

【AWS Hands-on for Beginners】

# Serverless #1

オリジナルAPIの作成を通してサーバーレスの基本を学ぼう

アマゾンウェブサービスジャパン株式会社

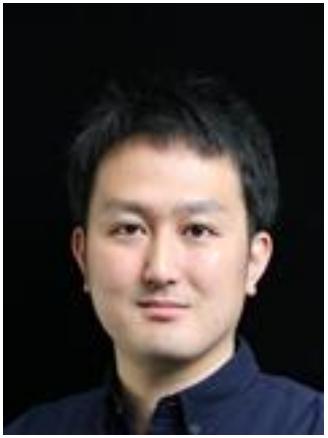
Solutions Architect

金澤 圭 / Kei Kanazawa

(収録日: 2019/10/24)



# 自己紹介



- 名前: 金澤 圭  @ketancho
- 役割: Technical Solutions Architect
- 前職:
  - お客様先で新規事業や新規プロダクトの開発をするエンジニア
  - AWS をはじめとした技術を教えることが好きです
- 好きな AWS サービス
  -  AWS Lambda

# AWS Hands-on for Beginners とは



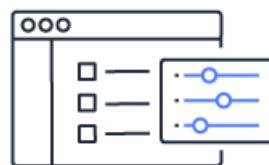
実際に手を動かしながら AWS の各サービスを学んでいただきます



初めてそのサービスを利用される方がメインターゲットです



お好きな時間、お好きな場所でご受講いただけるオンデマンド形式



テーマごとに合計1~2時間の内容 & 細かい動画に分けて公開  
スキマ時間の学習や、興味のある部分だけの聴講も可能

# 内容についての注意点

- 本資料では2019年10月24日収録時点のサービス内容および価格についてご説明しています。最新の情報は AWS公式ウェブサイト(<http://aws.amazon.com>)にてご確認ください。資料作成には十分注意しておりますが、資料とAWS公式ウェブサイトとで記載内容に相違があった場合、AWS公式ウェブサイトの記載を優先させていただきます。
- マネージメントコンソールについても、収録時点のものとなります。差異がある場合がございますので、ご注意ください。
- ハンズオンでは、AWS の各種サービスの利用、リソースの作成を行います。無料枠を超えるコースもございますが、その場合、ご利用料金が発生することをあらかじめご認識ください。
- 学習後のリソースの削除についても、お客様の責任でご実施いただくようお願いいたします。

# 本コースのゴール/前提条件・知識

- 本コースのゴール
  - AWS Lambda, Amazon API Gateway, Amazon DynamoDB の基本を学ぶ
  - 上記のサービスを組み合わせて、サーバーレスな Web API を作成する
- 本コースの前提条件・知識
  - AWS アカウントをお持ちであること
    - ハンズオンの作業が同一AWSアカウントの他のリソースに影響が出る場合があります。
    - ハンズオン用にAWSアカウントを取得していただくことをオススメします。
  - Python を書いたことがあること（必須ではありません）

# このコースの Agenda

1 ) Serverless アーキテクチャの概要

2 ) AWS Lambda の紹介とハンズオン

2 – 1 ) AWS Lambda の概要

2 – 2 ) AWS Lambda ハンズオン① Lambda を単体で使ってみる

2 – 3 ) AWS Lambda ハンズオン② 他のサービスを呼び出してみる

3 ) Amazon API Gateway の紹介とハンズオン

3 – 1 ) Amazon API Gateway の概要

3 – 2 ) Amazon API Gateway ハンズオン① API Gateway を単体で使ってみる

3 – 3 ) Amazon API Gateway ハンズオン② API Gateway と Lambda を組み合わせる

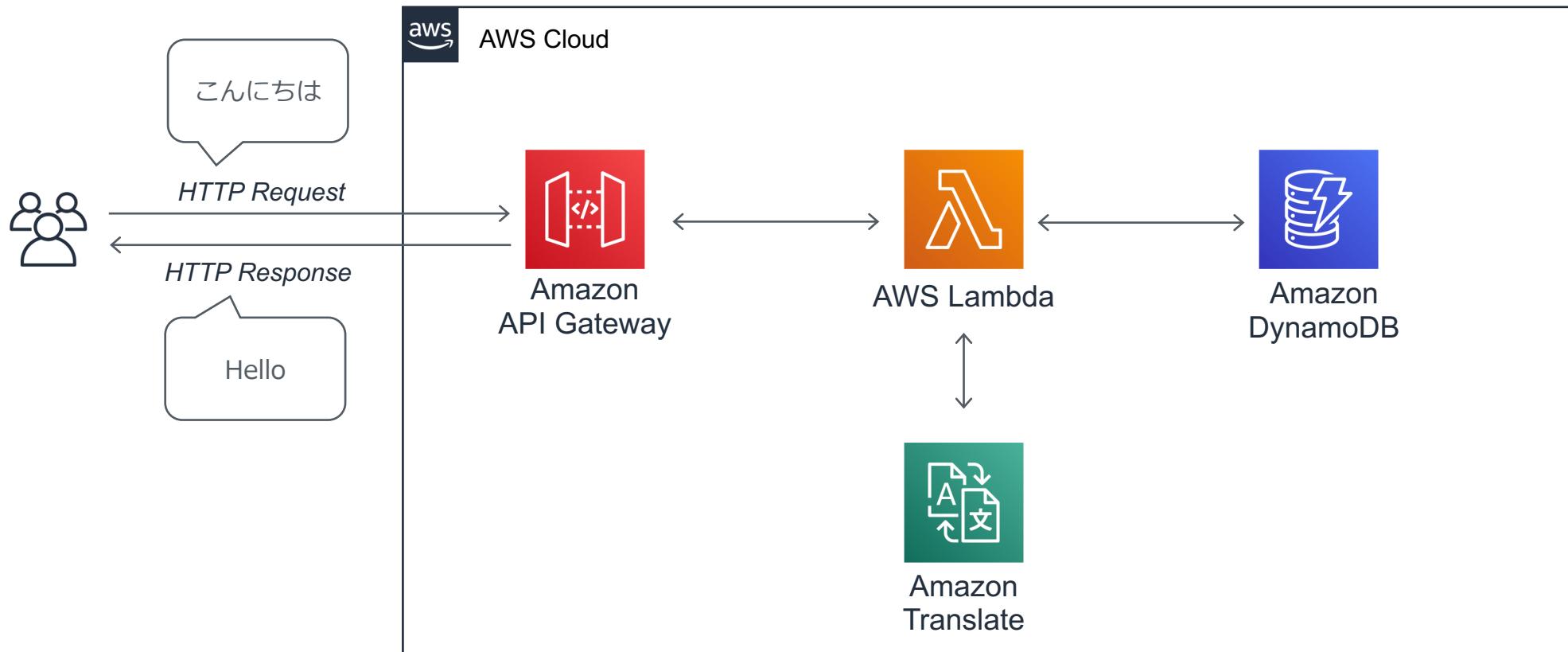
4 ) Amazon DynamoDB の紹介とハンズオン

4 – 1 ) Amazon DynamoDB の概要

4 – 2 ) Amazon DynamoDB ハンズオン① テーブルを作ってみる

4 – 3 ) Amazon DynamoDB ハンズオン② API Gateway と Lambda と DynamoDB を組み合わせる

# このコースで構築するアーキテクチャ



# 1) Serverless アーキテクチャの概要

# なぜサーバーレスなのか？

- 開発者の方がやりたいことは何か？



サーバーの管理



エンドユーザーに価値を届ける

- しかし、ビジネスの価値に直接は繋がらないが、必要な作業はたくさん..



サーバーの  
セットアップ



ミドルウェアや  
ランタイムの  
セットアップ



セキュリティ  
パッチの適用



耐障害性を確保する  
アーキテクチャの検討

# サーバーレスアーキテクチャの特徴



インフラのプロビジョニングや  
管理が不要



自動でスケール

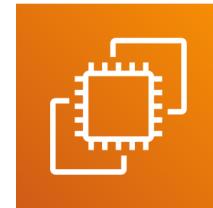


高い可用性



価値に対する支払い

# AWS における Compute サービス



Amazon EC2



Container Services



AWS Lambda

自由度

高い😊



低い😢

管理の手間

多い😢

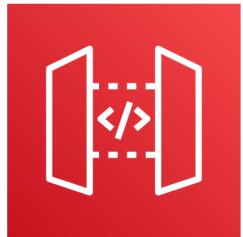


少ない😊

# サーバーレスアーキテクチャでよく使われるサービス



AWS Lambda



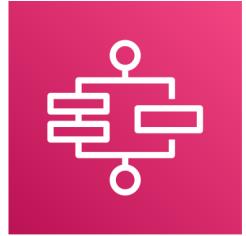
Amazon API  
Gateway



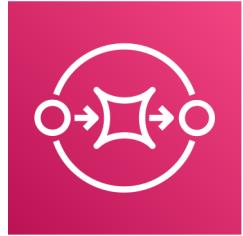
Amazon  
DynamoDB



Amazon Simple  
Storage Service  
(S3)



AWS Step  
Functions



Amazon Simple  
Queue Service  
(SQS)

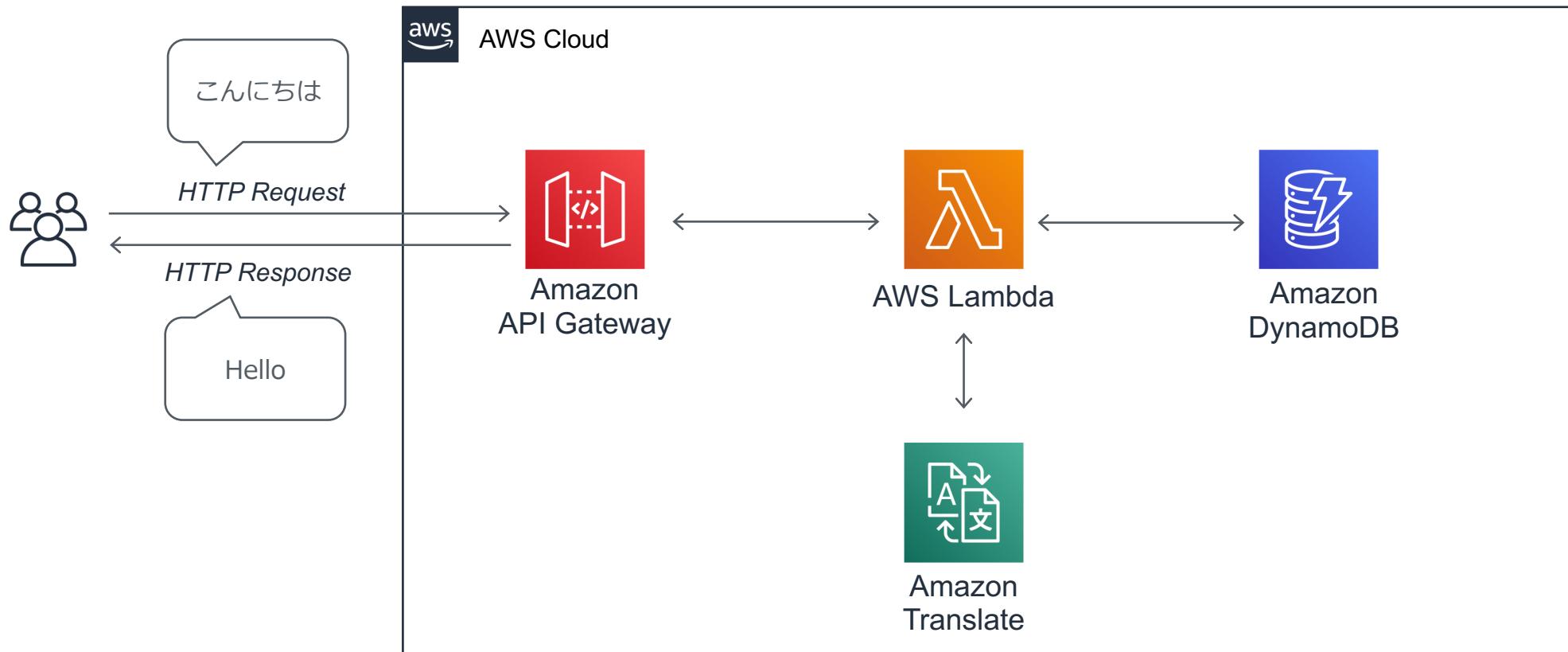


Amazon  
CloudWatch



AWS X-Ray

# このコースで構築するアーキテクチャ



## 2) AWS Lambda の紹介とハンズオン

## 2 – 1 ) AWS Lambda の概要

# AWS Lambda の特徴

- ・ サーバーのプロビジョニング/管理なしでプログラムを実行できるサービス
- ・ コードの実行やスケーリングに必要なことは、Lambda 側で実施するので、開発者の方はコードを書くことにより集中できる
- ・ リクエストベースの料金体系



実行回数  
(無料枠あり)

+



実行時間  
(単価は確保したメモリによる)  
(無料枠あり)

# AWS Lambda におけるコーディングイメージ

- 対応言語
  - Java、Go、PowerShell、Node.js、C#、Python、Ruby ※2019/10 現在
  - サポートされていない言語は、カスタムランタイムを実装することで利用可能
- ハンドラーで呼び出す関数を指定する
  - デフォルトでは lambda\_function.lambda\_handler

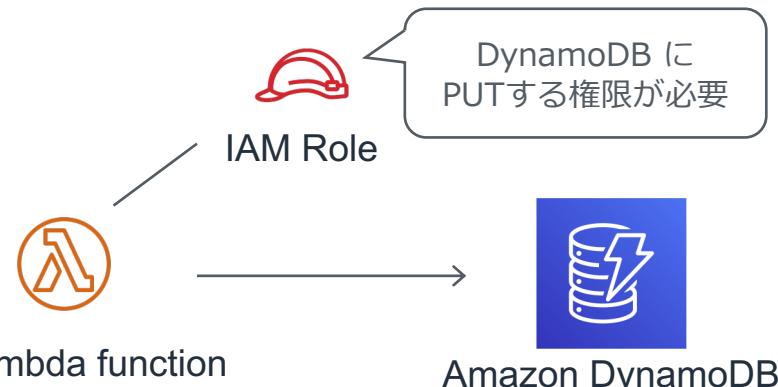
The screenshot shows the AWS Lambda Function Editor interface. At the top, there are three configuration dropdowns: 'コード エントリ タイプ' (Code entry type) set to 'コードをインラインで編集' (Edit inline), 'ランタイム' (Runtime) set to 'Python 3.7', and 'ハンドラ 情報' (Handler information) set to 'lambda\_function.lambda\_handler'. Below these, the main workspace displays a file structure on the left with 'Environment' and a folder named 'test' containing a file named 'lambda\_function.py'. The code editor window shows the Python script:

```
1 import json
2 import urllib.parse
3 import boto3
4
5 s3 = boto3.client('s3')
6
7 def lambda_handler(event, context):
8
9     bucket = event['Records'][0]['s3']['bucket']['name']
10    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')
```

# AWS Lambda で設定できる項目

※一部抜粋、設定値は2019/10現在

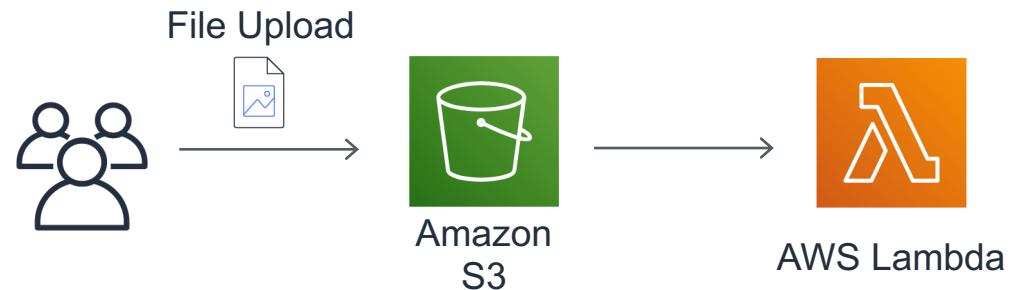
- 確保するメモリの量
  - 128MB ~ 3,008MB (64MBごと)
  - CPU 能力は確保するメモリの量に比例
- タイムアウト値
  - 最大で 900秒
- 実行 IAM ロール



The screenshot shows the AWS Lambda function configuration interface. On the left, under '実行ロール' (Execution Role), it says: '関数のアクセス権限を定義するロールを選択します。カスタムロールを作成するには、[IAM コンソール](#)に移動します。' (Select the role that defines the function's access permissions. To create a custom role, move to the [IAM console](#)). A dropdown menu is open, showing '既存のロールを使用する' (Use existing role) and a list item 'service-role/demo-lambda-translate...'. On the right, under '基本設定' (Basic Settings), there is a '説明' (Description) input field, a 'メモリ (MB)' (Memory (MB)) slider set to 128 MB, and a 'タイムアウト' (Timeout) section with inputs for 0 minutes and 10 seconds.

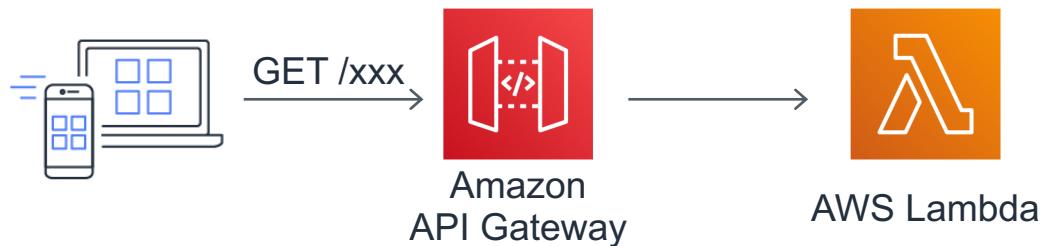
# AWS Lambda のイベントソースと呼び出しタイプ<sup>†</sup>

## ◆非同期呼び出しの例



Lambdaへのリクエストが  
正常に受け付けられたかどうかのみを返却

## ◆同期呼び出しの例



Lambdaの実行完了時にレスポンスが返却

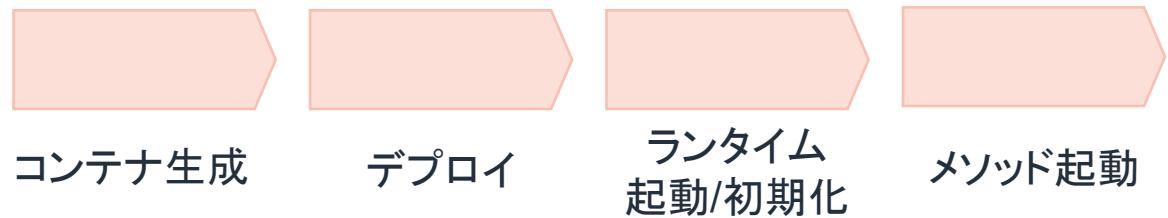
参考) BlackBelt シリーズ: AWS Lambda Part1

[https://d1.awsstatic.com/webinars/jp/pdf/services/20190402\\_AWSBlackbelt\\_AWSLambda%20Part1%262.pdf](https://d1.awsstatic.com/webinars/jp/pdf/services/20190402_AWSBlackbelt_AWSLambda%20Part1%262.pdf)

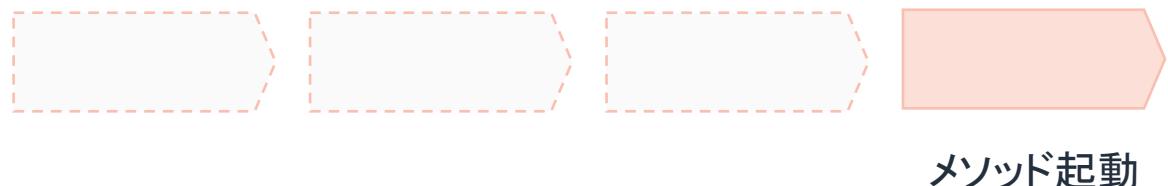
# Lambda Function のライフサイクル

- Lambda は呼び出されると、コンテナ上でプログラムが実行する
- 1つのコンテナで同時に実行できるのは1つのリクエストまで
- コンテナは再利用されるが、利用可能なコンテナがないときはコールドスタート

## ◆コールドスタートの場合



## ◆ウォームスタートの場合



## 2 – 2) AWS Lambda ハンズオン① Lambda を単体で使ってみる

# AWS Lambda ハンズオン① Lambda を単体で使ってみる

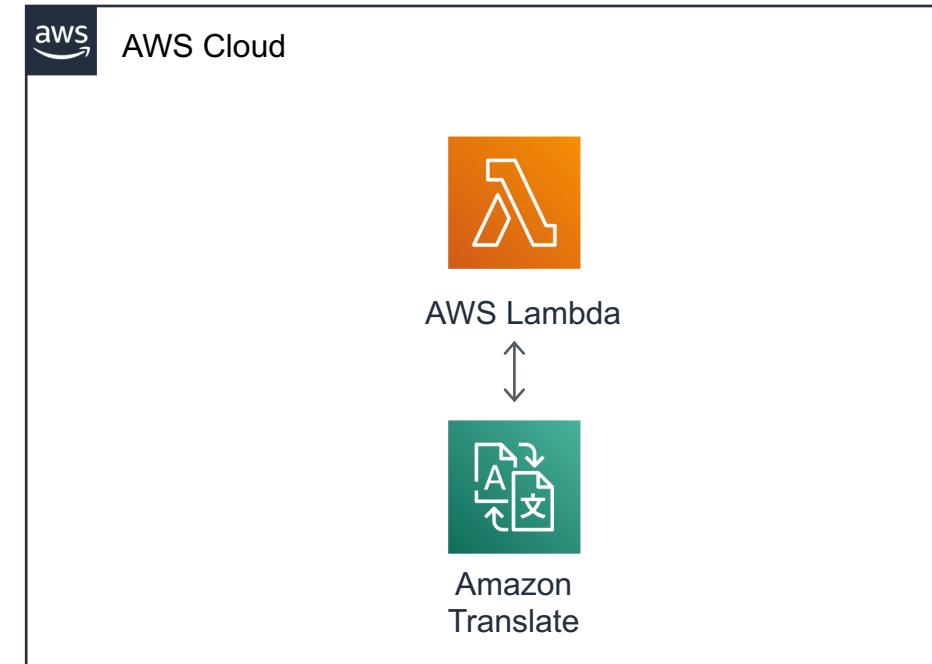
- 簡単な Lambda Function を作成し、基本的な使い方を理解する
  - Lambda Function を「一から作成」する
  - 確保するメモリの量、タイムアウト値の変更をする
  - 自動的に作成される IAM ロールについて確認する
  - Lambda 関数をテスト実行する
  - ログを出すようにプログラムを修正して保存する
  - もう1度、テスト実行する
  - ログの確認をする

## 2 – 3) AWS Lambda ハンズオン② 他のサービスを呼び出してみる

# Lambda ハンズオン② 他のサービスを呼び出してみる

- Lambda 関数から、他の AWS サービスを呼び出す
- このコースでは、この Lambda 関数を育てていき、最終的に API 化する
  - Python SDK のドキュメントを見ながら、Translate の呼び出し方を実装する
  - IAM ロールを修正する
  - レスポンスとして、JSON 形式で英語訳を返すようにする
  - Lambda 関数をテスト実行する

◆現時点の構成図



# 3) Amazon API Gateway の紹介とハンズオン

## 3 – 1) Amazon API Gateway の概要

# 本ハンズオンで対象とする“API”

Application Programming Interface:

プログラムやソフトウェア同士がやり取りするための取り決め・仕様



言語処理系API  
(SDK、ライブラリなど)



ネットワーク  
呼び出しAPI

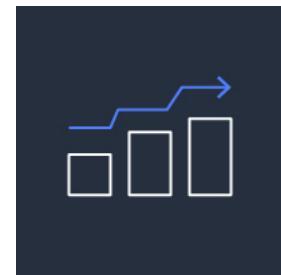


Web API

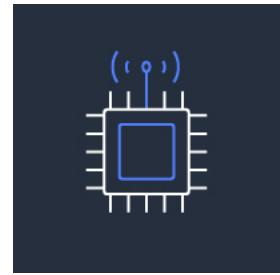
# Web API を開発するときに考えるべきこと



API自体の  
開発



高可用性/  
スケーラビリティ設計



インフラの管理



APIキーの  
管理



デプロイ管理

...

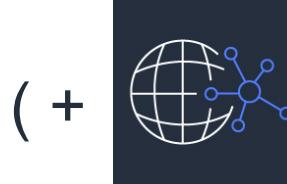
開発者はここに注力したい！

# Amazon API Gateway の特徴

- ・ サーバーをプロビジョニング/管理することなく、APIを作成・管理できるマネージドサービス
- ・ 可用性の担保、スケーリング、API キー管理といった API 開発で必要なことを API Gateway に任せることで、開発者は**ビジネスの差別化に繋がる作業に集中**できる
- ・ REST API と WebSocket に対応
- ・ リクエストベースの料金体系 (REST API の場合)



実行回数  
(AWSサインアップ日から  
12ヶ月間、無料枠あり)

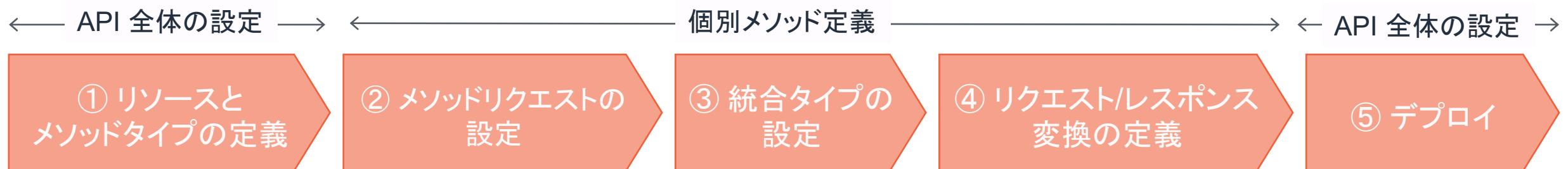


( + )

キャッシュ  
メモリ量 + データ転送料金

# API Gateway の使い方

- API Gateway には多種多様な設定項目がある
- 今回のハンズオンでは REST API の開発を行います
- ハンズオンの流れにそって API Gateway を用いた API 開発の主要な流れを説明します



# API Gateway の使い方



## ◆API の例

- ・ユーザー一覧を取得
- ・ユーザーIDを指定して、ユーザー情報を取得
- ・レポートの一覧を取得
- ・レポートを投稿する
- ・レポートIDを指定して、レポート情報を取得
- ・レポートIDを指定して、レポートを削除する

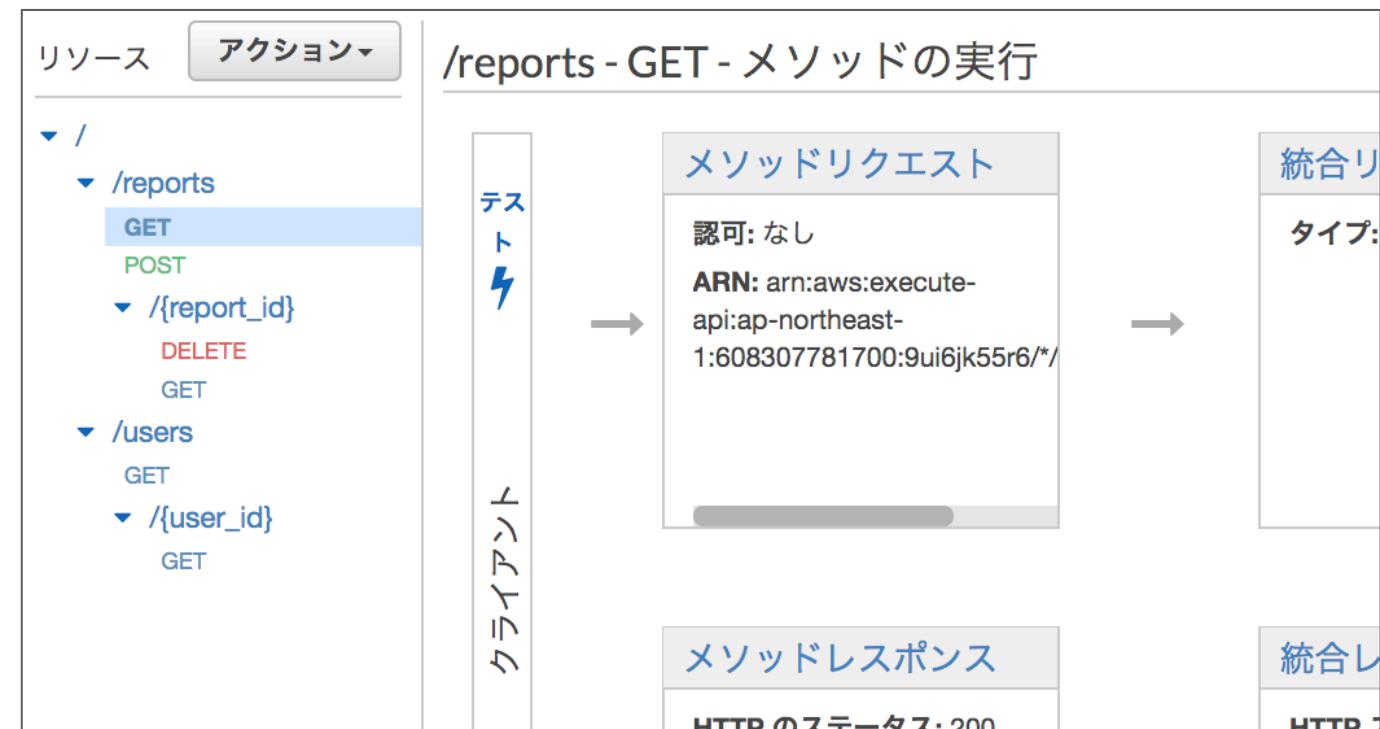
リソース: /

リソース: /reports **GET, POST**

リソース: /{report\_id} **GET, DELETE**

リソース: /users **GET**

リソース: /{user\_id} **GET**



# API Gateway の使い方

① リソースと  
メソッドタイプ定義

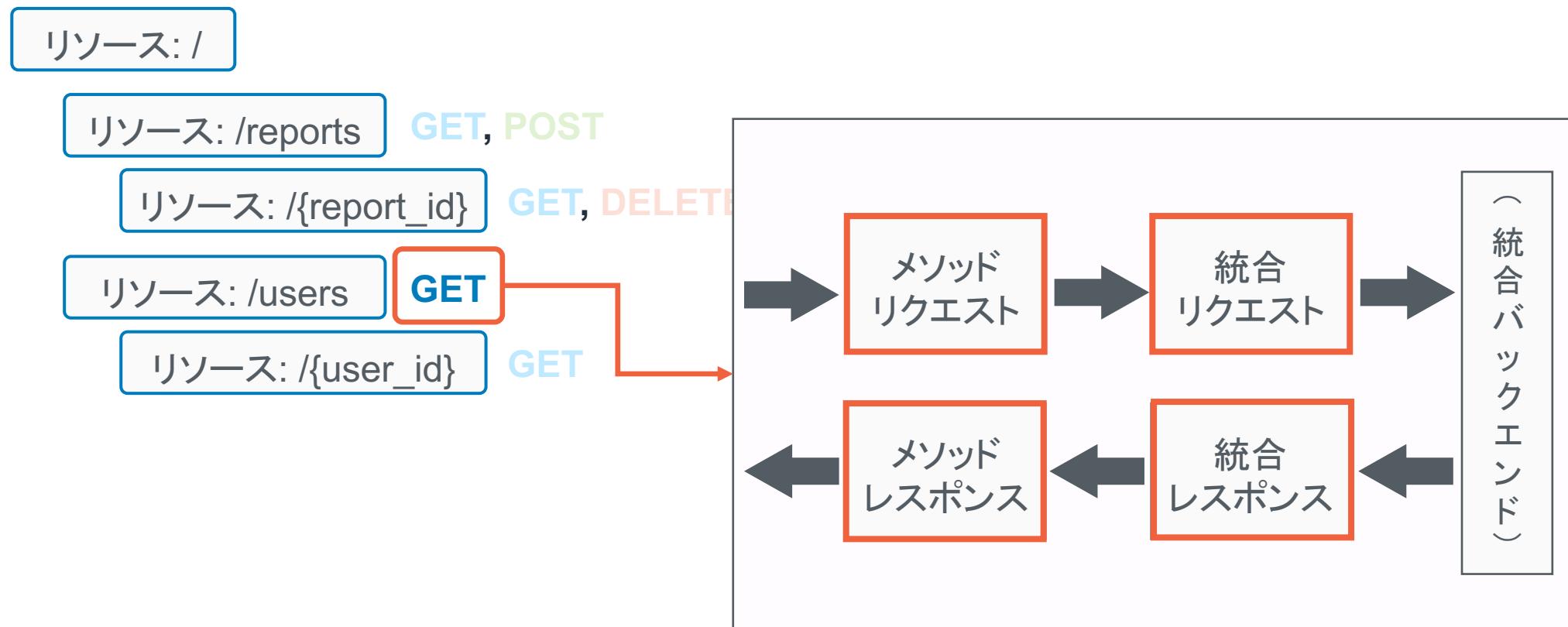
② メソッドリクエスト  
設定

③ 統合タイプ  
設定

④ Req/Res変換  
定義

⑤ デプロイ

- リソース × メソッドごとに API のフローを定義する



# API Gateway の使い方

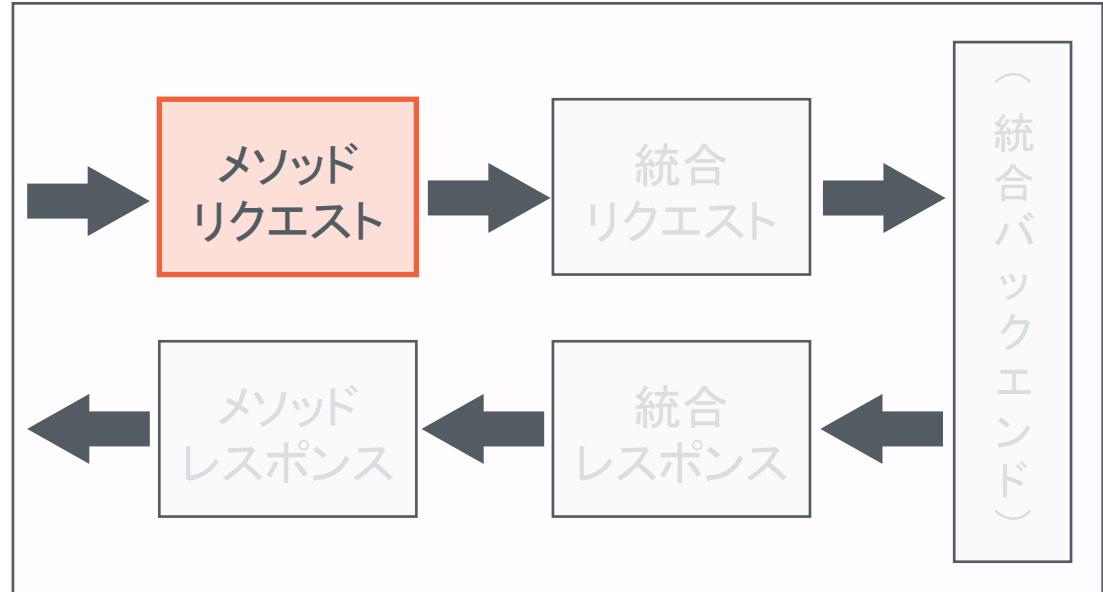
① リソースと  
メソッドタイプ定義

② メソッドリクエスト  
設定

③ 統合タイプ  
設定

④ Req/Res変換  
定義

⑤ デプロイ



- リクエストの受付に関する設定を行う

- 認証の設定
- 受け付けるクエリパラメータ
- 必須とするHTTPヘッダの設定

認可 なし

リクエストの検証 なし

API キーの必要性 false

▼ URL クエリ文字列パラメータ

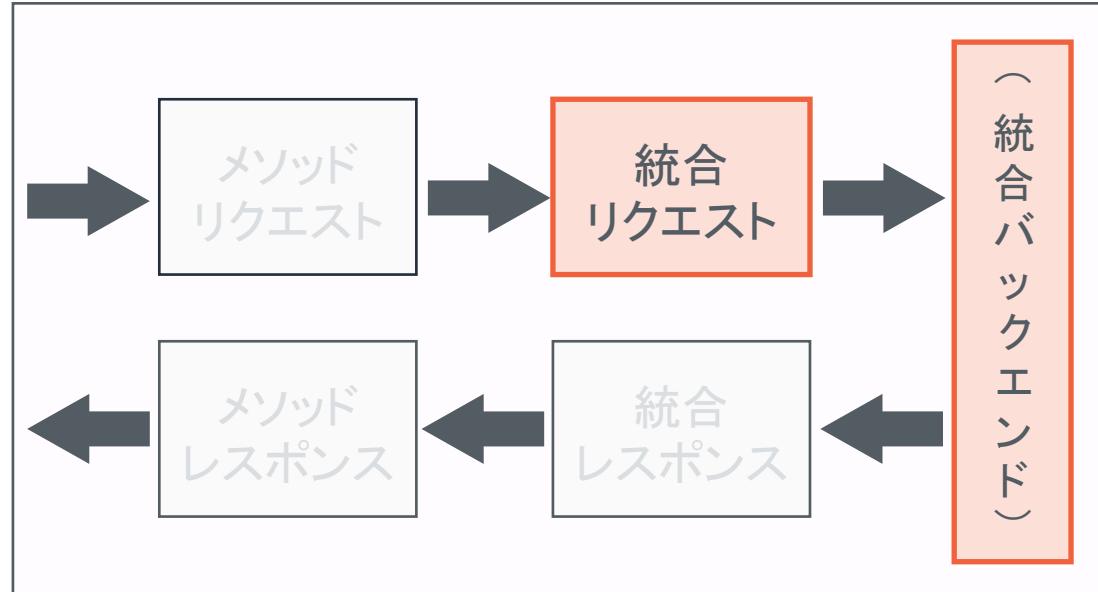
| 名前         | 必須                                  | キャッシュ                    |
|------------|-------------------------------------|--------------------------|
| input_text | <input checked="" type="checkbox"/> | <input type="checkbox"/> |

クエリ文字列の追加

▼ HTTP リクエストヘッダー



# API Gateway の使い方



- ・ バックエンドの種別を選択する

- Lambda 関数
- HTTP
- Mock
- AWS サービス
- VPC リンク

統合タイプ  Lambda 関数 [i](#)

HTTP [i](#)

Mock [i](#)

AWS サービス [i](#)

VPC リンク [i](#)

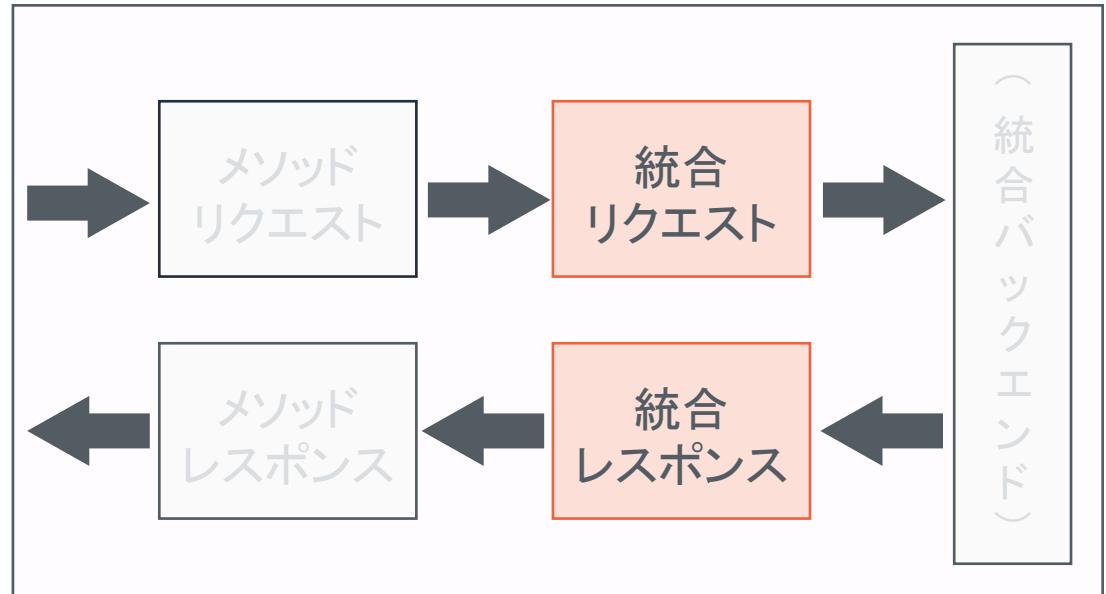
Lambda プロキシ統合の使用  [i](#)

Lambda リージョン ap-northeast-1 [編集](#)

Lambda 関数 demo-lambda-translate [編集](#)

# API Gateway の使い方

- ① リソースと  
メソッドタイプ定義
- ② メソッドリクエスト  
設定
- ③ 統合タイプ  
設定
- ④ Req/Res変換  
定義
- ⑤ デプロイ



- ・ バックエンドへの Input、バックエンドからの Output 変換することができる

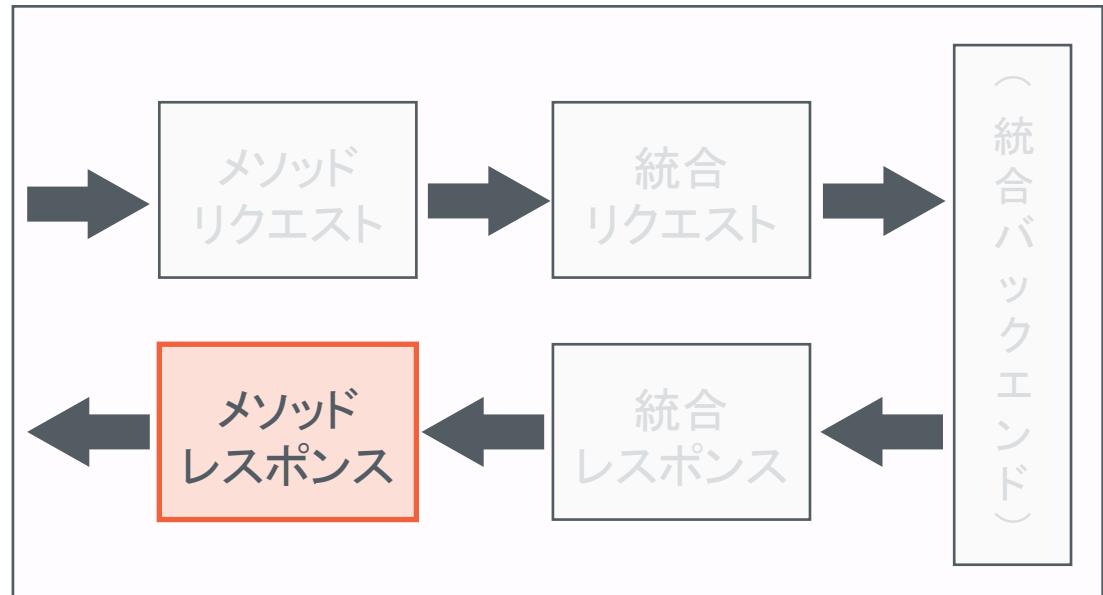
Generate template:

```
1 #set($inputRoot = $input.path('$'))  
2 {  
3     "track": "$input.params('track')"  
4 }  
5
```

- ・ プロキシ統合を利用してことで、  
変換せずにパススルーすることも可能

# API Gateway の使い方

- ① リソースと  
メソッドタイプ定義
- ② メソッドリクエスト  
設定
- ③ 統合タイプ  
設定
- ④ Req/Res変換  
定義
- ⑤ デプロイ



- リクエストに対する最終的な API Gateway としてのレスポンスに関する設定
  - ステータスコード
  - HTTPレスポンスヘッダ
  - など

The screenshot shows the configuration for an HTTP response:

**HTTP のステータス**

|       |                |                  |
|-------|----------------|------------------|
| ▼ 200 | 200 のレスポンスヘッダー | 200 のレスポンス本文     |
|       | 名前             | コンテンツタイプ         |
|       | ヘッダーなし         | application/json |
|       | + ヘッダーの追加      | モデル              |
|       |                | Empty            |
|       | + レスポンスマル型の追加  |                  |
|       |                |                  |
|       | + レスポンスの追加     |                  |

# API Gateway の使い方

- API ごとにデプロイ
- ステージを作成し、そのステージのみにデプロイ可能



# API Gateway のその他の機能

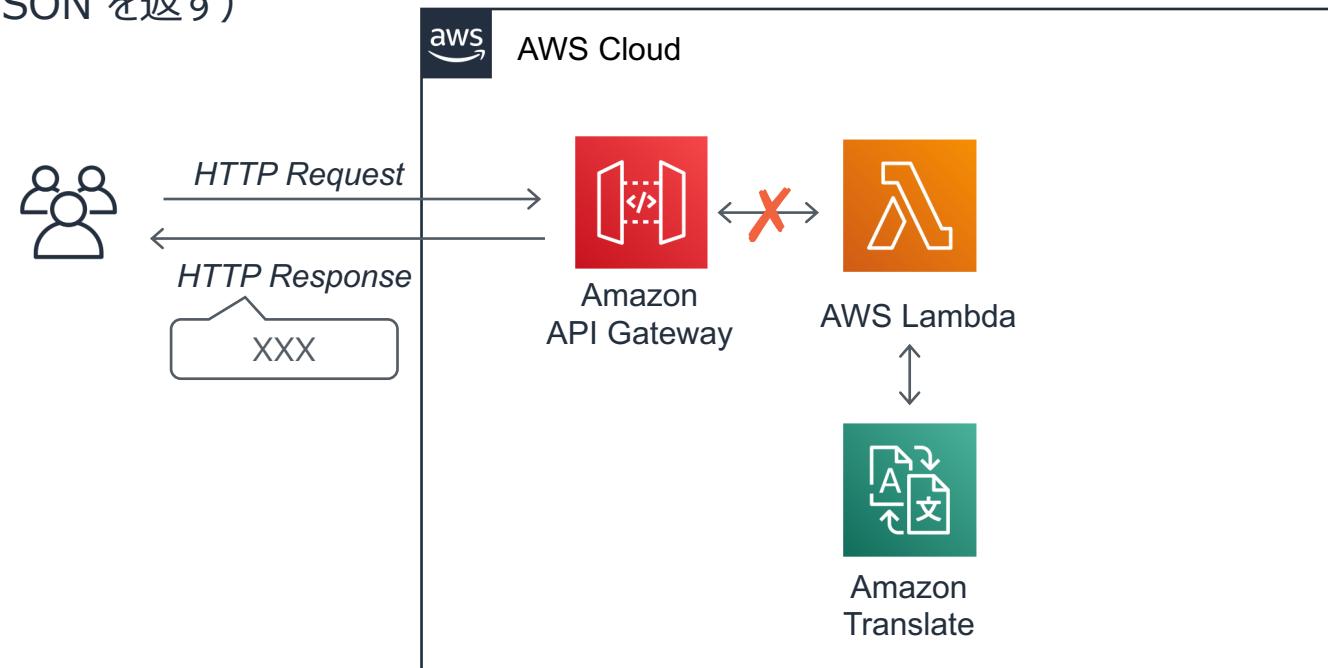
- バージョニング
- API キーと使用量プラン
- ログとモニタリング

## 3 – 2) Amazon API Gateway ハンズオン①

### ～ API Gateway を単体で使ってみる ～

# API Gateway ハンズオン① API Gateway を単体で使ってみる

- Mock データを返す API を作成する
  - 新規に API を作成する
  - sample リソースを作成し、GETメソッドを作成
  - 統合タイプとして Mock を選択する  
(まだ Lambda 関数との連携は行わず、固定の JSON を返す)
  - dev ステージにデプロイする
  - API を試しに実行してみる

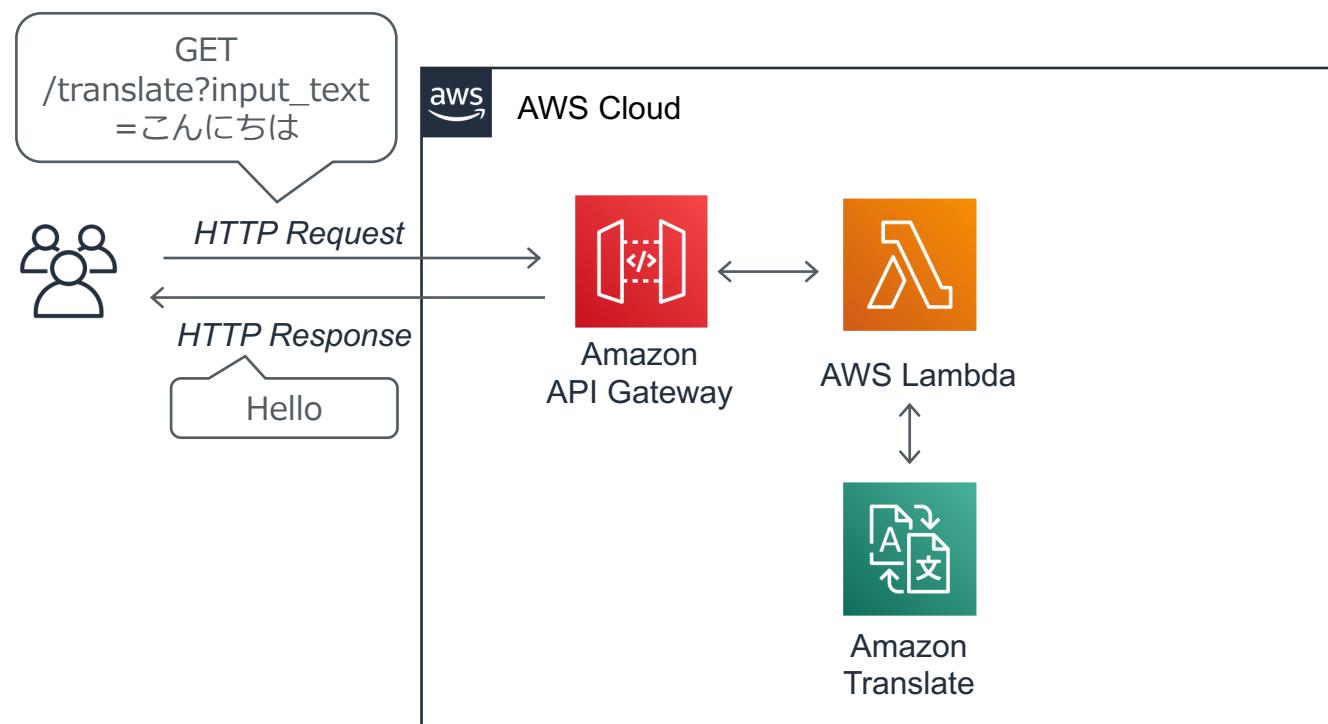


## 3 – 3) Amazon API Gateway ハンズオン②

### ～ API Gateway と Lambda を組み合わせる ～

# API Gateway ハンズオン② API Gateway と Lambda を組み合わせる

- 入力した日本語を英語に翻訳するAPIを作成する
  - /translate リソースを作成し、GET メソッドを作成する
  - 統合タイプとして Lambda 関数を選択する
  - プロキシ統合を設定し、Input / Output をパススルーする
  - メソッドリクエストでクエリパラメータの設定
  - Lambda 関数を修正する
  - Lambda 関数をテスト実行する
  - API をデプロイする
  - API を叩いてみる
    - 結果は返ってくるか？



# 4 ) Amazon DynamoDB の紹介とハンズオン

## 4 – 1 ) Amazon DynamoDB の概要

# AWS のデータベースサービス

※一部抜粋



Amazon RDS



Amazon Aurora



Amazon DynamoDB



Amazon Redshift



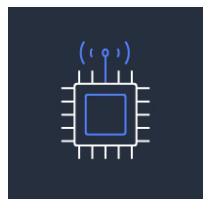
Amazon ElastiCache



Amazon Neptune

# Amazon DynamoDB の特徴

- ・フルマネージド型の NoSQL データベースサービス
- ・3つの Availability Zone に保存されるので信頼性が高い
- ・性能要件に応じて、テーブルごとにスループットキャパシティを定義するキャパシティの Auto Scaling、オンデマンドキャパシティといった設定も可能
- ・ストレージの容量制限がない
- ・料金体系（キャパシティを定義する場合）

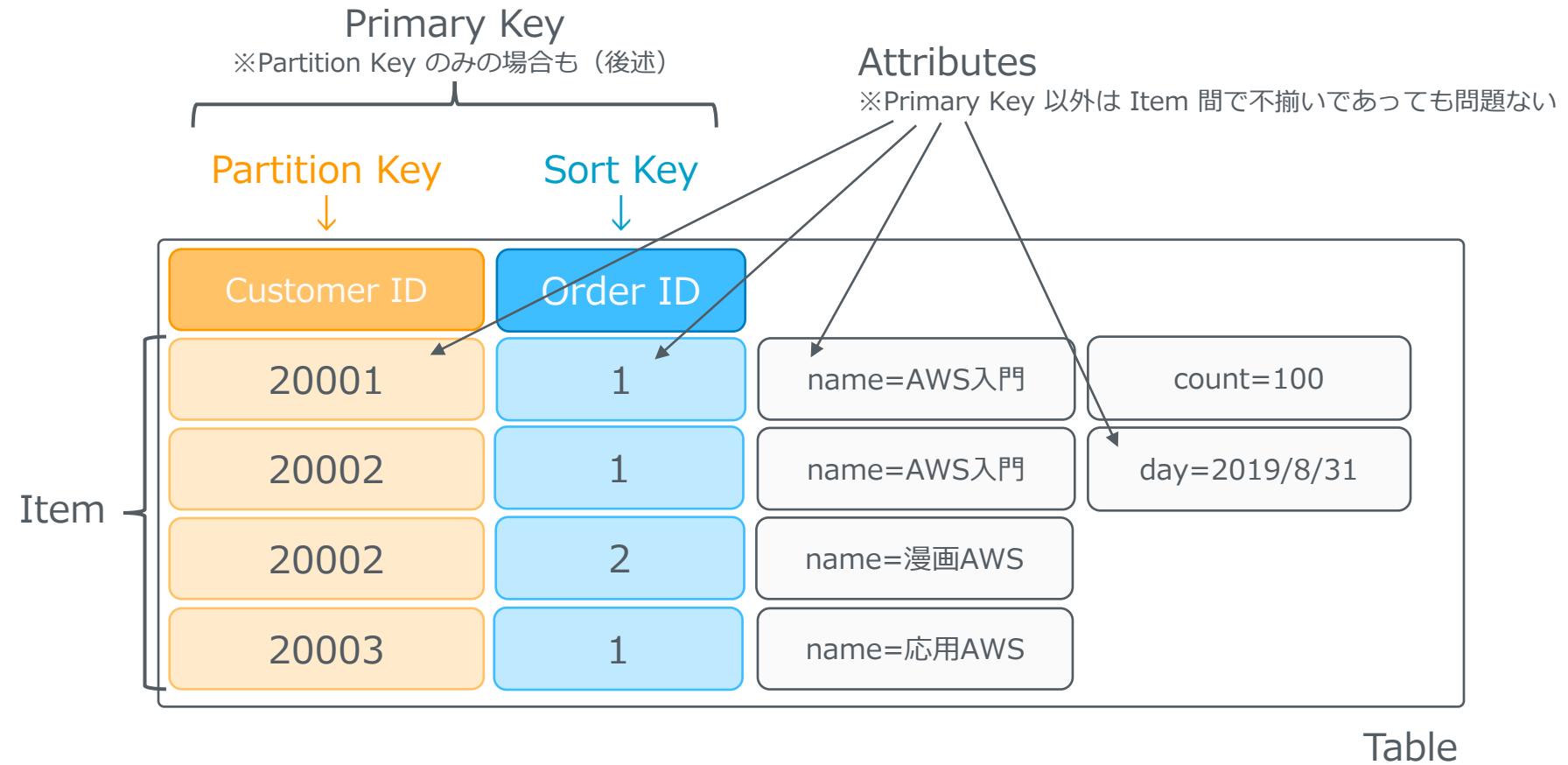


設定した  
Read キャパシティユニット  
Write キャパシティユニット  
(無料枠あり)



ストレージ利用料 ( + オプション機能料金 )

# Amazon DynamoDB の用語



# Amazon DynamoDB におけるテーブル設計①

- DynamoDB では2種類のプライマリーキーをサポートする
  - Partition Key
  - Partition Key & Sort Key

## ◆ Partition Key

| Product ID |              |               |
|------------|--------------|---------------|
| 1001       | name=AWS入門   | price=3,240   |
| 1002       | name=応用AWS   |               |
| 1003       | day=20191224 | publisher=x出版 |
| 1004       | name=漫画AWS   | price=980     |

↑ Partition Key  
(同じPartition Key の Item は登録できない)

## ◆ Partition Key & Sort Key

| Customer ID | Order ID |            |               |
|-------------|----------|------------|---------------|
| 20001       | 1        | name=AWS入門 |               |
| 20002       | 1        | name=AWS入門 | count=100     |
| 20002       | 2        | name=漫画AWS |               |
| 20003       | 1        | name=応用AWS | day=2019/8/31 |

↑ Partition Key      ↑ Sort Key  
(Partition Key と Sort Key がともに同じ Item は登録できない)

# Amazon DynamoDB におけるテーブル設計②

- DynamoDB では2種類の Secondary Index を利用することができる
    - Local Secondary Index (LSI)
      - Sort Key 以外に絞り込み検索を行う Key を持つことができる
      - Partition Key はベーステーブルと同じ / Sort Key が異なる
    - Global Secondary Index (GSI)
      - Partition Key 属性の代わりとなる、Partition Key をまたいで検索を行うためのインデックス

## ◆ Local Secondary Index の例

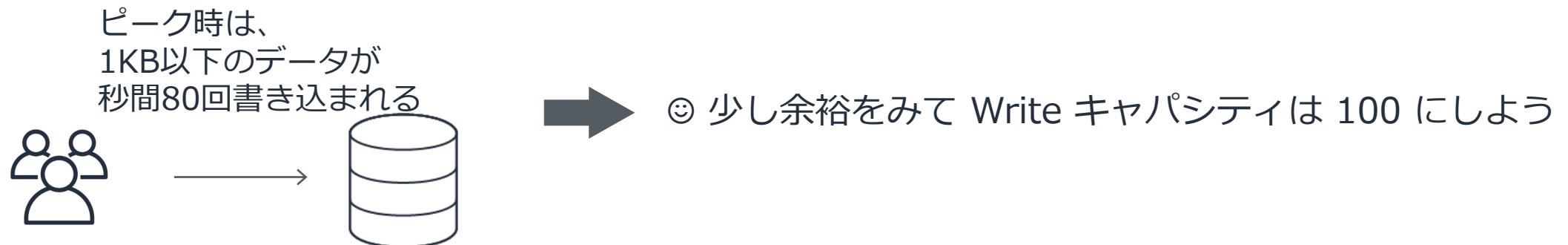
| Customer ID | Order ID | (Key以外のスキーマはItemごと) |              |
|-------------|----------|---------------------|--------------|
| 20001       | 1        | Book Name: AWS入門    | Price: 2,160 |
| 20001       | 2        | Book Name: 応用AWS    | Price: 3,240 |

「ある顧客のあるオーダー」以外に、「ある顧客の幾らのオーダー」も検索可能になる

# Amazon DynamoDB におけるテーブル設計③

- キャパシティユニットの考え方
  - Read: 1ユニットにつき、最大4KBのデータを1秒に1回読み込み可能  
(強い一貫性を持たない読み込み設定であれば、1秒あたり2回)
  - Write: 1ユニットにつき、最大1KBのデータを1秒に1回書き込み可能

## ◆考え方の例



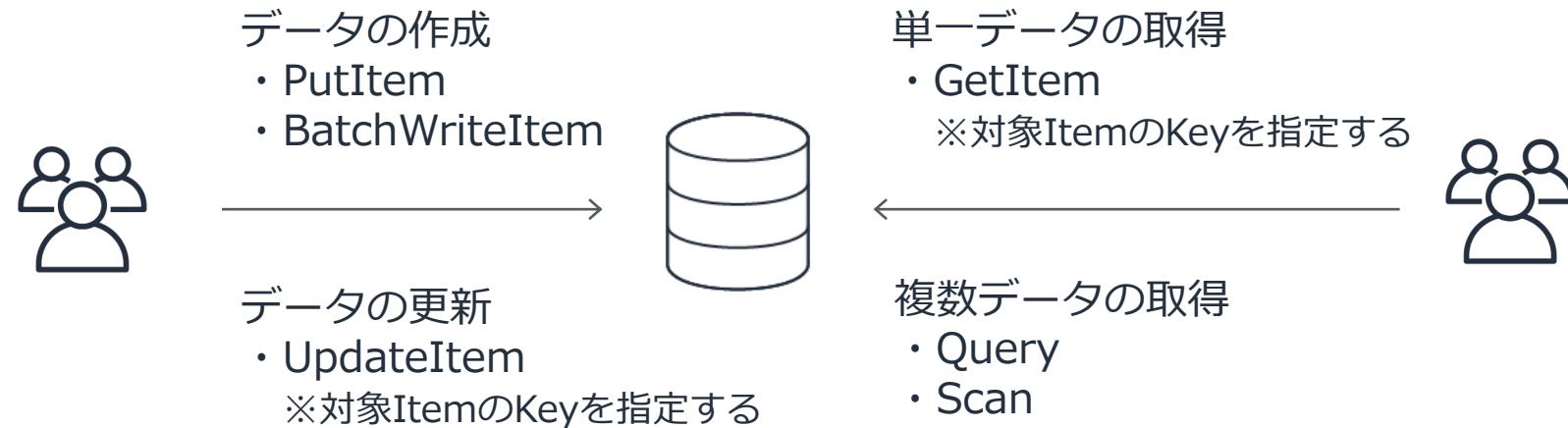
参考) 結果整合性のある読み込み / 強力な整合性のある読み込み

[https://docs.aws.amazon.com/ja\\_jp/amazondynamodb/latest/developerguide/HowItWorks.ReadConsistency.html](https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/HowItWorks.ReadConsistency.html)

# Amazon DynamoDB におけるデータの操作方法

- HTTPベースのAPIで操作を行う

## ◆API の例



参考) DynamoDB API

[https://docs.aws.amazon.com/ja\\_jp/amazondynamodb/latest/developerguide/HowItWorks.API.html](https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/HowItWorks.API.html)

参考) クエリとスキャンデータのベストプラクティス

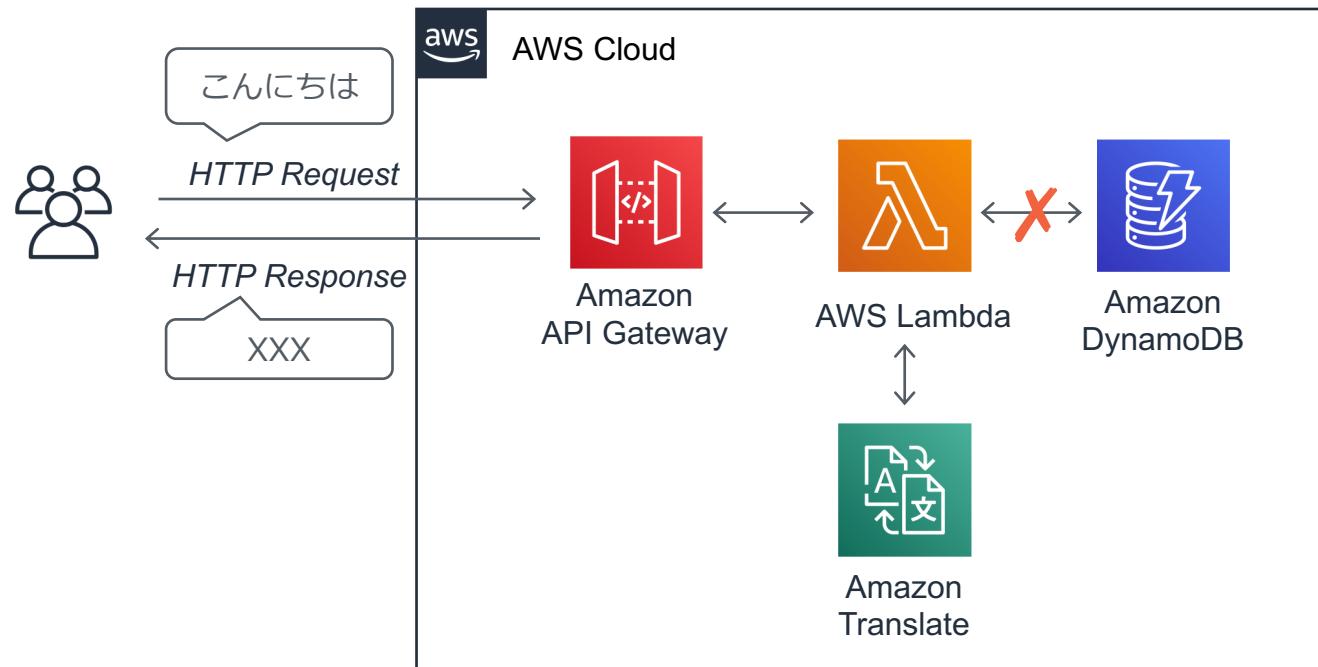
[https://docs.aws.amazon.com/ja\\_jp/amazondynamodb/latest/developerguide/bp-query-scan.html](https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/bp-query-scan.html)

## 4 – 2) Amazon DynamoDB ハンズオン①

### ～ テーブルを作つてみる ～

# DynamoDB ハンズオン① テーブルを作ってみる

- DynamoDB のテーブルを作る
  - Partition Key として timestamp を指定する
  - テーブル設定では、下記の設定を行う
    - Read CU: 1
    - Write CU: 1
  - テーブル完成まで少し待つ
  - データを手動で入れてみる



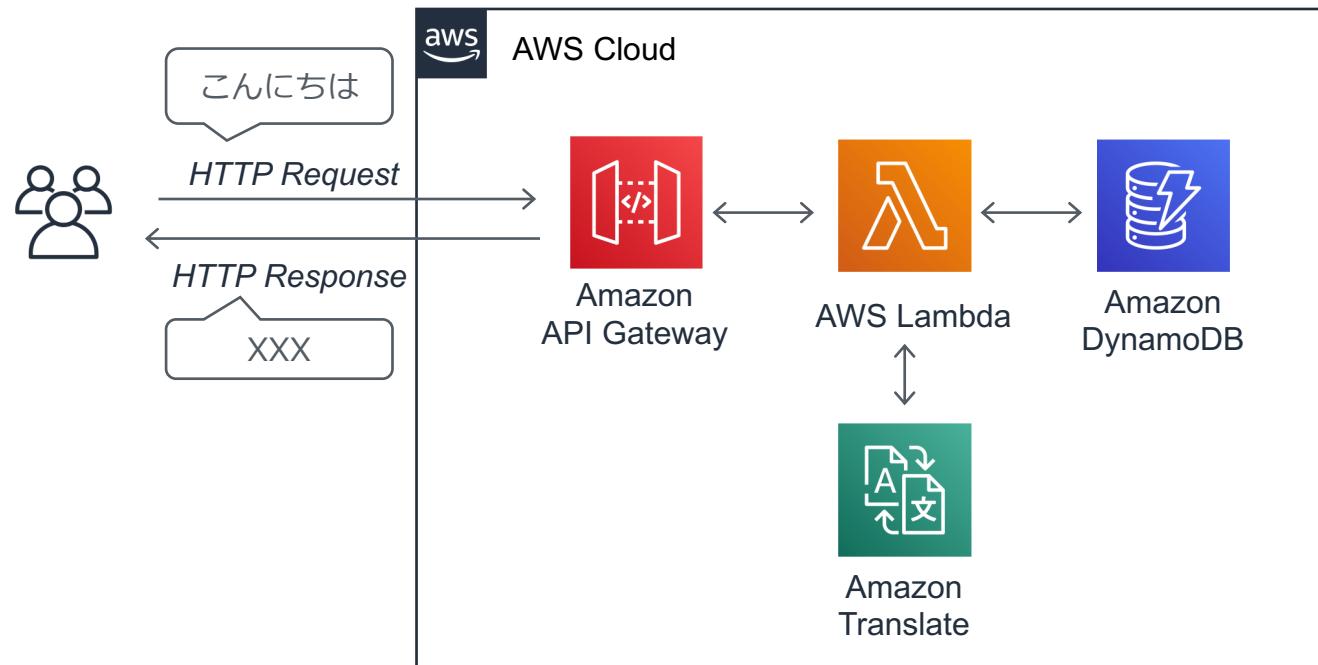
## 4 – 3) Amazon DynamoDB ハンズオン②

～ API Gateway と Lambda と DynamoDB を組み合わせる ～

# DynamoDB ハンズオン② API GW + Lambda に DynamoDB を組み合わせる

- 翻訳履歴を DynamoDB にストアするように修正する

- Python SDK のドキュメントを見ながら  
DynamoDB にストアするように Lambda 関数を修正する
- IAM ロール設定を修正する
- API を叩いてみる
  - 結果は返ってくるか？
  - DynamoDB にストアされるか？



# 落ち穂拾い & まとめ & 今後の Learning Paths

# AWS Lambda の落ち穂拾い① ~グローバルスコープへの記述~

- グローバルスコープに記述すると、  
コンテナが再利用された際 (=ウォームスタート) に再利用される
- ローカルスコープの場合は、再利用されない

```
 5 import datetime
 6
 7 translate = boto3.client('translate')          グローバルスコープ
 8
 9 dynamodb_translate_history_tbl = boto3.resource('dynamodb').Table('translate-history')
10
11 def lambda_handler(event, context):
12
13     input_text = event['queryStringParameters']['input_text']          ローカルスコープ
14
15     response = translate.translate_text(
16         Text=input_text
```

# AWS Lambda の落ち穂拾い② ~例外処理~

- ・今回の作成したソースコードでは、例外処理をしておりません
- ・実際の開発時は、例外ハンドリングを行うようにしてください

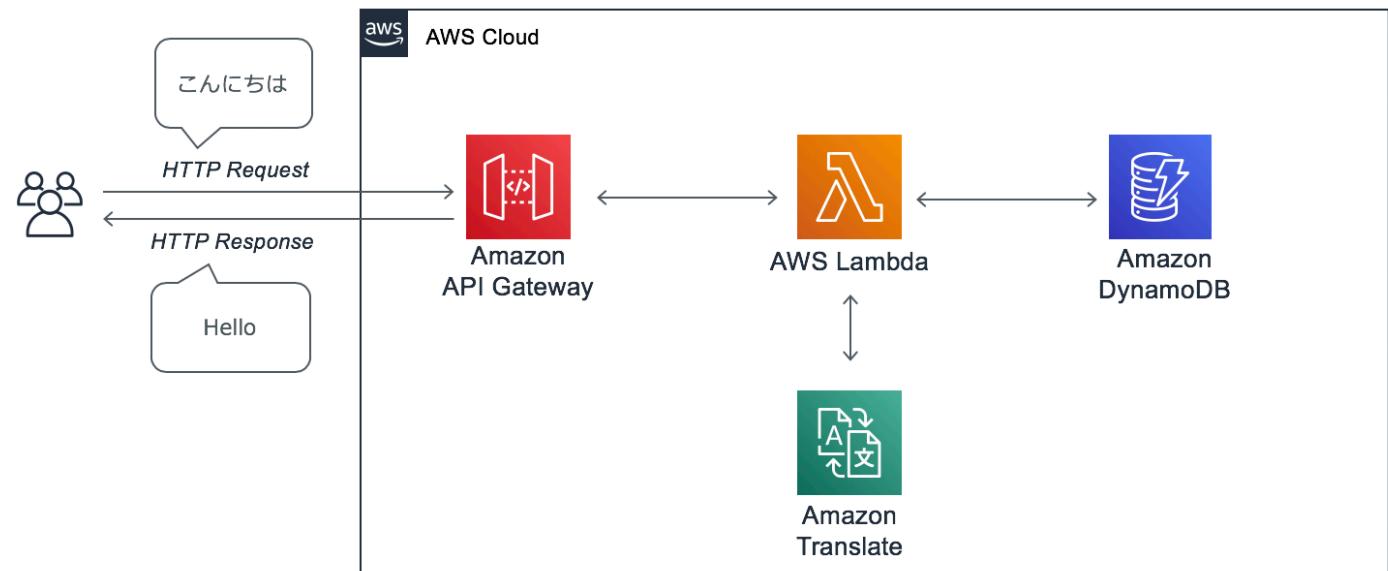
## ◆例外処理の例

```
11     input_text = event['queryStringParameters']['input_text']  
12  
13     try:  
14         response = translate.translate_text(  
15             Text=input_text,  
16             SourceLanguageCode="ja",  
17             TargetLanguageCode="en"  
18         )  
19  
20     except Exception as e:  
21         logging.error(e.response['Error']['Message'])  
22         raise Exception("[ErrorMessage]: " + str(e))  
23  
24     output_text = response.get('TranslatedText')
```



# 本コースのまとめ

- ・サーバーレスの特徴や利点について学んでいただきました
- ・AWS Lambda, Amazon API Gateway, Amazon DynamoDBといった AWS サービスの基本知識を学んでいただきました
- ・実際に手を動かし、サーバーレスアーキテクチャで Web API を作成していただきました



# 参考資料(1)

- BlackBelt: AWS Lambda Part1
  - [https://d1.awsstatic.com/webinars/jp/pdf/services/20190402\\_AWSBlackbelt\\_AWSLambda%20Part1%262.pdf](https://d1.awsstatic.com/webinars/jp/pdf/services/20190402_AWSBlackbelt_AWSLambda%20Part1%262.pdf)
- AWS Lambda 料金
  - <https://aws.amazon.com/jp/lambda/pricing/>
- Python の AWS Lambda 関数ログ作成
  - [https://docs.aws.amazon.com/ja\\_jp/lambda/latest/dg/python-logging.html](https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/python-logging.html)
- AWS SDK for Python (Boto3)
  - <https://aws.amazon.com/jp/sdk-for-python/>
- BlackBelt: Amazon API Gateway
  - [https://d1.awsstatic.com/webinars/jp/pdf/services/20190514\\_AWS-Blackbelt\\_APIGateway.pdf](https://d1.awsstatic.com/webinars/jp/pdf/services/20190514_AWS-Blackbelt_APIGateway.pdf)

# 参考資料(2)

- BlackBelt: Amazon DynamoDB
  - [https://d1.awsstatic.com/webinars/jp/pdf/services/20170809\\_AWS-BlackBelt-DynamoDB.pdf](https://d1.awsstatic.com/webinars/jp/pdf/services/20170809_AWS-BlackBelt-DynamoDB.pdf)
- Amazon DynamoDB 料金
  - <https://aws.amazon.com/jp/dynamodb/pricing/>
- Amazon DynamoDB 結果整合性のある読み込み / 強力な整合性のある読み込み
  - [https://docs.aws.amazon.com/ja\\_jp/amazondynamodb/latest/developerguide/HowItWorks.ReadConsistency.html](https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/HowItWorks.ReadConsistency.html)
- Amazon DynamoDB API
  - [https://docs.aws.amazon.com/ja\\_jp/amazondynamodb/latest/developerguide/HowItWorks.API.html](https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/HowItWorks.API.html)
- Amazon DynamoDB クエリとスキヤンデータのベストプラクティス
  - [https://docs.aws.amazon.com/ja\\_jp/amazondynamodb/latest/developerguide/bp-query-scan.html](https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/bp-query-scan.html)

# 参考資料(3)

- ホワイトペーパー: サーバーレスアーキテクチャでのアプリケーションの構築
  - <https://aws.amazon.com/jp/lambda/serverless-architectures-learn-more/>
- ホワイトペーパー: AWS Lambda を使用した サーバーレスアーキテクチャ
  - [https://d1.awsstatic.com/whitepapers/ja\\_JP/serverless-architectures-with-aws-lambda.pdf?did=wp\\_card&trk=wp\\_card](https://d1.awsstatic.com/whitepapers/ja_JP/serverless-architectures-with-aws-lambda.pdf?did=wp_card&trk=wp_card)
- 形で考えるサーバーレス設計
  - <https://aws.amazon.com/jp/serverless/patterns/serverless-pattern/>

# Learning Paths / Next Action

## 深く学ぶ

- サーバーレスアーキテクチャで何かプロダクトを作る
  - 例：チャットツールのボット、SNSのボット、オリジナル Web API などなど
- その上で、実際に運用もしてみる
  - モニタリングはどうする？構築を自動化するなら？★
- 今回のハンズオン以外の構成も調べてみる
  - フロントエンド部分やバッチ処理はどうする？★

★については次回以降の講座にご期待ください！🎉

## 広く学ぶ

- トレーニングコースの受講
- 資格取得のための学習
  - 様々なAWSサービス群を知っていただき、よいアーキテクチャ設計ができるようになる
- 他の AWS Hands-on Beginners
  - ELB-EC2-RDS の構成（近日公開予定）
  - ネットワーク入門（近日公開予定）
  - などなど

サイト内のアンケートリンクから  
フィードバックをお願いします！🙏

