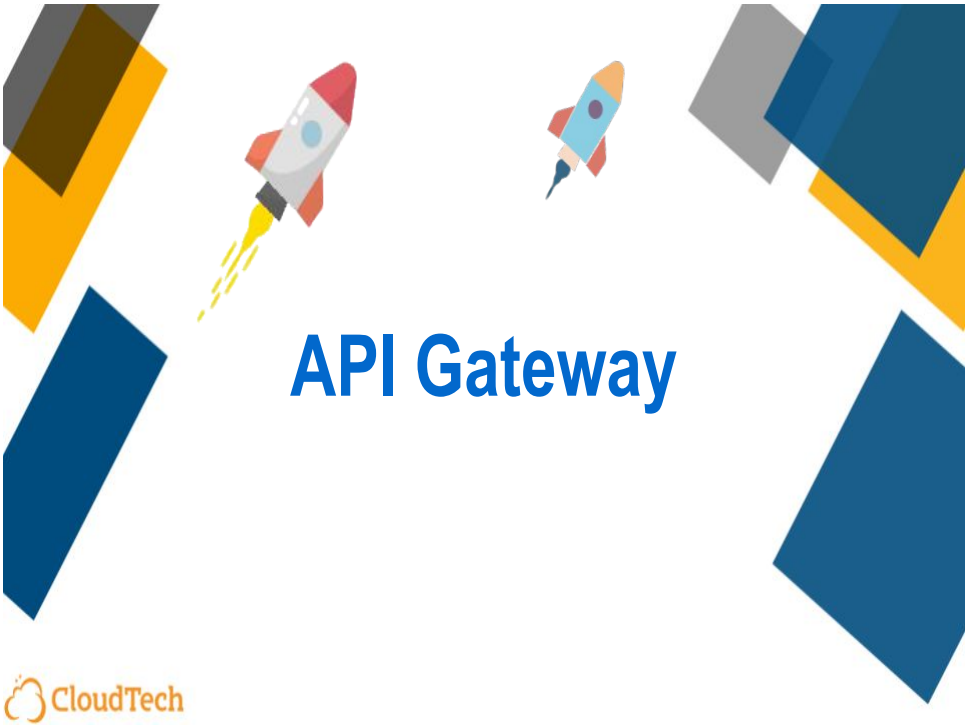
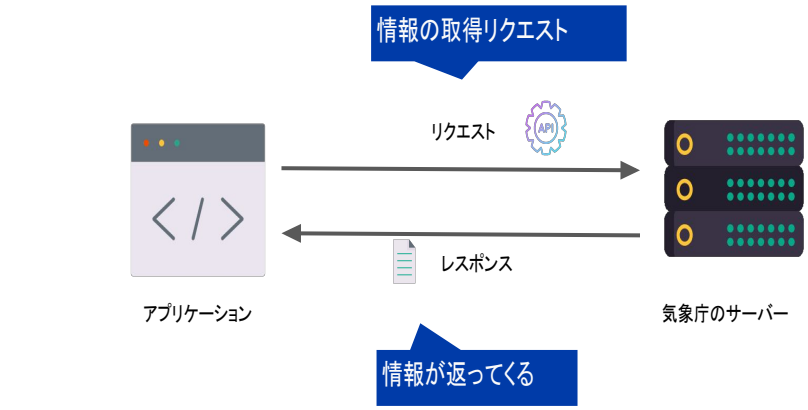


- API Gatewayは、アプリケーションプログラミングインターフェースであるAPIを作成できるサービスです。
- AWSサービスで作成したアプリの入口として機能します。

そもそもAPIとは

API = Application Programming Interface

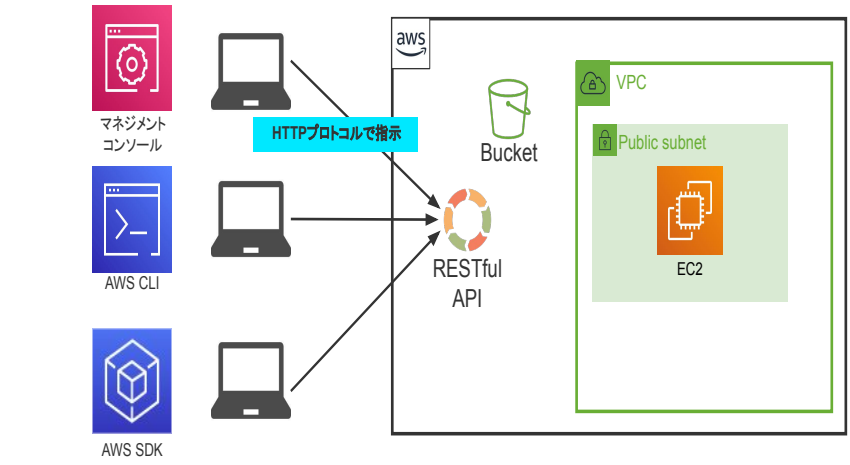


CloudTech

- 前提として、APIとはそもそも何でしょうか？
  - アプリケーションプログラミングインターフェース、の略です。
    - この図のように、アプリケーションがどこかに情報の取得をリクエストしたら、何かしらの結果を返してくる。この仕組みそのものや、一連の通信がAPIと呼ばれるものです。
  - 例えば、あなたがお天気アプリを作っているとしたら、
    - 気象庁の気象データが入っているサーバーに対してしかるべきAPIの形式でリクエストを発信する（これをAPIを投げる、とも表現します）が、目的の天気情報が得られるという形です。
- 大切なのは、あなたのアプリは、しかるべき APIを投げるだけでよくて、別に気象庁サーバーの仕組みを気にしなくても良いということです。

- 私たちは、とにかく目的の気象情報を手っ取り早く取得したい。
  - 気象庁に電話で聞いて、それをアプリに入力することもできますが、現実的ではありません。
  - APIは情報を効率的に、素早く取得するための仕組みです。
    - たいていは、HTTPを利用して入力と出力をやり取りすることになります。

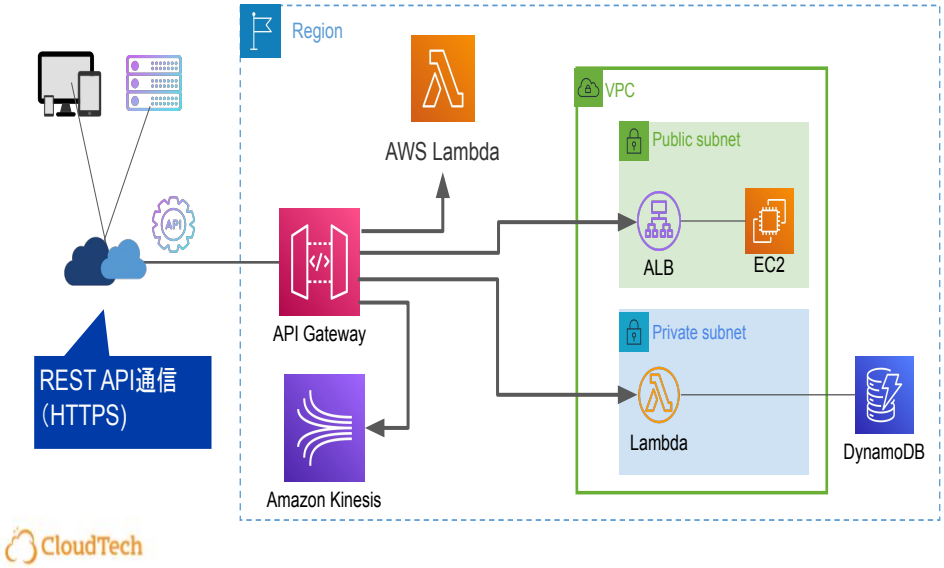
そもそもAPIとは



CloudTech

- AWSでも、操作方法は違えど、APIを使用して操作をしています。
- 画面でEC2インスタンスを起動するボタンを押したら、EC2が起動し始めますよね？
    - あの行為も、ボタンを押すことで APIがAWSに投げられ、AWSの内部で色々と動いているのですが、私たちは気にしません。
    - EC2を起動してくださいというリクエストを投げて、結果OKでしたよというレスポンスを AWSから受け取っています。
    - APIという概念がざっくり掴めたら、次は API Gatewayの設計例をみていきましょう。

API Gatewayの概要



- PCやスマホ、もしくは外部サービスなど、様々なクライアントが
  - インターネットを通して、AWSで構築したサービスに「REST API通信で」アクセスする必要がある場合、
  - AWSのリージョン内に作成されたAPI Gatewayが出入り口となって機能します。

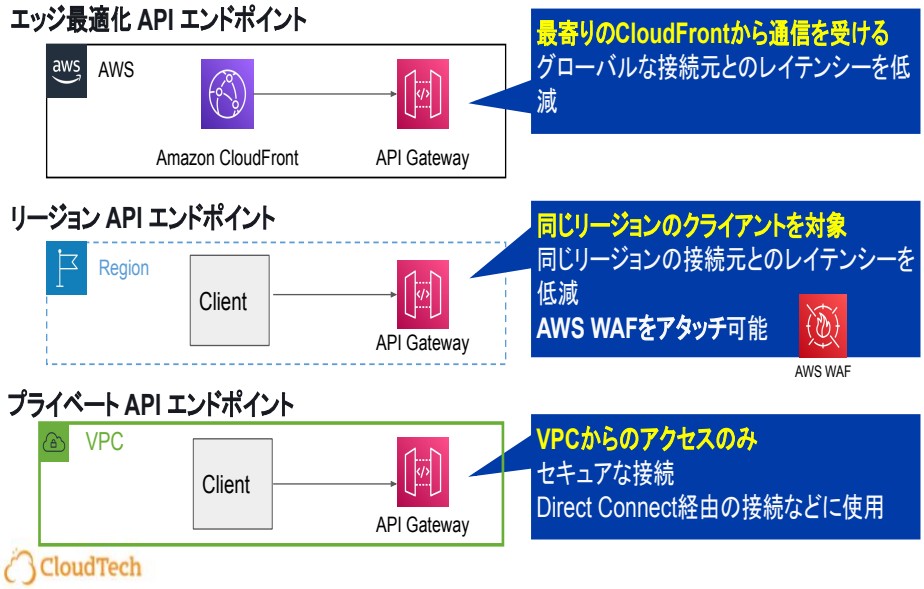
    - API Gatewayで受け付けたリクエストは、実際のビジネスロジックとして動いている場所に送られます。
      - 例えば、Lambda関数だったり、
      - ALBに紐づけられたEC2インスタンスだったり、
      - または、VPC内にデプロイしたLambdaを呼び出し、DynamoDBに値を書き込んだり、
      - Kinesisに連携したり、

    - または、AWS以外の外部サービスである可能性もあります。

- 基本的に、API GatewayはAWSサービスに接続するための入口だと考えてください。

API Gatewayには様々なデプロイメントタイプがあります。

API Gatewayデプロイタイプ



要件に応じた3つの種別のエンドポイントが提供されています。

- ますは、エッジ最適化 APIエンドポイント
- CloudFront Distributionの背後にAPI Gatewayを配置するものです。
- CloudFrontのエッジロケーションを経由して APIのリクエストが送られてくるため、世界中からの通信を想定したような、リージョンを跨いだリクエストであっても、リクエストのレイテンシー（応答時間）を減らすことができます。

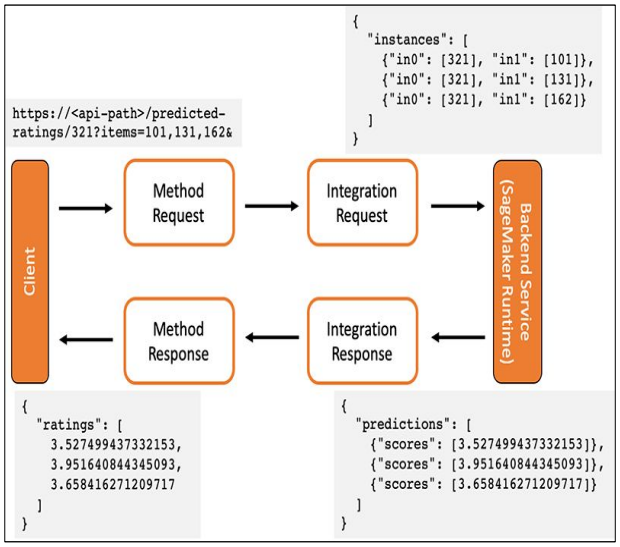
- 次に、リージョン APIエンドポイントです。
- 先ほどはCloudFrontから通信をうけていましたが、これはクライアントから直接 REST APIにリクエストが届くものです。
- リクエストの発信元が同一リージョンであれば、先ほどのものよりレイテンシが低くなります。
- また、こちらの設定では、API GatewayにAWS WAFをア

- タッチして、セキュリティ対策をすることも可能です。

- そして最後に、プライベート APIエンドポイントがあります。
- これはVPC内にAPI Gatewayを配置するため、先ほどの 2つのタイプとは異なり、パブリックなエンドポイントをもちません。最も安全に公開する方法と言ってもいいでしょう。
- VPC 内からのアクセス、または Direct Connect接続、もしくは Private Linkを経由したアクセスのみ可能になります .
    - どれを利用するかは、クライアントの性質によって異なってきますが、基本的には EdgeかRegionを選択することになるかと思います。

- 次は、リクエストとレスポンスの構造を見てみましょう。

API Gatewayリクエスト/レスポンス構造

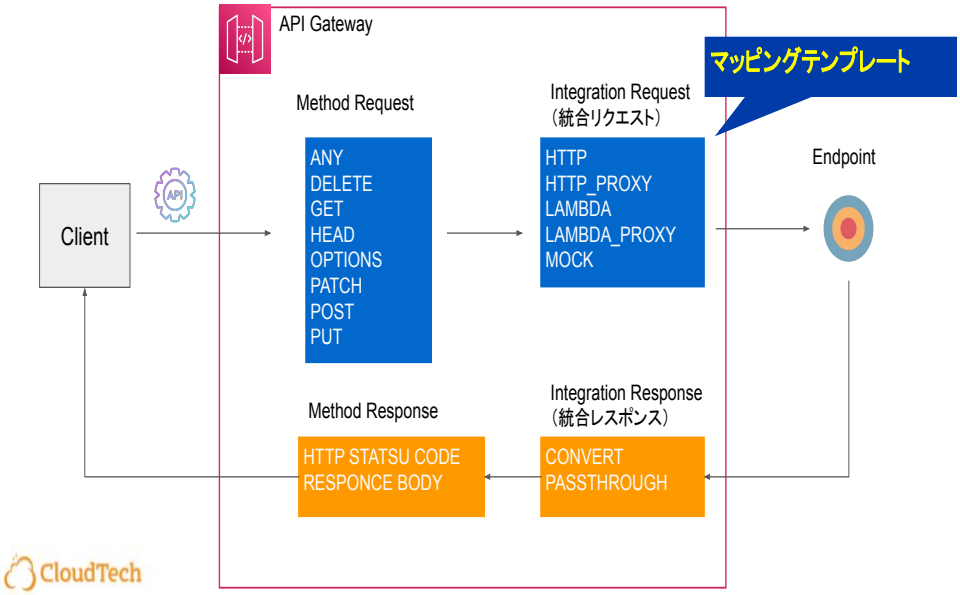


CloudTech

公式ブログからの図です。

- 左側がクライアントです。右側がバックエンド（例えばLambda関数）などです。
- クライアントから受け取ったリクエストをバックエンドに渡して何かしらの処理をし、最終的にバックエンドからクライアントにレスポンスが返ってくるまでを図にしたものです。
  - API Gatewayでは、このように4つの設定があります。
    - リクエストとレスポンスに対して、それぞれメソッドとインテグレーションと呼ばれるものを設定することができます。詳しく見ていきましょう。

API Gatewayリクエスト/レスポンス構造



- Method Request
  - クライアントのリクエストが API Gatewayに到達した時の受付のような役割です。
    - 受信したリクエストに対して、ヘッダー、クエリパラメーター、認証、バリデーター、後述するAPI keyを持っているかのチェック、などの設定が可能です。
    - メソッドの種類は、HTTPメソッドと同様に、DELETE、GET、POST、PUTなどがあります。
      - 間違った形式のリクエストを後続に流してしまうと、処理を無駄に走らせることになります。
      - そうならないためにも、Method Requestで仕様や文法などが合っている適切に記述か？（バリデーションと言います。）これが必要になります .

        - そして次にリクエストが渡される先は Integration Requestになります。

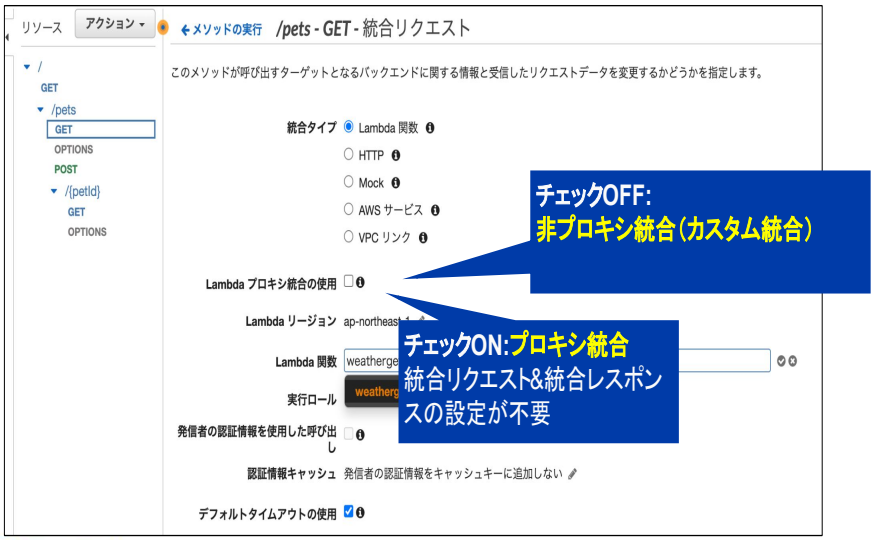
- Integration Reques
    - このIntegration Requestで、バックエンドとの接続情報を設定します。
        - 例えば、Lambda関数と接続する場合はどのLambdaかを指定します。
    - 大切なのが、API Gatewayは実際のアプリケーションの処理を行うものではありません。 実際の処理はバックエンドが行います。
    - なので、役割としては、実際にリクエストを処理するエンドポイント（例えばLambda関数）に対して、どのようにデータを渡すか（必要に応じてデータの形を整える）という役割があります。
        - 例えばXML形式で受け取ったリクエストを JSON形式にする場合に有用な、マッピングテンプレートがあります。
        - データをAPI Gateway 側で加工できるものですね。

- Integration Response
    - バックエンドから返ってきたレスポンスに関する設定を行います.
        - 例えばHTTPステータスコードをマッピングしたり , レスポンスの内容の変換を行ったりします .
            - 何も行わないパススルーという動作もあります。
            - なお、この後のスライドで説明しますが , Integration Requestで非プロキシ統合をした場合にのみ設定可能です .

- Method Response
    - クライアントに応答する最終的なレスポンスを定義するものです。
        - デフォルトでは、APIエンドポイントが呼び出されると、200の正常な応答を返して成功するように設定されています。
        - ここで、APIやアプリケーションの仕様に応じて、ステータスコードを設定することができます。

- さて、統合リクエストを詳しくみていきましょう。

API Gateway統合



CloudTech

- 統合リクエストとは、API Gatewayから実際に処理をするアプリケーションにデータを繋ぐもので、いくつか種類があります。

- Lambda
  - Lambdaとのサーバーレス構成は定番です。

  - ここのチェックボックスをOFFにすると、Lambda 非プロキシ統合(カスタム統合)となります。
    - これは、先ほど解説したように、データを加工して次に渡す設定です。
  -
  - チェックボックスをONにすると、Lambda プロキシ統合となります。
    - 統合リクエスト・統合レスポンスの設定が不要な統合タイプです。
    - プロキシは単にリクエストを渡すだけですが、

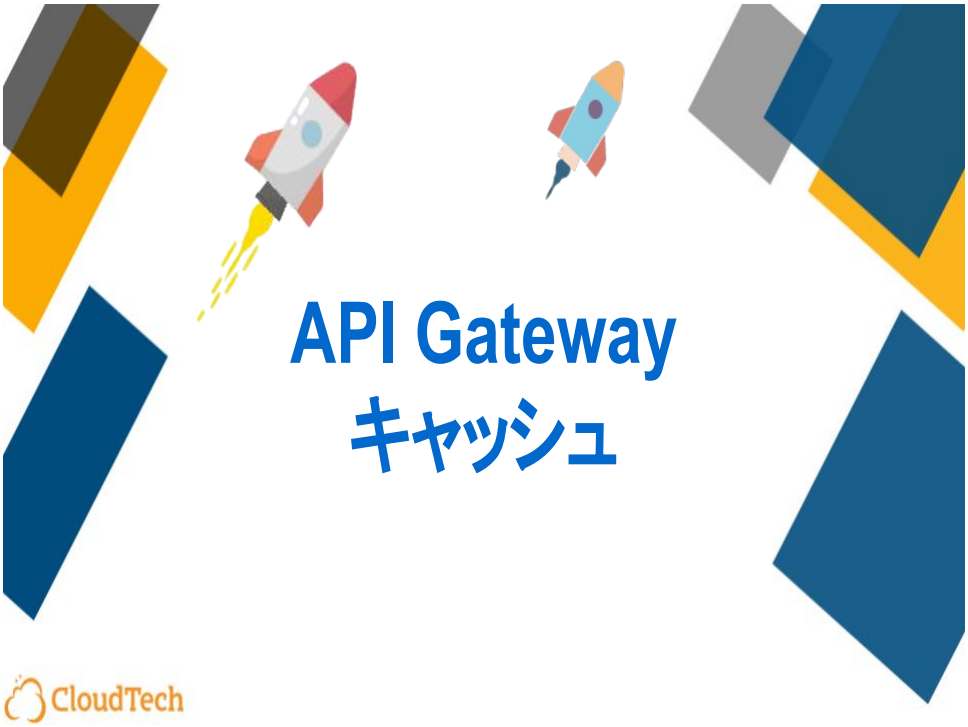
        - カスタムはより柔軟性があります。

- HTTP
- HTTP, またはHTTPS通信ができるエンドポイントに対して API Gatewayからリクエストを投げるものです。
    - 任意のURL, 任意のHTTPメソッドが利用可能です .
        - こちらも、非プロキシ統合、プロキシ統合があります。

- Mock
    - テスト用の設定です。
        - API Gatewayに届いたリクエストを、バックエンドに渡すことなく, そのまま統合レスポンスに渡す動きです。

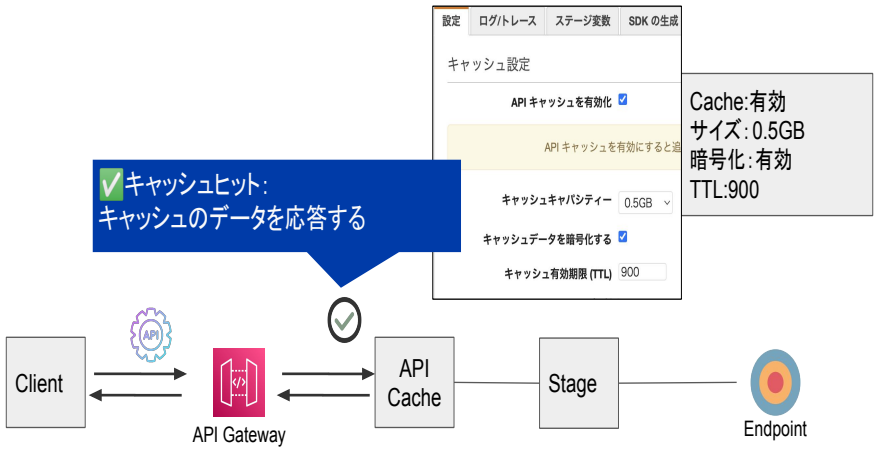
- AWS Service
    - Step Functions や SQS などといった、AWSサービスを呼び出します
        - 非プロキシタイプのみ

- VPC リンク
    - Priate link のエンドポイントに対してリクエストを投げるものです。



- 次は、リクエストとレスポンスの構造を見てみましょう。

API Gatewayキャッシュ

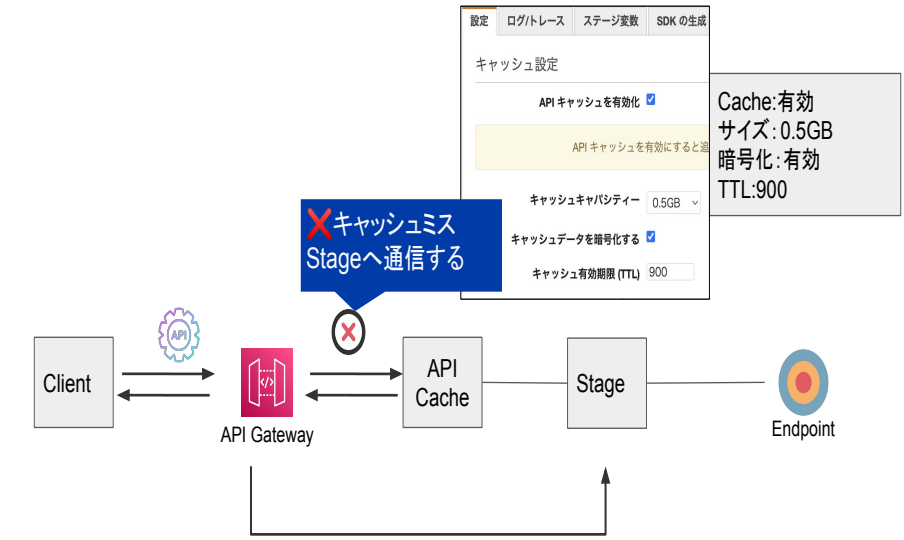


CloudTech

- APIのパフォーマンス向上に役立つ「キャッシュ」について見ていきましょう。
  - API Gatewayで、キャッシュを設定することで、エンドポイントからのレスポンスをキャッシュすることができます。
    - こうすることで、エンドポイントに通信する回数を減らすことができますし、リクエストの応答時間（レイテンシー）を減らすことができます。

- リクエストは、まずキャッシュをチェックして
  - キャッシュがあればキャッシュヒット。
  - キャッシュのデータを応答して完了です。

API Gatewayキャッシュ



CloudTech

- リクエストを受け付けて、
  - キャッシュがなければ、
  - ステージに進み、
  - その後レスポンスが返ってきます

    - ※ここでいうステージとは、APIをデプロイする先の環境という意味です。
      - APIGatewayは、ステージを分けて運 用することができます。
      - 例えば開発環境用や、本番環境用 など。ステージを変えるとAPIのURLのパスが変わる点に注意 です。

- 次のリクエストから、情報がキャッシュされ、対象のリクエストはキャッシュから返されるようになり、レイテンシーが短縮されます。
  - キャッシュに保持するサイズや暗号化の有無を指定

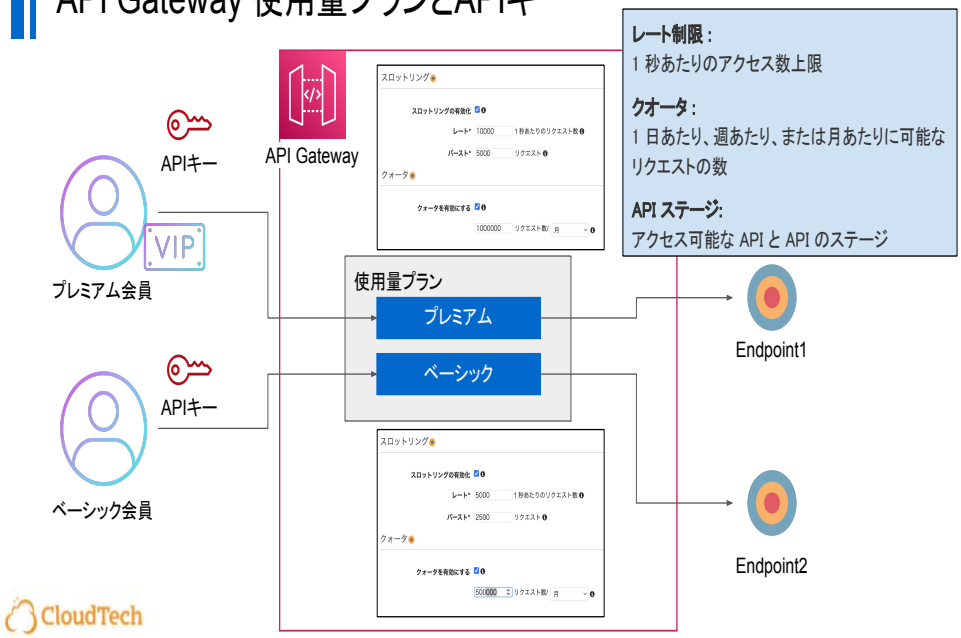
- した上で、TTL（Time to Live：有効期限）が切れるまでレスポンスをキャッシュする動きとなります。

API Gateway 使用量プランと APIキー

CloudTech

-

API Gateway 使用量プランとAPIキー



- まず、実例です。
- 使用量プランとAPIキーについて
  - 例えば、提供しているサービスで、有料のプレミアム会員と、
    - 無料のベーシック会員を設定しているとします。
    - プレミアム会員には、より質の高いサービスを提供すべきなので、APIのレート（例：1 秒あたりのアクセス数上限）を高い閾値に設定するなど、品質を分けることができます。
    - ベーシック会員は、品質を低く設定するなど、分けることができます。
    - スロットリングの設定、とも言いますが、APIへ対してアクセスできる量と速度を設定することです。
  - それでは、どうやって、受け付けたリクエストがプレミアム会員からのものか？ベーシック会員からのもの

    - か判別するのでしょうか。
        - APIキーを使用します。キーとは、実態は文字列ですが、この文字列情報を利用して、リクエストヘッダに入っている API キーによって判定します。
            - これで、APIキーに基づいた区別ができるようになりました。
            - 異なるステージや、異なるエンドポイントに接続することができます。
- また、ステージごとにメソッドごとのスロットリングの制限を設定することも可能です。

制限はこのように、

- レート制限：
  1 秒あたりのアクセス数上限
- クオータ：
  1 日あたり、週あたり、または月あたりに可能なリクエストの数
- API ステージ:
  アクセス可能な API と API のステージ

を設定できます。

次のスライドで詳細に解説しますが、ここで SAAの問題をひとつ解いてみましょう。

ある企業が、Amazon API GatewayおよびAWS Lambdaを使用するパブリックアクセス可能なサーバーレスアプリケーションを実行しています。最近ボットネットからの悪質なリクエストにより、アプリケーションでトラフィックスパイクが発生するようになりました。どのように対処できますか？というもの。
正規ユーザーとのみ共有されている APIキーを使用して使用プランを作成する。

API Gatewayにアクセス制限を掛ける方法として、APIキーを使

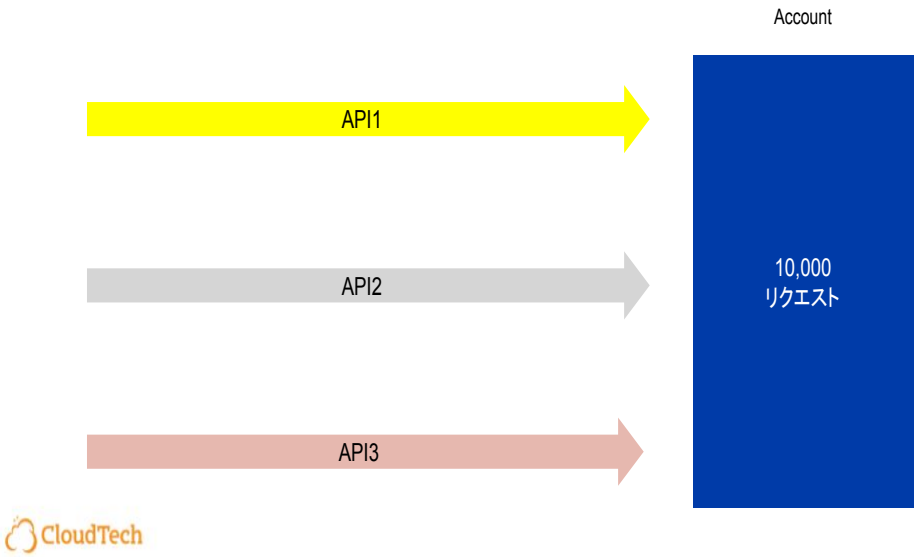
う方法があります。
単純にAPIとAPIキーを紐付けるのではなく、「使用量プラン」を介して
APIとキーは結びつける方法は、試験にも出てくるので覚えておきま
しょう。

API Gateway
スロットリング

CloudTech

-

API Gatewayスロットリング（デフォルト）



- **AWS アカウントレベルのスロットリング ：**
  デフォルトでは、API Gateway はリクエストのレート (1 秒あたりのリクエスト数上限）を 10,000 リクエスト / 秒に制限しています。

  また、バースト (1 ミリ秒あたりのアクセス数上限) を AWS アカウント内のすべての API にわたって 5,000 リクエストに制限しています。

  リクエストやバーストの上限数を超えた場合には、HTTP ステータス 429 (Too Many Requests) エラーが返されます。

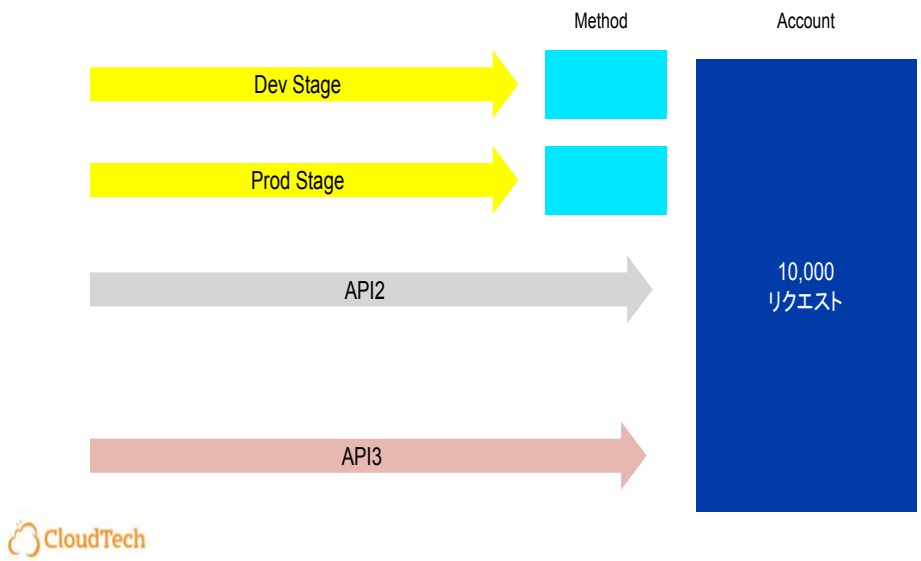
  これがデフォルトの設定です。

- **ステージ/メソッド：**
  特定のステージまたは API の個別のメソッドで、AWS アカウントレベルのスロットリング制限をオーバーライド (上書き) し、上限値を設定できます。

  ただし、AWS アカウントレベルのスロットリングの上限値を超えることはできません。

- **API キー：**
  クライアントの API キーごとに「使用量プラン」に基づいて、スロットリング制限を実施できます。
- **キー/メソッド：**
  クライアントの「使用量プラン」に基づいて、メソッドレベルでの追加のスロットリング制限を実施できます。

API Gatewayスロットリング（Stage/Method）



- **AWS アカウントレベルのスロットリング ：**
  デフォルトでは、API Gateway はリクエストのレート (1 秒あたりのリクエスト数上限）を 10,000 リクエスト / 秒に制限しています。

  また、バースト (1 ミリ秒あたりのアクセス数上限) を AWS アカウント内のすべての API にわたって 5,000 リクエストに制限しています。

  リクエストやバーストの上限数を超えた場合には、HTTP ステータス 429 (Too Many Requests) エラーが返されます。

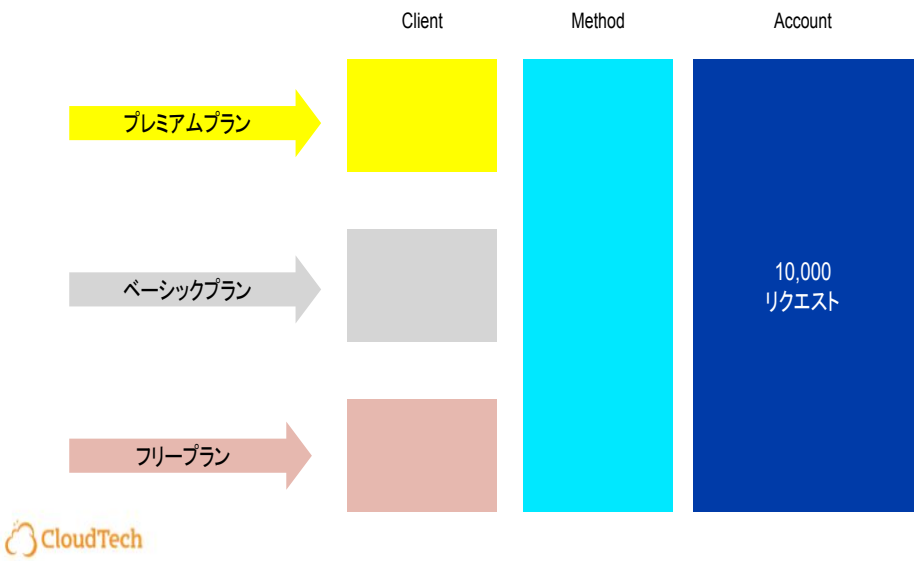
  これがデフォルトの設定です。

- **ステージ/メソッド：**
  特定のステージまたは API の個別のメソッドで、AWS アカウントレベルのスロットリング制限をオーバーライド (上書き) し、上限値を設定できます。

  ただし、AWS アカウントレベルのスロットリングの上限値を超えることはできません。

- **API キー：**
  クライアントの API キーごとに「使用量プラン」に基づいて、スロットリング制限を実施できます。
- **キー/メソッド：**
  クライアントの「使用量プラン」に基づいて、メソッドレベルでの追加のスロットリング制限を実施できます。

API Gatewayスロットリング（APIキー/使用量プラン）



- **AWS アカウントレベルのスロットリング ：**
  デフォルトでは、API Gateway はリクエストのレート (1 秒あたりのリクエスト数上限）を 10,000 リクエスト / 秒に制限しています。

  また、バースト (1 ミリ秒あたりのアクセス数上限) を AWS アカウント内のすべての API にわたって 5,000 リクエストに制限しています。

  リクエストやバーストの上限数を超えた場合には、HTTP ステータス 429 (Too Many Requests) エラーが返されます。

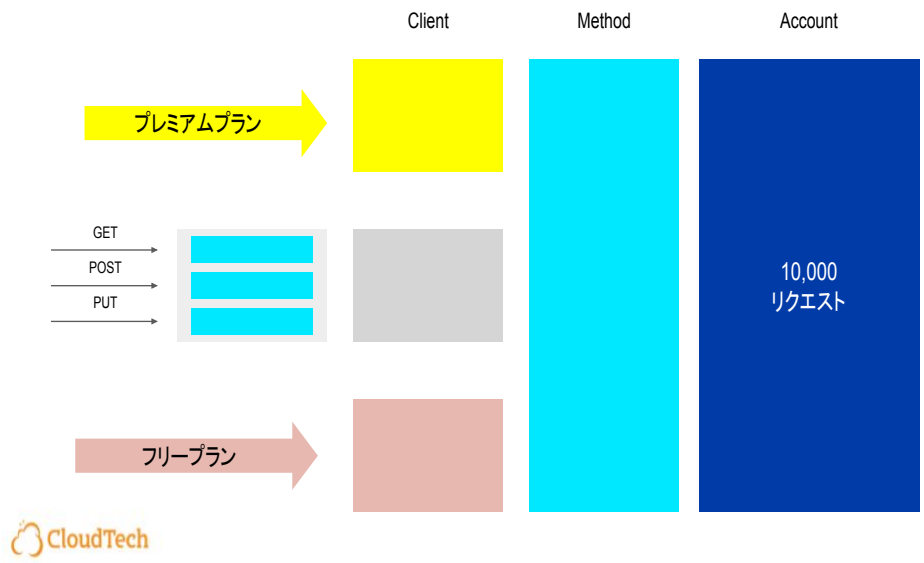
  これがデフォルトの設定です。

- **ステージ/メソッド：**
  特定のステージまたは API の個別のメソッドで、AWS アカウントレベルのスロットリング制限をオーバーライド (上書き) し、上限値を設定できます。

  ただし、AWS アカウントレベルのスロットリングの上限値を超えることはできません。

- **API キー：**
  クライアントの API キーごとに「使用量プラン」に基づいて、スロットリング制限を実施できます。
- **キー/メソッド：**
  クライアントの「使用量プラン」に基づいて、メソッドレベルでの追加のスロットリング制限を実施できます。

API Gatewayスロットリング（APIキー/Method）



- **AWS アカウントレベルのスロットリング ：**
  デフォルトでは、API Gateway はリクエストのレート (1 秒あたりのリクエスト数上限）を 10,000 リクエスト / 秒に制限しています。

  また、バースト (1 ミリ秒あたりのアクセス数上限) を AWS アカウント内のすべての API にわたって 5,000 リクエストに制限しています。

  リクエストやバーストの上限数を超えた場合には、HTTP ステータス 429 (Too Many Requests) エラーが返されます。

  これがデフォルトの設定です。

- **ステージ/メソッド：**
  特定のステージまたは API の個別のメソッドで、AWS アカウントレベルのスロットリング制限をオーバーライド (上書き) し、上限値を設定できます。

  ただし、AWS アカウントレベルのスロットリングの上限値を超えることはできません。

- **API キー：**
  クライアントの API キーごとに「使用量プラン」に基づいて、スロットリング制限を実施できます。
- **キー/メソッド：**
  クライアントの「使用量プラン」に基づいて、メソッドレベルでの追加のスロットリング制限を実施できます。

■公式ドキュメント API Gateway API に対してセットアップするエンドポイントタイプを選択する
https://docs.aws.amazon.com/ja_jp/apigateway/latest/developerguide/api-gateway-api-endpoint-types.html

■外部ブログ [API Gateway] APIキーと使用量プランを使用してアクセス制限を掛ける
https://dev.classmethod.jp/articles/api-gateway-api-key-plan-access/

■公式ブログ Creating a machine learning-powered REST API with Amazon API Gateway mapping templates and Amazon SageMaker
https://aws.amazon.com/jp/blogs/machine-learning/creating-a-machine-learning-powered-rest-api-with-amazon-api-gateway-mapping-templates-and-amazon-sagemaker/

■公式ブログ Building well-architected serverless applications: Regulating inbound request rates – part 1
https://aws.amazon.com/jp/blogs/compute/building-well-architected-serverless-applications-regulating-inbound-request-rates-part-1/