管理および保守できるようにするためのガイドラインとして優れています。多層アプリケーションは、ウェブサービスを使用するためのサービス指向アーキテクチャ (SOA) アプローチを使用して構築されることがよくあります。このアプローチでは、ネットワークが各層の間の境界として機能します。ただし、アプリケーションの一部としての新しいウェブサービス層の作成には、未分化要素が多数あります。多層ウェブアプリケーション内に記述されたコードの多くは、パターン自体の直接的な結果です。例として、1 つの層を別の層に統合するコード、層と層の間をやりとりするために各層で使用される API とデータモデルを定義したコード、層と層の統合ポイントが望ましくない方法で公開されないようにするためのセキュリティ関連コードなどがあります。

Amazon API Gateway[1] は、API を作成および管理するためのサービスであり、AWS Lambda[2] は、任意のコードを実行するためのサービスです。これらを一緒に使用することで、堅牢な多層アプリケーションを簡単に作成することができます。

Amazon API Gateway の AWS Lambda との統合により、ユーザー定義の HTTPS リクエスト を介してユーザー定義のコード関数を直接トリガーすることができます。要求されたリクエストのボリュームに関係なく、API Gateway および Lambda は、アプリケーションのニーズに正確に対応できるように自動的にスケールされます。これらの組み合わせにより、アプリケーション用の層を作成し、アプリケーションにとって重要なコードを記述することができます。多層アーキテクチャーの実装に関する他の様々な未分化要素 (高可用性のための設計、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) 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) に焦点を当てません。

最近、AWS は、お客様の Amazon Virtual Private Cloud (Amazon VPC)[3] 内で実行される Lambda 関数を作成する機能を発表しました。この機能では、API Gateway と Lambda の組み合わせによるメリットが拡張され、ネットワークプライバシーが必要とされるさまざまなユースケースに対応できるようになりました。たとえば、機密情報が格納されているリレーショナルデータベースをウェブサービスに統合する必要があるような場合です。Lambda との統合 Amazon VPC により、間接的に Amazon API Gateway の機能も拡張されました。Amazon VPC の一部としてプライベートかつ安全に保たれるバックエンドの前面で、インターネットアクセスが可能な HTTPS API を開発者が独自に定義することができるためです。この強力なパターンの利点は、多層アーキテクチャの各層にわたって見ることができます。このホワイト

ペーパーでは、最も一般的な多層アーキテクチャの例である **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、典型的な 3 層ウェブアプリケーションを超えて適用することができます。

3 層アーキテクチャの概要

3 層アーキテクチャは、ユーザ向けのアプリケーションで一般的なパターンです。このアーキテクチャを構成する各層は、**プレゼンテーション層**、**ロジック層**、および**データ層**です。プレゼンテーション層は、ユーザが直接操作するコンポーネント (ウェブページ、モバイルアプリ UI など) を表します。ロジック層には、プレゼンテーション層におけるユーザーの操作を解釈し、アプリケーションの動作を駆動する機能に変換するために必要なコードが含まれています。データ層は、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) から構成されています。図 1 は、シンプルな 3 層アプリケーションの例を示しています。

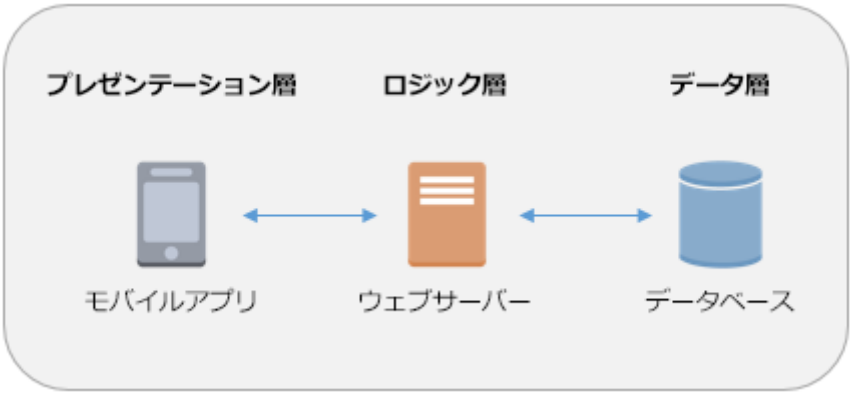

図 1: シンプルな 3 層アプリケーションのアーキテクチャパターン

一般的な 3 層アーキテクチャパターンの詳細について学ぶことができる多数の優れたリソースがオンラインで公開されています。このホワイトペーパーでは、Amazon API Gateway と AWS Lambda を使用する、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。

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を表します。このため、Amazon API Gateway と AWS Lambda を統合してロジック層を形成することは、非常に画期的です。この 2 つのサービスの機能により、可用性、拡張性、安全性に優れた本稼働アプリケーションをサーバーレスで構築することができます。アプリケーションは数千台のサーバーを使用することもありますが、このパターンを使用することで、サーバーの管理が一切不要になります。また、これらのマネージド型サービスを一緒に使用することで、次のようなメリットを得ることができます。

- オペレーティングシステムの選定、保護、パッチ、管理が必要ありません。
- サーバーのサイズ調整、監視、スケールアウトが必要ありません。
- オーバープロビジョニングによるコストのリスクがありません。
- アンダープロビジョニングによるパフォーマンスのリスクがありません。

さらに、各サービス内には、多層アーキテクチャパターンに利点をもたらす機能があります。

Amazon API Gateway

Amazon API Gateway は、API を定義、展開、および保守するための、完全マネージド型のサービスです。クライアントは標準的な HTTPS リクエストを使用して API と統合されます。サービス指向の多層アーキテクチャに対する適用性は明らかです。一方で、ロジック層での大きな強みとなる機能と性質も備えています。

AWS Lambda との統合

Amazon API Gateway を使用することで、アプリケーションは、簡単な方法 (HTTPS リクエスト) で AWS Lambda のイノベーションを直接活用することができます。API Gateway は、AWS Lambda で作成した関数とプレゼンテーション層をつなぐブリッジになります。API を使用したクライアントとサーバーの関係を定義した後、クライアントの HTTPS リクエストの内容が Lambda 関数に渡され、実行されます。この内容には、リクエストのメタデータ、リクエストヘッダー、リクエスト本文が含まれています。

全世界で安定した API パフォーマンス

Amazon API Gateway の各デプロイメントには、Amazon CloudFront[4] ディストリビューションが含まれています。Amazon CloudFront は、お客様の API に統合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したコンテンツ配信ウェブサービスです。これは、API の合計応答時間レイテンシーの短縮に役立ちます。Amazon CloudFront では、世界中の複数のエッジロケーションを使用することにより、分散サービス妨害 (DDoS) 攻撃への対処も可能になります。詳細については、ホワイトペーパー DDoS 攻撃に対処するための AWS の ベストプラクティス [5]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応答を格納して、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。この方法は、繰り返し発生する API リクエストのパフォーマンス上のメリットが得られるだけではなく、バックエンドの実行回数を減らすことができるため、全体的なコストを削減できます。

イノベーションの促進

新しいアプリケーションを構築するために必要な開発作業は、投資です。プロジェクトを開始するには、それを正当化する必要があります。開発のタスクと時間に必要な投資金額を削減すると、もっと自由に試行錯誤し、新しいことを取り入れる余裕が生まれます。

多層ウェブサービスをベースにした多くのアプリケーションの場合、プレゼンテーション層は容易に複数のユーザー用 (個別のモバイルデバイス、ウェブブラウザなど) に断片化されます。また、これらのユーザーは地理的な結びつきがない場合が多くあります。ただし、分離型のロジック層は、ユーザーによってブル知的に断片化されるわけではありません。すべてのユーザーは、ロジック層を実行する同じインフラストラクチャに依存するため、インフラストラクチャの重要性が拡大します。最初にロジック層を実装する際の手抜き (「運用当初に数値を測定する必要はありません」、「初期使用率は低くなるため、スケーリング方法は後で考えましょう」など) が、新しいアプリケーションを早く提供するためのメカニズムとして提案することが少なくありません。本稼働環境で既に実行中のアプリケーションを変更する必要が生じた場合、これは技術的な負債や運用リスクとなる場合があります。*Amazon API Gateway* を使用すると、既にサービスが実装されているため、コストを削減し、迅速に提供することが可能になります。

全体的なアプリケーションの寿命は、不明であるか短命であると判明しているかのどちらかです。そのため、新しい多層アプリケーションの検討資料を作成することは困難な場合があります。Amazon API Gateway から提供されるマネージド型機能が開始地点で既に含まれていれば、その作業が容易になります。また、インフラストラクチャコストが発生は、お客様の API がリクエストを受け取り初めてから発生します。詳細については、Amazon API Gateway 料金表を参照してください。[6]

迅速に反復して俊敏性を維持

新しいアプリケーションでは、ユーザーベースの定義 (サイズ、使用パターンなど) が十分でない場合があります。そのため、ユーザーベースが形になっても、ロジック層には機敏性を維持する必要があります。アプリケーションとビジネスは、早期導入ユーザーからの変化し続ける要望に対応し、変化する必要があります。Amazon API Gateway を使用すると、API を開始からデプロイに勧めるために必要な開発サイクルの数を減少させることができます。Amazon API Gateway では、Mock 統合 [7]を作成できるため、API 応答を 直接 API Gateway から作成できます。この API Gateway に対してバックエンドロジック全体を開発しながら、並行してクライアントアプリケーションを開発することができます。この利点は、API の最初のデプロイ時だけでなく、ビジネス上の決定によりユーザーに応じてアプリケーション (および既存の API) を迅速に方向転換することになった場合にも当てはまります。API Gateway と AWS Lambda ではバージョニングが可能であるため、独立した API/関数バージョンとして新しい機能をリリースする場合も、既存の機能とクライアントの依存関係はそのままにしておくことができます。

セキュリティ

3 層で構成される公開ウェブアプリケーションのロジック層をウェブサービスとして実装すると、すぐにセキュリティがトピックに挙がります。アプリケーションでは、許可されたクライアントのみが (ネットワークで公開される) ロジック層にアクセスできるようにする必要があります。Amazon API Gateway では、バックエンドの安全性を確認できる方法を通じてセキュリティのトピックに取り組んでいます。アクセス制御については、クライアントアプリケーションに静的な API キーの文字列を提供する方法に頼らないでください。クライアントによって抽出され、別の場所で使用される可能性があります。Amazon API Gateway によってロジック層を保護できる方法がいくつかあり、これらを利用することができます。

- 転送中の暗号化を可能にするために、API へのすべてのリクエストには HTTPS を使用することができます。
- 信頼関係が Amazon API Gateway 内の特定の API と AWS Lambda 内の特定の関数の間だけに限定されるように、AWS Lambda 関数はアクセスを制限することができます。Lambda 関数の起動には、その関数の公開のために選択した API を使用する以外の方法がなくなります。
- Amazon API Gateway を使用すると、API に統合するためのクライアント SDK を生成することができます。この SDK では、API で認証が必要とされる場合にはリクエストの署名も管理します。クライアント側で認証に使用される API 認証情報は、直接

AWS Lambda 関数に渡されるため、作成した独自のコード内で必要に応じてさらに認証を行うことができます。

- 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、独自の特定 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)[8] ポリシー内で参照することができます。
  - これは、他の AWS 所有の API と同様に、お客様の API も最重要として扱われることを意味します。IAM ポリシーはきめ細かく指定することができます。IAM ポリシーでは、Amazon API Gateway を使用して作成した API の特定のリソース/メソッドを個別に参照できます。
  - アプリケーションコードのコンテクスト外で作成した IAM ポリシーでは、API アクセスが強制されます。つまり、このようなアクセスレベルを認識または強制するためのコードを記述する必要はありません。コードが存在しないため、コードに含まれるバグや悪用を心配する必要もありません。
  - AWS 署名バージョン 4 (SigV4)[9] を使用した認証と IAM ポリシーで API アクセスのためのクライアント認証を行うと、必要に応じて、同じ認証情報で他の AWS 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。

AWS Lambda

AWS Lambda は、本質的には、サポートされている任意の言語 (2015 年 11 月時点では Node、JVM ベース、Python) で記述された任意のコードに対し、イベントへの応答としてトリガーされることを許可しています。このイベントには、AWS によって利用可能となる、プログラムによるトリガーの 1 つを使用できます。これは、**イベントソース**と呼ばれるものです (現在サポートされているイベントソースについては、こちらを参照してください [10])。AWS Lambda に関する一般的な多くのユースケースは、イベント駆動型データ処理ワークフローに従って展開されます。その例として、Amazon Simple Storage Service (Amazon S3)[11] に格納されているファイルの処理や、Amazon Kinesis[12] 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 との組み合わせで使用すると、AWS Lambda 関数は、典型的なウェブサービスのコンテキストに存在させることができ、HTTPS リクエストによって直接トリガーすることができます。Amazon API Gateway は、ロジック層の正面玄関として機能しますが、ロジックはこれらの API の背後で実行する必要があります。ここで AWS Lambda が必要になります。

ビジネスロジックの適用

AWS Lambda では、イベントによるトリガー時に実行される、**ハンドラー**と呼ばれるコード関数を記述できます。たとえば、API への HTTPS リクエストなどのイベントが発生したときにトリガーされるハンドラーを記述することができます。Lambda では、必要な粒度レベル (1 つは API 単位、1 つは API メソッド単位、など) でモジュール式のハンドラーを作成し、個別に更新、呼び出し、変更を行うことができます。ハンドラーは、自身が持つ他の依存関係 (独自のコードでアップロードした他の関数、ライブラリ、ネイティブバイナリ、外部ウェブサービスなど) にも自由に到達できます。Lambda を使用すると、すべての必要な依存関係を (作成時に) 関数定義にパッケージ化できます。関数を作成するとき、デプロイパッケージ内でどのメソッドをハンドラーとして使用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 関数定義に再利用することもできます。この場合、同じデプロイパッケージ内の各 Lambda 関数に、それぞれ異なるハンドラーを指定することもできます。サーバーレス多層アーキテクチャパターンでは、Amazon API Gateway で作成した個々の API が、必要なビジネスロジックを実行する Lambda 関数 (および対応するハンドラー) に統合されます。

Amazon VPC との統合

ロジック層のコアである AWS Lambda は、データ層と直接統合されるコンポーネントになります。データ層にはビジネスまたはユーザーの機密情報が含まれている場合があるため、データ層は厳密に保護する必要があります。Lambda 関数から統合できる AWS サービスの場合は、IAM ポリシーを使用してアクセス制御を管理することができます。これに該当するサービスには、Amazon S3、Amazon DynamoDB、Amazon Kinesis、Amazon Simple Queue Service (Amazon SQS)、Amazon Simple Notification Service (Amazon SNS)、その他の AWS Lambda 関数などがあります。ただし、リレーショナルデータベースなど、独自のアクセス制御が使用されるコンポーネントもあります。このようなコンポーネントの場合は、プライベートなネットワーク環境である Amazon Virtual Private Cloud (Amazon VPC)[13] にデプロイすることで、より高いセキュリティを確保できます。

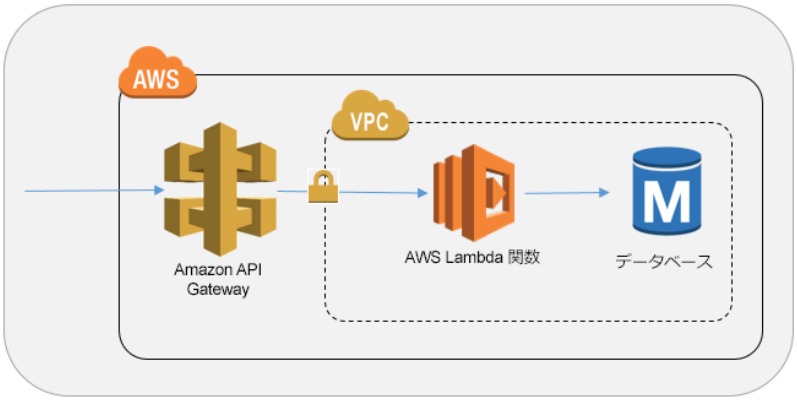

図 2: VPC を使用するアーキテクチャパターン

VPC を使用するということは、ビジネスロジックが依存しているデータベースやその他のストレージメディアに対して、インターネット経由のアクセスを不可能にすることができるということです。また、VPC を使用すると、インターネット経由でデータを操作する方法は、独自に定義した API と独自に作成した Lambda コード関数を経由する以外になくなります。

セキュリティ

Lambda 関数を実行するには、イベントまたはサービスによってトリガーする必要があり、その動作が IAM ポリシーで許可されている必要があります。独自に定義した API Gateway リクエストから呼び出されない限り、まったく実行できない Lambda 関数を作成することもできます。独自のコードは、作成した API で定義される、有効なユースケースの一部として処理されます。

各 Lambda 関数には、IAM 信頼関係を介して付与される権限である IAM ロールが設定されます。この IAM ロールでは、Lambda 関数が操作できる他の AWS サービス (Amazon DynamoDB テーブルや Amazon S3 バケットなど) が定義されます。独自の関数からアクセスできるサービスは、関数自体とは別の場所で定義および制御されます。これは小さなことですが、効果は強力です。これにより、作成するコードで AWS 認証情報を格納または取得する必要がなくなります。つまり、API キーをハードコーディングする必要がなく、API キーを取得してメモリ内に格納するためのコードを記述する必要もないということです。IAM ロールで呼び出しが許可されているサービスを Lambda 関数からの呼び出すことを可能にするかどうかは、サービス自体で管理されます。

データ層

ロジック層として AWS Lambda を使用する場合は、多数のデータストレージオプションから選択してデータ層に使用できます。これらのオプションは、Amazon VPC でホストされているデータおよび IAM 対応のデータストアという 2 つのカテゴリに大きく分けられます。AWS Lambda は、どちらにも安全に統合することができます。

Amazon VPC でホストされているデータストア

AWS Lambda と Amazon VPC の統合により、プライベートかつセキュアな方法でさまざまなデータストレージテクノロジーと統合するための機能を利用できるようになります。

- Amazon RDS[14]

  Amazon Relational Database Service (Amazon RDS) によって利用可能になるエンジンを使用することもできます。Amazon RDS には、Lambda で記述したコードから直接接続できます。この点は Lambda 以外での操作と同様ですが、データーベース認証情報の暗号化のために、AWS Key Management Service (AWS KMS) とシンプル荷統合できるという利点があることが異なります。

- Amazon ElastiCache[15]

  マネージド型のインメモリキャッシュと Lambda 関数を統合し、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。

- Amazon RedShift[16]

  エンタープライズデータウェアハウスへの照会を安全に行うための関数を作成できます。これは、レポートまたはダッシュボードを構築する場合や、アドホッククエリの結果を取得する場合に利用できます。

- Amazon Elastic Compute Cloud (Amazon EC2)[17] でホストされているプライベートウェブサービス

  VPC 内のプライベートな環境でウェブサービスを実行する既存のアプリケーションがある場合、Lambda 関数から、論理的にプライベートな VPC ネットワークを越えて HTTP リクエストを作成できます。

IAM 対応のデータストア

AWS Lambda は、IAM と統合されているため、IAM を使用して任意の AWS サービスと安全に統合し、AWS API を使用してサービスを直接活用することができます。

- Amazon DynamoDB[18]

  Amazon DynamoDB は、無限にスケーラブルな AWS の NoSQL データベースです。規模にかかわらず 数ミリ秒のパフォーマンスでデータレコード (本書の執筆時点では 400 KB 以下) を取得するには、Amazon DynamoDB の使用を検討してください。Amazon DynamoDB のきめ細やかなアクセス制御を使用すると、DynamoDB 内の特定のデータを照会する際に最小権限を使用するというベストプラクティスを Lambda 関数に適用できます。

- Amazon S3[19]

  Amazon Simple Storage Service (S3) を利用すると、インターネット規模のオブジェクトストレージを使用できるようになります。Amazon S3 は、オブジェクトに対する 99.999999999% の耐久性を実現するように設計されています。低コストで高耐久性のストレージがアプリケーションに必要な場合は、Amazon S3 の使用を検討してください。ま、Amazon S3 は、年間で最大 99.99% のオブジェクト可用性を実現するよう設計されています。可用性の高いストレージを必要とする場合も、使用を検討してください。Amazon S3 に格納されているオブジェクト (ファイル、イメージ、ログ、任意のバイナリデータ) には、HTTP 経由で直接アクセスできます。Lambda 関数 は、仮想プライベートエンドポイントを経由して安全に Amazon S3 と通信できます。S3 内のデータは、Lambda 関数に関連付けられている IAM ポリシーのみに制限できます。

- Amazon Elasticsearch Service[20]

  Amazon Elasticsearch Service (Amazon ES) は、一般的な検索/分析エンジンである Elasticsearch のマネージドバージョンです。Amazon ES は、クラスターのマネージド型プロビジョニング、障害検出、ノードの置換を処理します。Amazon ES API へのアクセスは、IAM ポリシーを使用して制御できます。

プレゼンテーション層

Amazon API Gateway により、プレゼンテーション層のさまざまな可能性が生まれます。HTTPS 通信が可能なクライアントであれば、インターネットアクセスが可能な HTTPS API を利用できます。アプリケーションのプレゼンテーション層用に使用を検討できる一般的な例を以下に示します。

- モバイルアプリ: Amazon API Gateway および AWS Lambda を通じてカスタムビジネスロジックと統合できることに加え、ユーザー ID の作成と管理のメカニズムとして Amazon Cognito[21] を使用できます。
- 静的ウェブサイトコンテンツ (Amazon S3 でホストされているファイルなど): Amazon API Gateway の API を Cross-Origin Resource Sharing (CORS) 対応にすることができます。これにより、静的なウェブページ内でウェブブラウザから直接 API 呼び出すことが可能になります。
- その他の HTTPS 対応クライアントデバイス: インターネットに接続された多くのデバイスは、HTTPS を使用した通信に対応しています。これは、Amazon API Gateway を使用して作成した API とクライアントが通信する方法における独自のものではなく、純粋な HTTPS です。特殊なクライアントソフトウェアやライセンスは必要ありません。

アーキテクチャパターン例

ロジック層を形成する接着剤として Amazon API Gateway と AWS Lambda を使用して、以下に示す一般的なアーキテクチャパターンを実装できます。それぞれの例では、ユーザーが独自のインフラストラクチャを管理する必要のない AWS サービスのみが使用されています。

モバイルバックエンド

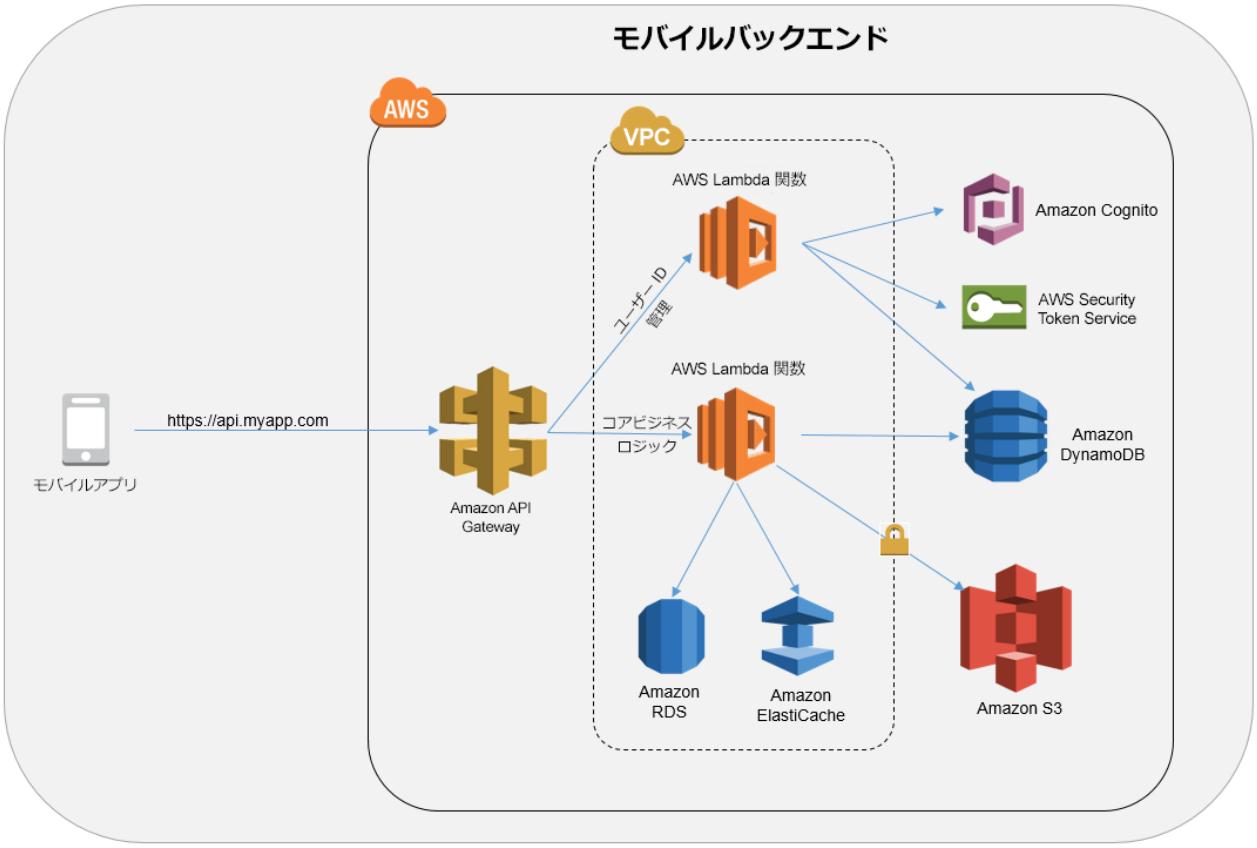

図 3: モバイルバックエンドのためのアーキテクチャパターン

- **プレゼンテーション層:** 各ユーザーのスマートフォン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。ロジック層は、各 Amazon API Gateway の一部として作成された Amazon CloudFront ディストリビューションによってグローバルに配布されます。Lambda 関数としては、Amazon Cognito で管理される、ユーザー/デバイスの ID 管理と認証に固有のものがいくつか考えられます。Amazon Cognito は、一時的なユーザーアクセス認証情報用に IAM と統合されます。一般的なサードパーティの ID プロバイダーとも統合できます。そのほかの Lambda 関数でモバイルバックエンド用にコアビジネスロジックを定義できます。
- **データ層:** 必要に応じて、さまざまなデータストレージサービスを活用できます。本書に記載されているオプションを利用できます。

Amazon S3 でホストされているウェブサイト

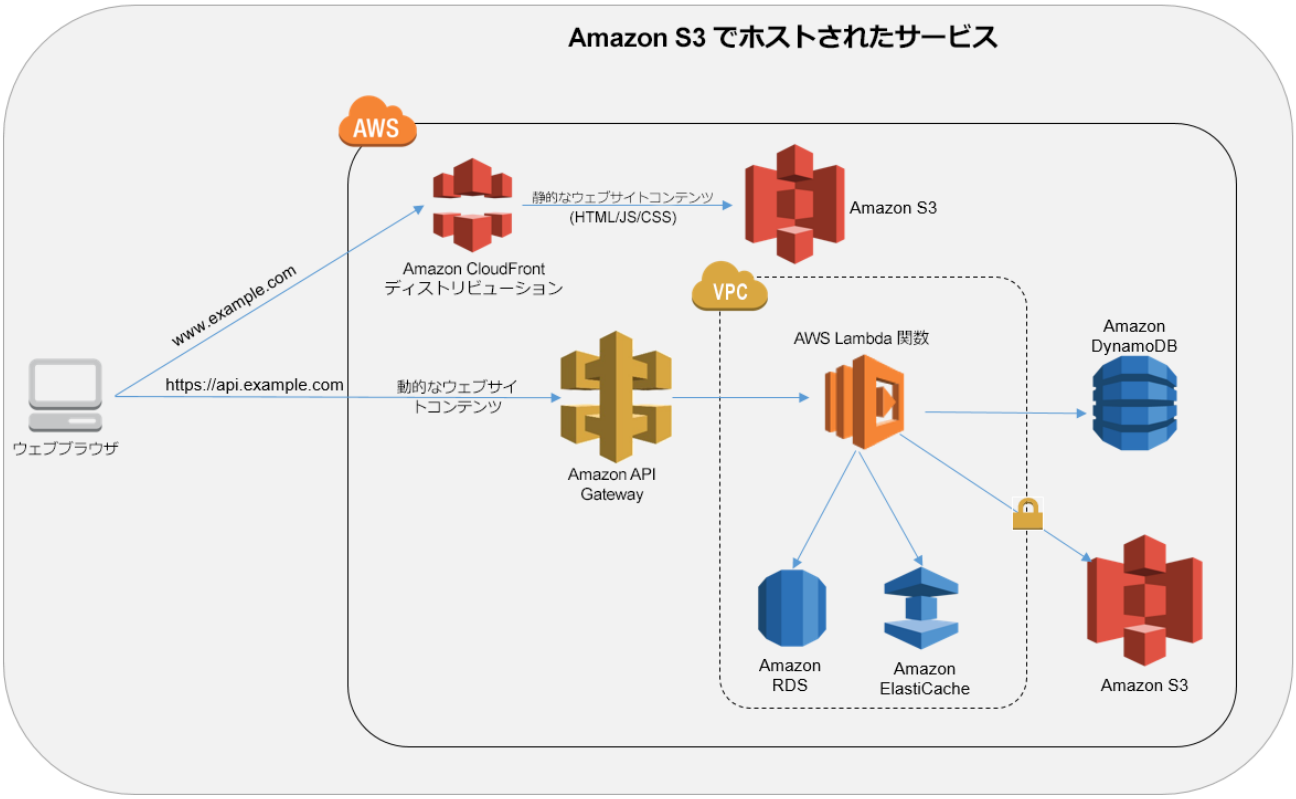

図 4: Amazon S3 でホストされる静的なウェブサイト用のアーキテクチャパターン

- **プレゼンテーション層:** 静的なコンテンツは Amazon S3 でホストされ、Amazon CloudFront によって配布されます。静的なウェブサイトを Amazon S3 でホストすると、サーバーベースのインフラストラクチャでコンテンツをホストする場合よりも優れたコスト効率を実現することができます。ただし、ウェブサイトに豊富な機能を含めるには、静的なコンテンツと動的なバックエンドの統合が必要となる場合があります。
- **ロジック層:** Amazon API Gateway と AWS Lambda。Amazon S3 にホストされている静的ウェブコンテンツは、CORS 対応が可能な Amazon API Gateway と直接統合できます。
- **データ層:** 必要に応じて、さまざまなデータストレージサービスを活用することができます。本書に記載されているオプションを使用できます。

マイクロサービス環境

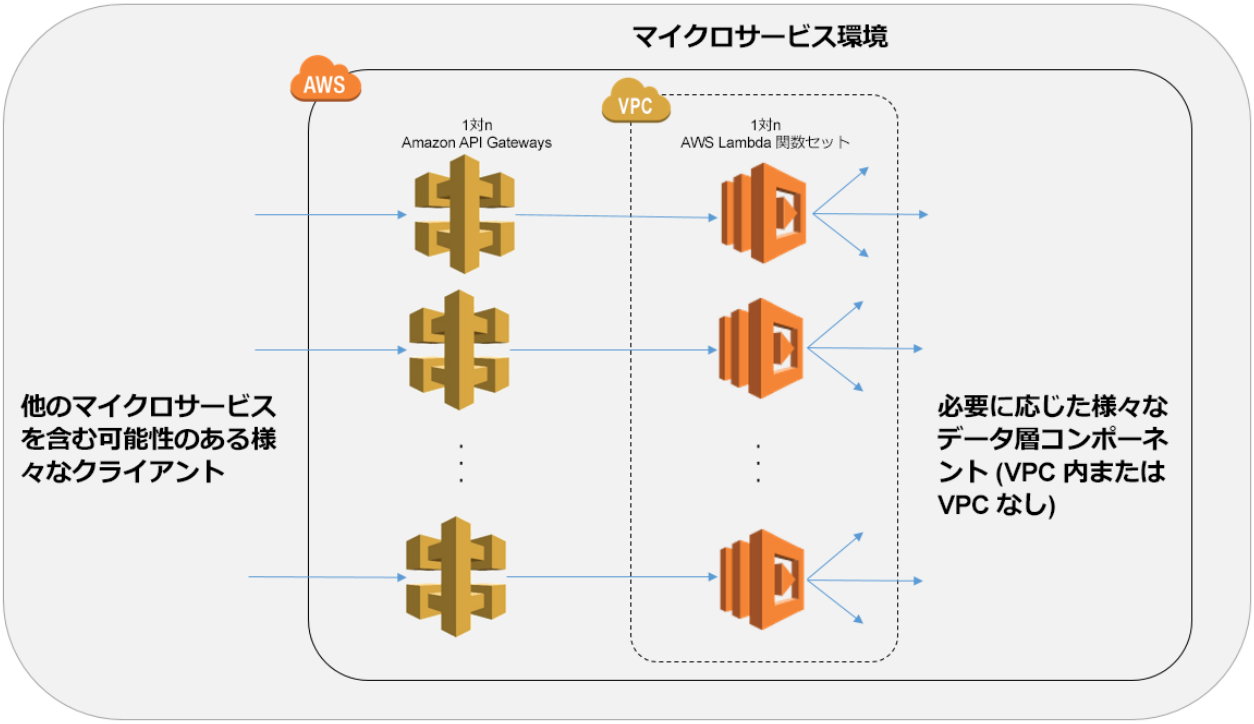

図 5: マイクロサービス環境用のアーキテクチャパターン

マイクロサービスアーキテクチャパターンは、このホワイトペーパーで説明した典型的な 3 層アーキテクチャには該当しません。マイクロサービスアーキテクチャでは、ソフトウェアコンポーネントの分離が大量に行われるため、全体的に多層アーキテクチャの利点が増幅されます。マイクロサービスを構築するために必要なのは、Amazon API Gateway で作成した API と、AWS Lambda で実行される後続の関数のみです。お客様のチームはこれらのサービスを自由に使用して、必要な粒度レベルまで環境を分離し、断片化することができます。

一般的に、マイクロサービス環境を使用すると、各マイクロサービスを新しく作成するときに毎回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、複数バージョンの複数マイクロサービスを同時に実行する複雑さ、多数の別々のサービスと統合するためのクライアント側コード要件の増加などの問題につながることがあります。

ただし、AWS サーバーレスパターンを使用してマイクロサービスを作成する場合には、これらの問題の解決がより簡単になり、場合によっては問題が完全に消滅することもあります。AWS マ

イクロサービスパターンでは、構造の各マイクロサービスを作成するための障害が小さくなります (Amazon API Gateway では既存 API の複製も可能です)。このパターンでは、サーバーの使用率を考慮する必要がなくなります。API Gateway と Lambda の両方によって、シンプルなバージョニングが可能になります。Amazon API Gateway では、統合のオーバーヘッドを縮小するために、プログラムで生成されたクライアント SDK が多数の一般的な言語で提供されています。

まとめ

多層アーキテクチャパターンでは、保守や分離が容易でスケーラブルなアプリケーションコンポーネントを作成するというベストプラクティスが推奨されています。ロジック層を作成し、Amazon API Gateway を介して統合して AWS Lambda 内でコンピューティングを実行している場合は、目標を達成するための労力を削減しながら、その実現に近づいていきます。これらのサービスを連携させることで、クライアント用の HTTPS API フロントエンドと、ビジネスロジックを実行するための VPC 内のセキュアな環境を実現できます。これにより、典型的なサーバーベースのインフラストラクチャを自分で管理する代わりに、多くの一般的なシナリオを使用して、これらのマネージド型サービスを利用することができます。

寄稿者

本書の執筆に当たり、次の人物および組織が寄稿しました。

Andrew Baird (AWS ソリューションアーキテクト)

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 (AWS Mobile、シニアプロダクトマネージャー)

注記

[1] http://aws.amazon.com/api-gateway/

[2] http://aws.amazon.com/lambda/

[3] https://aws.amazon.com/vpc/

[4] https://aws.amazon.com/cloudfront/

[5] https://d0.awsstatic.com/whitepapers/DDoS_White_Paper_June2015.pdf

[6] https://aws.amazon.com/api-gateway/pricing/

[7] http://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-mock-integration.html

[8] http://aws.amazon.com/iam/

[9] http://docs.aws.amazon.com/general/latest/gr/signature-version-4.html

[10] http://docs.aws.amazon.com/lambda/latest/dg/intro-core-components.html#intro-core-components-event-sources

[11] https://aws.amazon.com/s3/

[12] https://aws.amazon.com/kinesis/

[13] https://aws.amazon.com/vpc/

[14] https://aws.amazon.com/rds/

[15] https://aws.amazon.com/elasticache/

[16] https://aws.amazon.com/redshift/

[17] https://aws.amazon.com/ec2/

[18] https://aws.amazon.com/dynamodb/

[19] https://aws.amazon.com/s3/storage-classes/

[20] https://aws.amazon.com/elasticsearch-service/

[21] https://aws.amazon.com/cognito/