AWS サーバーレス多層アーキテクチャ

Amazon API Gateway と AWS Lambda の活用

2015 年 11 月

注意

本書は情報提供のみを目的としています。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 製品またはサービスの利用について、独自の評価に基づき判断する責任を負います。いずれの AWS 製品またはサービスも、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」提供されます。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。

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) の革新的な機能を使用して、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一般的なパターンの多層アーキテクチャを設計する方法をどのように変えることができるか説明します。Amazon API Gateway と AWS Lambda を使った実装パターンを使用することにより、アーキテクトと開発者は、多層アプリケーションの作成と運用管理に必要な開発サイクルと運用サイクルを低減できるようになりました。

はじめに

多層アプリケーション (3 層、n 層など) は、この数十年にわたり、根幹となるアーキテクチャパターンとして使用されています。多層パターンは、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構築し、そのコンポーネントを (多くの場合、別々のチームによって) 個別に管理および維持するためのガイドラインを提供します。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にサービス指向アーキテクチャ (SOA) 手法を使用して構築されることがありました。この手法では、ネットワークは階層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新しいウェブサービス層を作成するという多くの単調な作業を行う必要があります。多層ウェブアプリケーション内に記述されたコードの大部分は、多層パターンの直接的な実装です。サンプルには、ある階層と別の階層を統合するコード、お互いを理解するために階層で使用する API とデータモデルを定義するコード、階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のコードなどがあります。

API を作成、管理できる Amazon API Gateway[1] と任意のコード関数を実行できる AWS Lambda[2] 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 HTTPS リクエスト経由でユーザー定義のコード関数を直接トリガーできます。必要なリクエストの量に関係なく、API Gateway と Lambda は両方とも自動的にスケールして、アプリケーションのニーズに正確に対応します。この 2 つのサービスを組み合わせてアプリケーションの階層を作成すると、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に集中できます。また、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、クライアント認可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その他の単調な作業を行う必要が**なくなります**。

AWS には、Amazon Virtual Private Cloud (Amazon VPC)[3] 内で実行する Lambda 関数を作成する機能があります。この機能により、API Gateway と Lambda を組み合わせることで得られる利点はさらに拡張され、ネットワークの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ます。例えば、機密情報が保存されているリレーショナルデータベースとウェブサービスを統合する必要があるとします。Lambda と Amazon VPC の統合により、

Amazon API Gateway の機能を間接的に拡張し、開発者は、バックエンドを Amazon VPC でプライベートかつセキュアに保護したまま、その前面にインターネットからアクセス可能なオリジナルの HTTPS API を配置できます。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で適用できます。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、ユーザー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は、**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。プレゼンテーション層は、ユーザーが直接操作するコンポーネント (ウェブページ、モバイルアプリケーションの UI など) です。ロジック層は、プレゼンテーション層のユーザーアクションを、アプリケーションの動作を制御する機能に変換するために必要なコードを含んでいます。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) で構成されます。図 1 に、簡単な 3 層アプリケーションの例を示します。

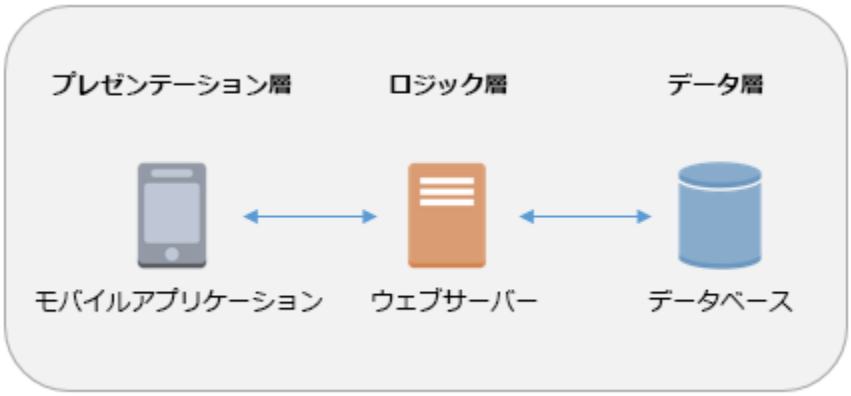

図 1: 簡単な 3 層アプリケーションのアーキテクチャパターン

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。このホワイトペーパーでは、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。Amazon API Gateway と AWS Lambda を統合してロジック層を構築すると、この頭脳に大きな変革をもたらすことができます。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本番環境向けサーバーレスアプリケーションを構築できます。アプリケーションで数千台のサーバーを使用するとしても、このパターンでは、サーバーを 1 台も管理する必要がありません。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、以下のような利点も得られます。

- オペレーティングシステムの選定、保護、パッチ適用、管理を行う必要がない
- サーバーのサイジング、モニタリング、スケールアウトを行う必要がない
- 過剰プロビジョニングによる、コスト発生のリスクがない
- 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がない

また、各サービスの特定の機能は、多層アーキテクチャパターンで効果を発揮します。

Amazon API Gateway

Amazon API Gateway は、API の定義、デプロイ、保守を行うためのフルマネージドサービスです。クライアントは、標準の HTTPS リクエストを使用して API と統合します。このような特徴は明らかに、サービス指向の多層アーキテクチャに適しています。さらに、このサービスに固有の機能と品質は、ロジック層の強力な利点となり得ます。

AWS Lambda との統合

Amazon API Gateway により、アプリケーションは、AWS Lambda の革新的な機能を簡単な方法 (HTTPS リクエスト) で直接利用できます。API Gateway は、プレゼンテーション層と AWS Lambda で記述した関数との間をつなぐ橋の役割を担います。API を使用してクライアントとサーバーの関係を定義すると、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。渡される内容は、リクエストメタデータ、リクエストヘッダー、リクエスト本文です。

世界中で安定した API パフォーマンス

Amazon API Gateway でデプロイする各 API は、Amazon CloudFront[4] 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 と統合するクライアントの接続ポイントとして、Amazon 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 の応答時間のレイテンシーを短縮できます。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、DDoS 攻撃の対策についての AWS のベストプラクティスホワイトペーパー[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。これにより、繰り返し実行される API リクエストのパフォーマンスが向上するだけでなく、バックエンドの実行時間を短縮して、全体的なコストの削減を実現することができます。

イノベーションを促進

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といえます。プロジェクトを開始するには、その正当性を証明する必要があります。開発作業に費やされる投資の金額を減らし、時間を節約することで、より多くの実験を行ったり、より自由にイノベーションを実現したりできます。

ウェブサービスベースの多層アプリケーションでは多くの場合、プレゼンテーション層はユーザーごとの違い(モバイルデバイス、ウェブブラウザなどの違い)によって簡単に断片化されてしまいます。さらに、ユーザーはほとんどの場合、地理的にも分散しています。しかし、疎結合な形で作られたロジック層は、ユーザーごとに物理的に断片化されるものではありません。多くの場合、すべてのユーザーは、同じインフラストラクチャを利用してロジック層を実行します。このことは、インフラストラクチャの重要性を強調することになります。「サービス開始初期にメトリクスを測定する仕組みを実装する必要はない」とか、「初期の使用量は少ないため、スケールする方法は後で考えればよい」といった、ロジック層の実装の初期段階にありがちな安易な考えが、新しいアプリケーションを迅速に提供するための近道として提案されることがあります。このことは、既に本番環境で実行中のアプリケーションに変更をデプロイする必要が生じるときに、技術的負債と運用上のリスクを発生させる可能性をはらんでいます。

Amazon API Gateway にはそれらが既に実装されているため、本当の近道を使ってアプリケーションを迅速に提供できます。

アプリケーションのライフタイムがどの程度なのかわからなかったり、または短期間であることがわかっているというようなことはあります。このような状況で、新しい多層アプリケーションを使用するビジネスケースをきちんと定義することは難しい場合があります。Amazon API Gateway を使用すれば、マネージド型の機能がスタートする段階で組み込まれており、インフラストラクチャのコストが API でリクエストを受信し始めたときに初めて発生するため、ビジネスケースの定義は簡単になります。詳細については、Amazon API Gateway の料金[6] を参照してください。

迅速に反復し、俊敏性を維持

新しいアプリケーションの場合、ユーザーベースの定義がまだ十分でない可能性があります (サイズ、使用パターンなど)。ユーザーベースが明確になるまで、ロジック層の俊敏性を維持する必要があります。アプリケーションと企業は、早期導入ユーザーの要望の変化に順応できる必要があります。Amazon API Gateway では、API の登録からデプロイまでに必要な開発サイクルの工数を減らすことができます。Amazon API Gateway ではモック統合[7] を作成できるため、API Gateway から直接 API レスポンスを生成できます。これにより、API レスポンスを使用するクライアントアプリケーションを開発しながら、完全なバックエンドロジックの開発作業を並行して進めることができます。この利点は、API を最初にデプロイするときだけでなく、企業がユーザーの要望に応えてアプリケーション (と既存の API) を迅速に転換する必要があると判断し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機能を使用できるため、既存の機能とクライアントの依存関係をそのまま維持した状態で、新しい機能を API または関数の別のバージョンとしてリリースできます。

セキュリティ

3 層の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について考慮する必要が生じます。アプリケーションでは、認可されたクライアントのみが (ネットワーク上に公開される) 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、バックエンドを確実に保護することでこのセキュリティ上の問題に対応します。アクセスコントロールについては、クライアントアプリケーションに静的な

API キー文字列を提供する方法だけに頼るということはしないでください。この API キー文字列はクライアントから抽出され、他の場所で流用される可能性があるためです。
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。

- 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。
- AWS Lambda 関数へのアクセスを制限して、Amazon API Gateway 内の特定の API と AWS Lambda の特定の関数の間にのみ信頼関係を設定できます。Lambda 関数は、その関数に関連付けた API を使用する場合にのみ、呼び出すことができます。
- Amazon API Gateway では、API と統合するためのクライアント SDK を生成できます。SDK では、API で認証が必要な場合にリクエストの署名を管理することもできます。認証のためにクライアント側で使用されるこれらの API 認証情報は、AWS Lambda 関数に直接渡されます。必要に応じて、独自に記述したコードによって関数内で追加の認証を行う場合もあります。
- 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、固有の 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)[8] ポリシーにリストアップされます。
  - つまり、ユーザーの API は、AWS が管理する他の API と同様に第一級オブジェクトとして処理されます。IAM ポリシーはきめ細かく設定できます。IAM ポリシーによって、Amazon API Gateway を使用して作成した API に関連付けられた特定のリソースまたはメソッドを参照できます。
  - API アクセスは、アプリケーションコードとは関係なく作成された IAM ポリシーによって制御されます。つまり、このアクセスレベルを認識または適用するためにコードを記述する必要がないということです。コードがなければ、コードでバグが発生することも、コードの脆弱性が悪用されることもありません。
  - AWS 署名バージョン 4 (SigV4)[9] 認可と API アクセス用の IAM ポリシーを使用してクライアントを認可することで、必要に応じて同じ認証情報を使用して、AWS の他のサービスやリソース (例えば、

Amazon S3 バケットや Amazon DynamoDB テーブル) へのアクセスを制限または許可できます。

AWS Lambda

基本的に、AWS Lambda では、サポートされている言語のいずれかで記述された任意のコードをイベントに対する応答としてトリガーできます。このイベントは、AWS が提供するプログラムによるトリガーのうちの 1 つで、**イベントソース**と呼ばれます (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 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 Simple Storage Service (Amazon S3)[11] に保存されているファイルの処理、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。

AWS Lambda 関数は、Amazon API Gateway と組み合わせて代表的なウェブサービスに組み込み、HTTPS リクエストによって直接トリガーできます。Amazon API Gateway がロジック層の窓口として機能しますが、ロジックはその API の背後で実行させる必要があります。そこで AWS Lambda が活躍します。

ビジネスロジックをここに配置

AWS Lambda では、**ハンドラー**と呼ばれるコード関数を記述できます。このハンドラーは、イベントによってトリガーされたときに実行されます。例えば、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できます。Lambda では、ユーザーが選択した詳細レベルでモジュール式のハンドラーを作成できます (API ごとに 1 つ、または API メソッドごとに 1 つ)。このハンドラーは、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、依存関係のある任意のその他のモジュール (コードと一緒にアップロードしたその他の関数、ライブラリ、ネイティブバイナリ、外部のウェブサービスなど) と自由にやり取りできます。Lambda では、作成時に必要なすべての依存関係を関数定義にパッケージ化できます。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 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 Lambda 関数の定義で同じデプロイパッケージを自由に再利用できます。サーバーレスの多層アーキテクチャパターンでは、Amazon API Gateway で作成した各 API が、必要なビジネスロジックを実行する Lambda 関数 (および内部のハンドラー) と統合されます。

Amazon VPC との統合

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。データ層にはビジネスやユーザーに関する機密情報を保存することが多いため、データ層はセキュリティで厳重に保護する必要があります。Lambda 関数から統合できる AWS のサービスは、IAM ポリシーを使用してアクセスコントロールを管理できます。これらのサービスには、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、独自でアクセスコントロールを管理するコンポーネントが存在する可能性があります。そのようなコンポーネントは、プライベートネットワーク環境、つまり Amazon Virtual Private Cloud (Amazon VPC)[13] にデプロイすることで、セキュリティを強化できます。

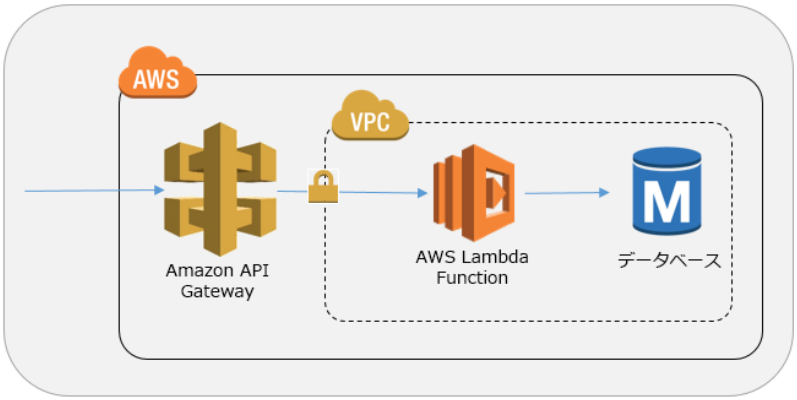

図 2: VPC を使用するアーキテクチャパターン

VPC を使用すると、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 Lambda コード関数からのリクエストに**限定する**こともできます。

セキュリティ

Lambda 関数を実行するには、IAM ポリシーによって許可されている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ます。ユーザーが定義した API Gateway リクエストによって呼び出さない限り**決して**実行できない

Lambda 関数を作成することもできます。コードは、作成した API によって定義されているユースケースに当てはまる場合にのみ処理されます。

各 Lambda 関数は IAM ロールを継承します。ロールの権限は、IAM の信頼関係に基づいて付与されることになっています。この IAM ロールによって、Lambda 関数によってやり取りできる AWS のその他のサービスやリソース (Amazon DynamoDB テーブル、Amazon S3 バケットなど) を定義します。関数からアクセスできるサービスは、関数の外側で定義および制御されます。これは、些細なことですが、高い効果があります。AWS 認証情報を保存したり、取得したりするロジックをコードに記述する必要がないということを意味しているのです。つまり、API キーをハードコーディングする必要はなく、API キーを取得し、メモリに保存するためのコードを記述する必要もありません。IAM ロールの定義に従って許可されているサービスを呼び出すために Lambda 関数を有効にする工程は、サービスによって自動的に行われます。

データ層

AWS Lambda をロジック層として使用すると、データ層のデータストレージをさまざまな選択肢の中から選択できます。これらの選択肢は、Amazon VPC でホストされるデータストアと IAM の下で認可されるデータストアという 2 つのカテゴリに分類できます。AWS Lambda では、そのどちらともセキュアに統合できます。

Amazon VPC でホストされるデータストア

AWS Lambda と Amazon VPC の統合により、関数とさまざまなデータストレージ技術をプライベートかつセキュアな方法で統合できます。

- Amazon RDS[14]

  Amazon Relational Database Service (Amazon RDS) で提供されているエンジンのいずれかを使用します。Lambda で記述したコードから直接 Amazon RDS に接続します。Lambda の外部から接続する場合と同様ですが、Lambda を使用することには、AWS Key Management Service (AWS KMS) と簡単に統合して、データベースの認証情報を暗号化できるという利点があります。

- Amazon ElastiCache[15]

Lambda 関数をマネージド型のインメモリキャッシュと統合して、アプリケーションのパフォーマンスを向上させます。

- Amazon RedShift[16]

  レポート、ダッシュボードを作成するために、またはアドホッククエリの結果を取得するためにエンタープライズデータウェアハウスに対して安全にクエリを実行する関数を作成できます。

- Amazon Elastic Compute Cloud (Amazon EC2)[17] によってホストされる内部ウェブサービス

  既に VPC 内で内部ウェブサービスとしてアプリケーションを実行している場合は、論理的にプライベートな VPC ネットワークのみを経由する HTTP リクエストを Lambda 関数から送信します。

IAM の下で認可されるデータストア

AWS Lambda は IAM と統合されているため、AWS API を使用して直接利用できる AWS のサービスとの統合を保護するために IAM を使用できます。

- 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。Amazon DynamoDB を使用すると、規模に関係なく、10 ミリ秒未満のパフォーマンスでデータレコード (現時点で、400 KB 以下) を取得できます。Amazon DynamoDB のきめ細かいアクセスコントロールを使用すると、Lambda 関数で DynamoDB の特定のデータについてクエリを実行するときに最小権限のベストプラクティスに従うことができます。

- Amazon S3[19]

  Amazon Simple Storage Service (Amazon S3) では、インターネット規模のオブジェクトストレージを提供します。Amazon S3 は、オブジェクトの耐久性が 99.999999999% になるように設計されているため、低コストで高い耐久性を備えたストレージを必要とするアプリケーションに適しています。また、Amazon S3 は 1 年を通してオブジェクトの可用性が最大 99.99% になるように設計されているため、高可用性ストレージを必要とするアプリケーションに適しています。Amazon S3 に保存され

たオブジェクト (ファイル、イメージ、ログ、任意のバイナリデータ) 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 Amazon S3 と安全にやり取りでき、S3 内のデータへのアクセスは、Lambda 関数に関連付けられた IAM ポリシーのみに制限できます。

- Amazon Elasticsearch Service[20]

  Amazon Elasticsearch Service (Amazon ES) は、人気のある検索分析エンジン Elasticsearch のマネージドサービスです。Amazon ES では、クラスターのプロビジョニング、障害検出、ノードの交換がマネージドサービスとして提供されます。Amazon ES API へのアクセスは IAM ポリシーを使用して制限できます。

プレゼンテーション層

Amazon API Gateway によって、プレゼンテーション層でのさまざまな可能性が広がります。インターネット経由でアクセス可能な HTTPS API は、HTTPS 通信を行うことができる任意のクライアントが利用できます。アプリケーションのプレゼンテーション層に使用される可能性がある一般的な例を以下に示します。

- モバイルアプリケーション: Amazon API Gateway と AWS Lambda を使用してカスタムビジネスロジックと統合できることに加えて、ユーザー ID を作成および管理するメカニズムとして Amazon Cognito[21] を使用できる可能性があります。
- ウェブサイトの静的コンテンツ (Amazon S3 でホストされたファイルなど): Amazon API Gateway の API で Cross-Origin Resource Sharing (CORS) を有効にすることができます。これにより、ウェブブラウザが、静的ウェブページ内から直接 API を呼び出すことができるようになります。
- 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 を経由して通信できます。Amazon API Gateway を使用して作成した API とクライアントは標準の HTTPS を使用して通信します。特別なものでも独自仕様でもありません。特定の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。

サンプルアーキテクチャパターン

ロジック層を構築するための接着剤として Amazon API Gateway と AWS Lambda を使用して、以下のような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、AWS のサービスのみを使用しているため、ユーザーが自分でインフラストラクチャを管理する必要はありません。

モバイルバックエンド

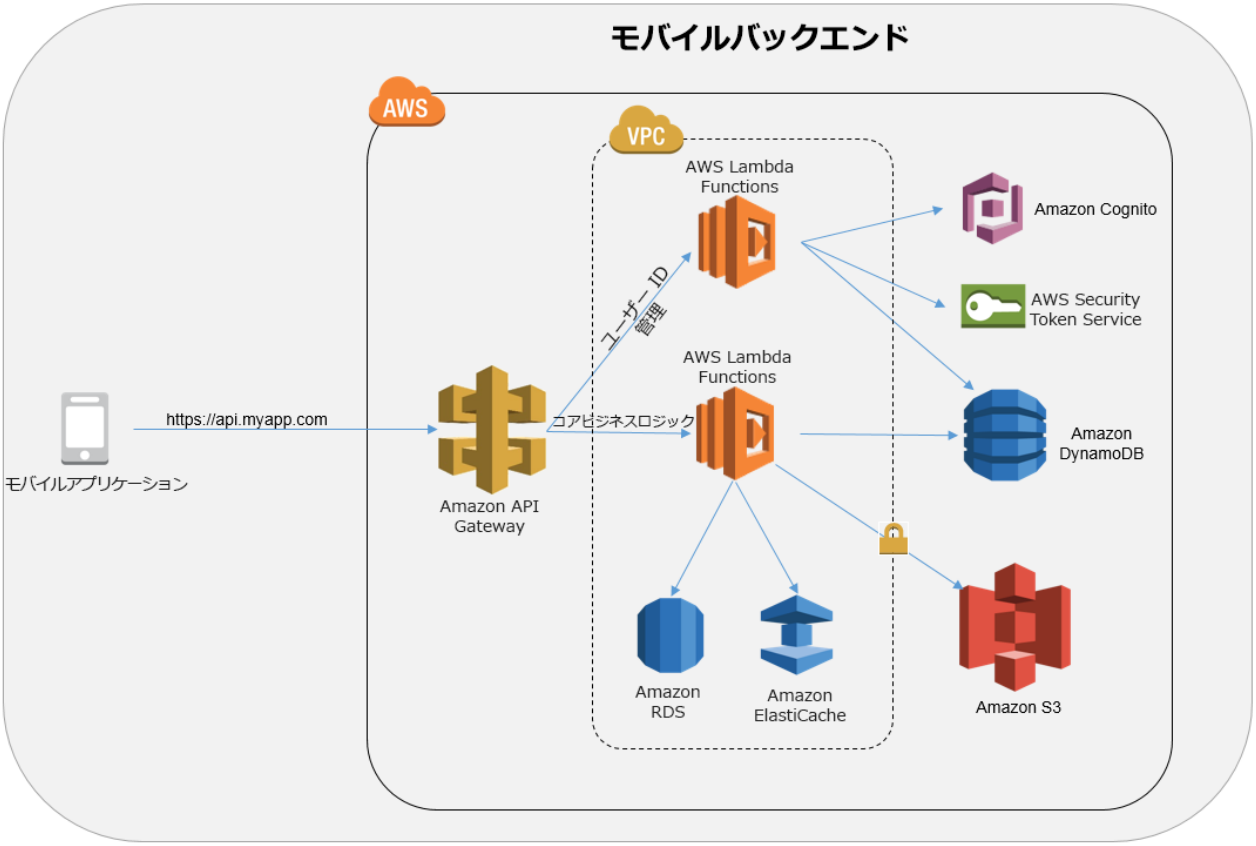

図 3: モバイルバックエンドのアーキテクチャパターン

- **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。このロジック層は、Amazon API Gateway の一部として作成される Amazon CloudFront ディストリビューションによって世界中に分散されます。Lambda 関数をユーザー/デバイスの ID 管理と認証専用として設定できます。同時に、これを Amazon Cognito で管理させることもできます。こうすることで、IAM と統合して一時ユーザーがアクセスするための認証情報を提供したり、一般的なサードパーティーの ID プロバイダーと統合したりできます。また、別で定義した Lambda 関数では、モバイルバックエンドに合わせて中核となる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利用できます。選択肢については、前述のセクションを参照してください。

Amazon S3 でホストするウェブサイト

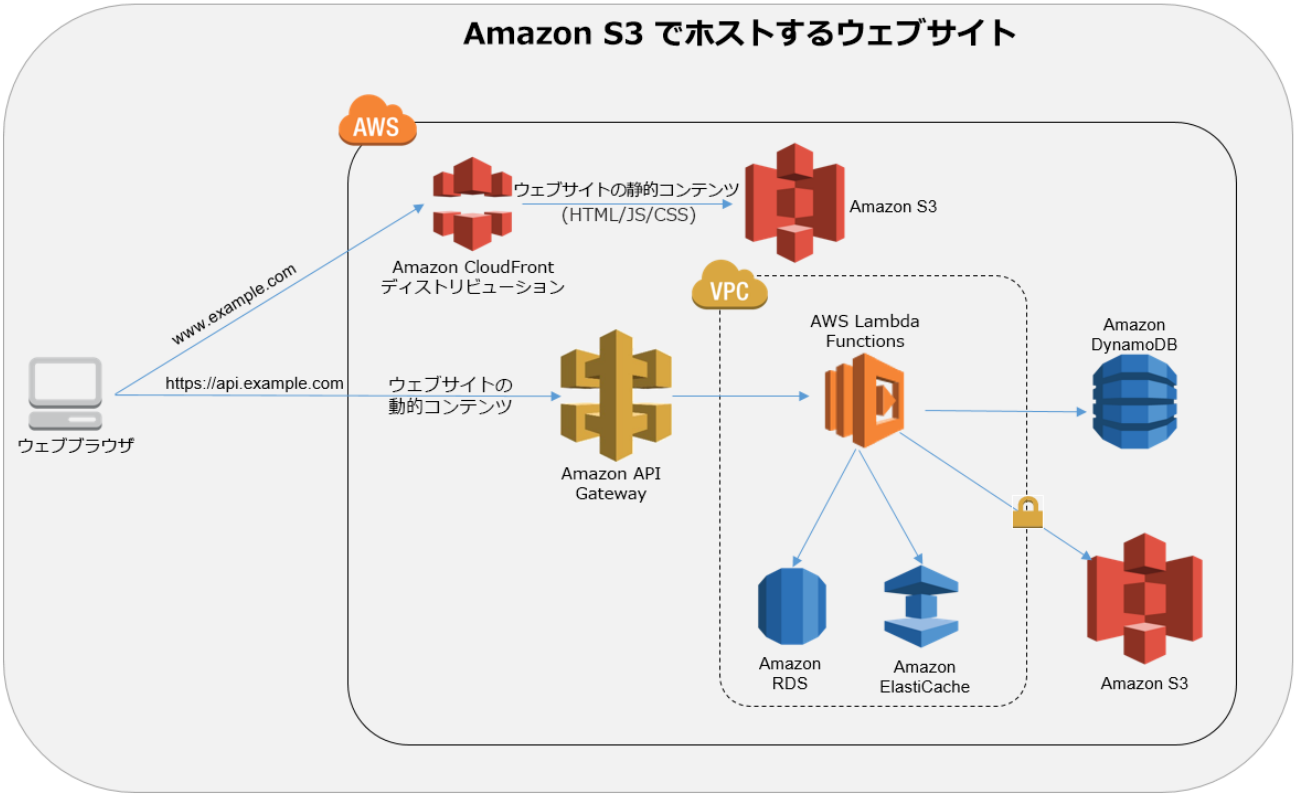

図 4: Amazon S3 でホストする静的ウェブサイトのアーキテクチャパターン

- **プレゼンテーション層:** Amazon S3 でホストし、Amazon CloudFront で配信する静的ウェブサイトのコンテンツ。Amazon S3 で静的ウェブサ

イトのコンテンツをホストすると、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よりもコスト効率が向上します。ただし、リッチな機能を提供するウェブサイトでは、多くの場合、静的コンテンツを動的なバックエンドと統合する必要があります。

- **ロジック層:** Amazon API Gateway と AWS Lambda。Amazon S3 でホストする静的ウェブコンテンツは、CORS を有効にして 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利用できます。選択肢については、前述のセクションを参照してください。

マイクロサービス環境

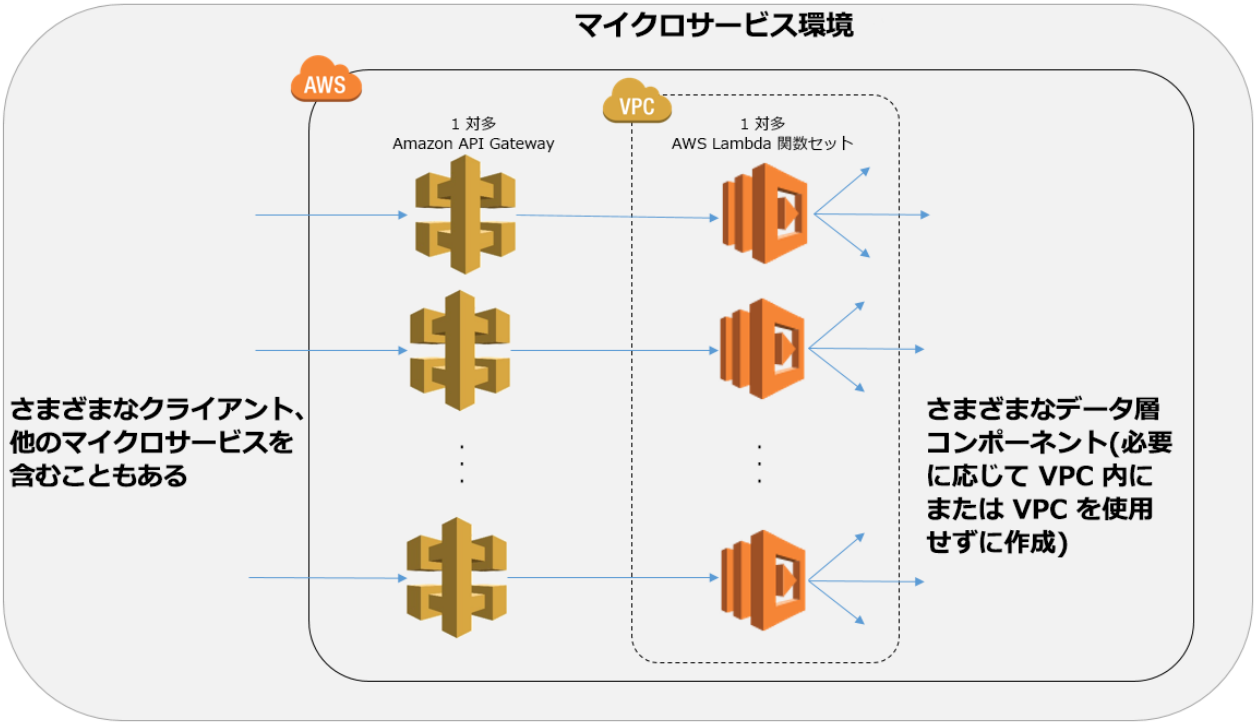

図 5: マイクロサービス環境のアーキテクチャパターン

マイクロサービスのアーキテクチャパターンは、このホワイトペーパーで説明している典型的な 3 層アーキテクチャとは異なるものです。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が徹底的に疎結合化されているため、多層アーキテクチャの利点が強化されます。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 によって実行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/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。

このような問題は、AWS の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。AWS の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 (Amazon API Gateway では既存の API のクローンを作成することもできます)。この実装パターンでは、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。API Gateway と Lambda にはシンプルなバージョニング機能が用意されています。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、管理しやすい疎結合型の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 でコンピューティングを行う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。これらのサービスは、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実行するための安全な環境を VPC 内に構築します。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、これらのマネージドサービスを使用する、人気のある多くのシナリオを利用できます。

寄稿者

本ドキュメントの執筆にあたり、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 (シニアプロダクトマネージャー、テクノロジー AWS Mobile)

Vyom Nagrani (シニアプロダクトマネージャー、AWS Mobile)

Ajay Nair)シニアプロダクトマネージャー、AWS Mobile)

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/