Alice_Maker の
インフラストラクチャー

Amazon API Gateway

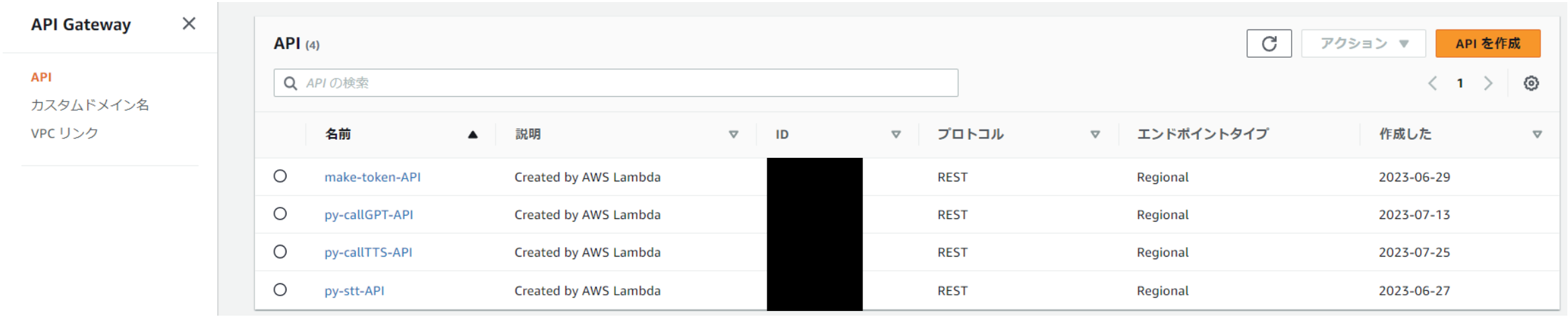
以下の4つのAPI Gatewayを使用しています

・make-token-API
→playfabからのtoken生成用エンドポイント

・py-callGPT-API
→openAI GPTシリーズを呼ぶエンドポイント

・py-callSTT-API
→openAI Whisperを呼ぶエンドポイント

・py-stt-API
→rinna株式会社のkomeotionを呼ぶエンドポイント



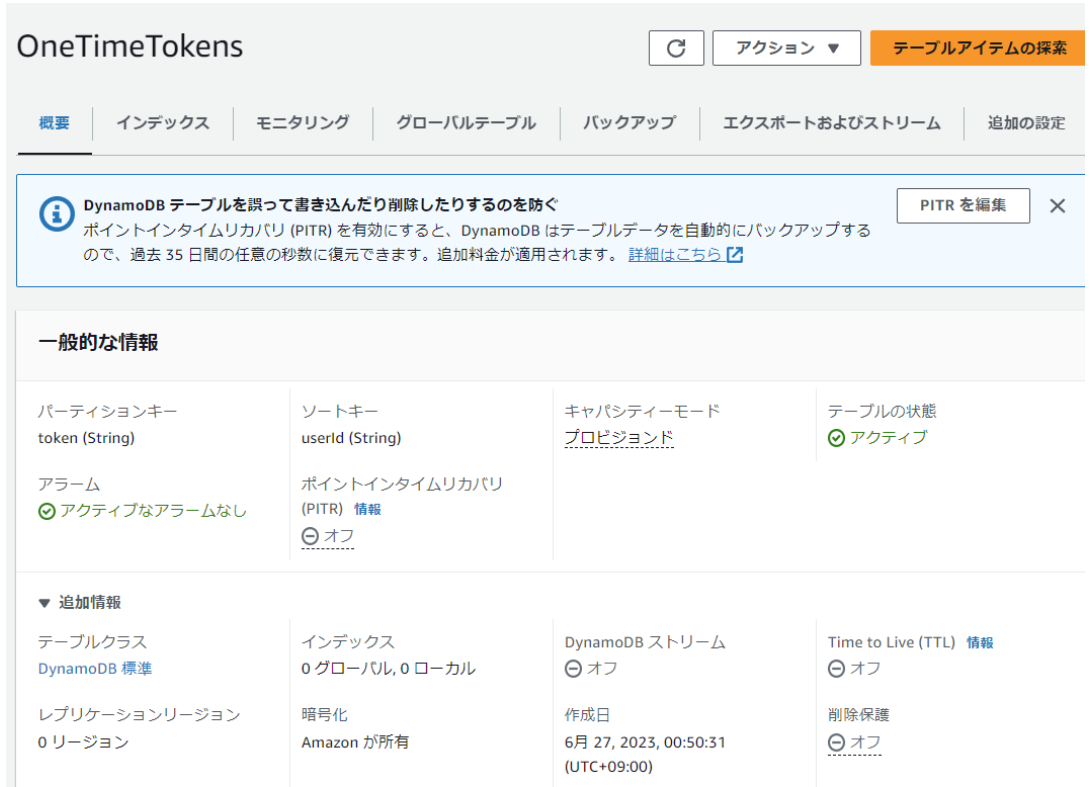
Amazon API Gateway

API gateway側での検証について。通常gateway→lambda関数の順で呼び出されますが、必要なヘッダーが無いとlambdaの起動自体が無駄です。よってgatewayでヘッダーの検証を行います。以下はヘッダーが無い場合のレスポンスです。

{　"message": "Missing required request parameters: [One-Time-Token, Model-Name, User-Id]"}

Amazon DynamoDB

DyanamoDBはミリ秒単位で書き込みが可能なNoSQL型のDBの為採用しました。
パーティーションキー(primary key)がtoken、ソートキーがuseridとなっています。



AWS Lambda

実行時にサーバーを起動させ、関数の処理が終わると終了するサービス。サーバーレスなアプリを作ることができる。複数の言語が使えるがメジャーなPython 3.1を選択。

注意
ソースコード作成にあたり、Github copilot、ChatGPT、Stack Overflow、Qiita、Zennを使用、参考にしています。ご注意ください。

make-token.py - メモ帳

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
import json
import boto3
import random
import string
from botocore.exceptions import BotoCoreError, ClientError

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('OneTimeTokens')

def lambda_handler(event, context):
    try:
        # POST request bodyからuserIdを取得
        user_id = json.loads(event['body'])['userId']
        token = ''.join(random.choice(string.ascii_letters + string.digits) for _ in range(16))

        response = table.put_item(
           Item={
                'token': token,
                'userId': user_id
            }
        )

        if response['ResponseMetadata']['HTTPStatusCode'] == 200:
            return {
                'statusCode': 200,
                'body': json.dumps({'token': token}),
                'headers': {
                    'Content-Type': 'application/json',
                    'Access-Control-Allow-Origin': '*'
                },
            }
        else:
            raise ValueError("DynamoDB write failed")

    except KeyError as e:
        return {
            'statusCode': 400,
            'body': 'userId parameter is missing',
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
        }
    except (BotoCoreError, ClientError) as e:
        return {
            'statusCode': 500,
            'body': 'Failed to write to DynamoDB',
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
        }
    except Exception as e:
        return {
            'statusCode': 500,
            'body': 'An error occurred: {}'.format(e),
            'headers': {
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*'
            },
        }
```

AWS Lambda

*py-callGPT.py - メモ帳

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
mport os
mport http.client
mport json
mport boto3

‡ DynamoDBのクライアントを作成
ynamodb = boto3.client('dynamodb')

ef lambda_handler(event, context):
    # API Gateway 経由のイベントの場合、body は文字列として提供されるため、json.loads() で変換が必要
    event_body = json.loads(event['body'])

    # ヘッダーから一時的なトークン、ユーザーID、モデル名を取得
    one_time_token = event['headers']['One-Time-Token']
    user_id = event['headers']['User-Id']
    model_name = event['headers']['Model-Name']

    # モデル名に応じて、OpenAIのモデル名を選択
    if model_name == 'GPT4':
        openai_model_name = 'gpt-4-0613'
    elif model_name == 'GPT3':
        openai_model_name = 'gpt-3.5-turbo-0613'
    else:
        return {
            'statusCode': 400,
            'body': json.dumps('Invalid model name')
        }

    # クライアントから受け取ったペイロードをデコード
    messages = event_body['messages']

    # DynamoDBからトークンを検索
    response = dynamodb.get_item(TableName='OneTimeTokens', Key={'token': {'S': one_time_token}, 'userId': {'S': user_id}})

    # トークンが存在しない場合はエラーメッセージを返す
    if 'Item' not in response:
        return {
            'statusCode': 403,
            'body': json.dumps('Invalid token')
        }

    # トークンが存在する場合は削除する
    dynamodb.delete_item(TableName='OneTimeTokens', Key={'token': {'S': one_time_token}, 'userId': {'S': user_id}})

    # OpenAIのAPIを叩く
    conn = http.client.HTTPSConnection("api.openai.com")
    headers = {
        'Authorization': 'Bearer sk-MYAPIKEY',
        'Content-Type': 'application/json'
    }
    data = {
        "model": openai_model_name,
        "messages": messages
    }
    conn.request("POST", "/v1/chat/completions", json.dumps(data), headers)
    response = conn.getresponse()
    return {
        'statusCode': response.status,
        'body': response.read().decode()
    }
```

*py-callTTS.py - メモ帳

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
import os
import http.client
import json
import boto3

# DynamoDBのクライアントを作成
dynamodb = boto3.client('dynamodb')

def lambda_handler(event, context):
    # API Gateway 経由のイベントの場合、body は文字列として提供されるため、json.loads() で変換が必要
    event_body = json.loads(event['body'])

    # ヘッダーから一時的なトークン、ユーザーIDを取得
    one_time_token = event['headers']['One-Time-Token']
    user_id = event['headers']['User-Id']

    # クライアントから受け取ったメッセージと音声特性をデコード
    message_text = event_body['message']
    speaker_x = event_body['speaker_x']
    speaker_y = event_body['speaker_y']
    style = event_body['style']

    # DynamoDBからトークンを検索
    response = dynamodb.get_item(TableName='OneTimeTokens', Key={'token': {'S': one_time_token}, 'userId': {'S': user_id}})

    # トークンが存在しない場合はエラーメッセージを返す
    if 'Item' not in response:
        return {
            'statusCode': 403,
            'body': json.dumps({'response': 'Invalid token'})
        }

    # トークンが存在する場合は削除する
    dynamodb.delete_item(TableName='OneTimeTokens', Key={'token': {'S': one_time_token}, 'userId': {'S': user_id}})

    # Rinna APIを叩く
    conn = http.client.HTTPSConnection("api.rinna.co.jp")
    headers = {
        'Content-Type': 'application/json',
        'Ocp-Apim-Subscription-Key': 'MYAPIKEY'
    }
    data = {
        "text": message_text,
        "speaker_x": speaker_x,
        "speaker_y": speaker_y,
        "style": style,
        "seed": 124,
        "speed": 1.0,
        "volume": 3.8,
        "output_format": "wav",
        "output_bitrate": "64k"
    }
    conn.request("POST", "/koeiromap/v1.0/infer", json.dumps(data), headers)
    response = conn.getresponse()
    response_body = json.loads(response.read().decode())

    # 返ってきた音声データから "data:audio/x-wav;base64," を取り除き、そのままクライアントに送り返す
    audio_data = response_body['audio'].replace("data:audio/x-wav;base64,", "")

    return {
        'statusCode': response.status,
        'body': json.dumps({'response': audio_data})
    }
```

AWS Lambda

*py-stt.py - メモ帳

ファイル(F)　編集(E)　書式(O)　表示(V)　ヘルプ(H)

```
mport base64
mport boto3
mport json
mport mimetypes
mport http.client
mport os
mport tempfile

t DynamoDBとTranscribeのクライアントを作成
ynamodb = boto3.client('dynamodb')

ef lambda_handler(event, context):
    # ヘッダーから一時的なトークンとユーザーIDを取得
    one_time_token = event['headers']['One-Time-Token']
    user_id = event['headers']['User-Id']

    # クライアントから受け取ったペイロードをデコード
    body = json.loads(event['body'])

    # base64エンコードされたwavデータを取得
    base64_wav_data = body['base64_wav_data']

    # DynamoDBからトークンを検索
    response = dynamodb.get_item(TableName='OneTimeTokens', Key={'token': {'S': one_time_token}, 'userId': {'S': user_id}})

    # トークンが存在しない場合はエラーメッセージを返す
    if 'Item' not in response:
        return {
            'statusCode': 403,
            'body': json.dumps('Invalid token')
        }

    # トークンが存在する場合は削除する
    dynamodb.delete_item(TableName='OneTimeTokens', Key={'token': {'S': one_time_token}, 'userId': {'S': user_id}})

    # base64エンコードされたwavデータをデコード
    wav_data = base64.b64decode(base64_wav_data)

    # デコードされたデータを一時ファイルに書き込む
    with tempfile.NamedTemporaryFile(delete=False) as tmp:
        tmp.write(wav_data)
        tmp_path = tmp.name

    # OpenAI APIにリクエストを送る
    conn = http.client.HTTPSConnection("api.openai.com")

    # Define the boundary value (a string that won't appear in the file contents)
    boundary = '--------------------------905191404154484336597275426 6'

    # Define the headers
    headers = {
        'Content-Type': 'multipart/form-data; boundary={}'.format(boundary),
        'Authorization': 'Bearer sk-MYAPIKEY',
    }

    # Define the parts of the multipart message
    parts = [
        '--' + boundary,
        'Content-Disposition: form-data; name="model"',
        '',
        'whisper-1',
        '--' + boundary,
        'Content-Disposition: form-data; name="file"; filename="file.wav"',
        'Content-Type: {}'.format(mimetypes.guess_type(tmp_path)[0] or 'application/octet-stream'),
        '',
        wav_data,
        '--' + boundary + '--',
        '',
    ]

    # Combine the parts into a single string
    data = b'¥r¥n'.join(part.encode('utf-8') if isinstance(part, str) else part for part in parts)

    # Send the POST request
    conn.request("POST", "/v1/audio/transcriptions", body=data, headers=headers)

    # Get the response
    response = conn.getresponse()

    # リクエストが成功したら一時ファイルを削除する
    os.unlink(tmp_path)

    # HTTP Responseをそのまま返す
    return {
        'statusCode': response.status,
        'body': response.read()
    }
```

Azure

Azure PlayFab

公式より引用

PlayFab は、マネージド ゲーム サービス、リアルタイム分析、および LiveOps を備えたライブ ゲーム向けの完全なバックエンド プラットフォームです。 これらの機能は、コストを削減しながら収益を上げ、プレイヤーのエンゲージメントを高めるのに役立ちます。

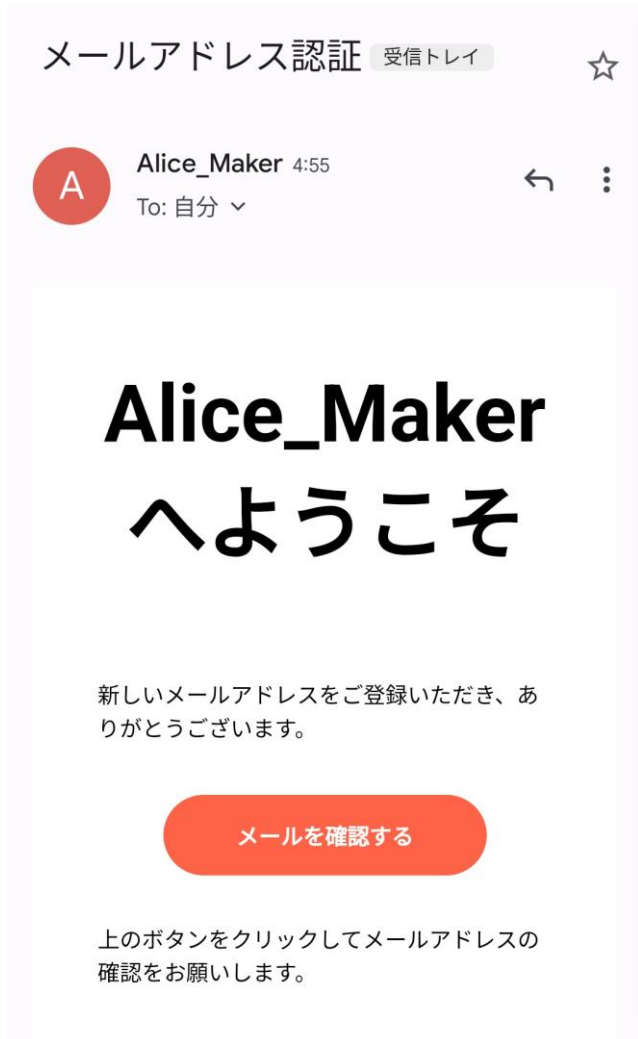
Cloudscriptにはjavascriptを使用

注意
ソースコード作成にあたり、Github copilot、ChatGPT、Stack Overflow、Qiita、Zenn、PlayFab Communityを使用、参考にしています。ご注意ください。

Maketoken 関数

```javascript
handlers.maketoken = function (args, context) {
    var playerData = server.GetPlayerProfile({
        "PlayFabId": currentPlayerId,
        "ProfileConstraints": { "ShowContactEmailAddresses": true }
    });

    var checkemail = "Error";

    if (playerData.Error == null) {
        var contactEmailAdresses = playerData.PlayerProfile.ContactEmailAddresses;
        if ((contactEmailAdresses != null) && (contactEmailAdresses.length > 0)) {
            var emailVerificationStatus = contactEmailAdresses[0].VerificationStatus;
            if (emailVerificationStatus != null) {
                if (emailVerificationStatus == "Confirmed") {
                    checkemail = "Email is verified.";
                    var apiGatewayResponse = callAwsApiGateway({ "userId": currentPlayerId });
                    return { playerId: currentPlayerId, emailCheck: checkemail, apiGatewayResponse: apiGatewayResponse };
                } else {
                    checkemail = "Current contact email is not verified.";
                }
            } else {
                checkemail = "Current email is not verified.";
            }
        } else {
            checkemail = "No valid email found.";
        }
    }
    return { playerId: currentPlayerId, emailCheck: checkemail };
};

function callAwsApiGateway(payload) {
    var headers = {
        "Content-Type": "application/json",
    };

    var url = "AWS API Gateway Endpoint here";
    var content = JSON.stringify(payload);
    var httpMethod = "post";
    var contentType = "application/json";

    var response = http.request(url, httpMethod, content, contentType, headers);

    return response;
}
```

Azure




```
<!DOCTYPE html>
<html>
<head>
    <style>
        body {
            font-family: 'Hiragino Kaku Gothic ProN', Meiryo, sans-serif;
            color: #444;
            background-color: #F7F7F7;
            margin: 0;
            padding: 0;
        }

        .email-container {
            max-width: 600px;
            margin: 0 auto;
            padding: 20px;
            background-color: #FFFFFF;
            box-shadow: 0px 0px 20px 0px rgba(0,0,0,0.1);
        }

        .email-header {
            text-align: center;
            font-size: 1.8em;
            margin-bottom: 20px;
        }

        .email-body {
            padding: 20px;
            line-height: 1.5;
        }

        .confirm-btn {
            display: block;
            width: 200px;
            height: 50px;
            margin: 30px auto;
            background-color: #FF6347;
            color: #FFFFFF;
            text-align: center;
            line-height: 50px;
            border-radius: 25px;
            text-decoration: none;
            font-weight: bold;
        }

        .confirm-btn:hover {
            background-color: #FF4500;
        }
    </style>
</head>
<body>
    <div class="email-container">
        <div class="email-header">
            <h1>Alice_Makerへようこそ</h1>
        </div>
        <div class="email-body">
            <p>新しいメールアドレスをご登録いただき、ありがとうございます。</p>
            <a href="$ConfirmationUrl$" class="confirm-btn">メールを確認する</a>
            <p>上のボタンをクリックしてメールアドレスの確認をお願いします。</p>
        </div>
    </div>
</body>
</html>
```