

そもそもクラウドとか Azure って何？

さて、冒頭で「今回は Azure についてです！」とは言ったものの、当初の私は Azure 以前にクラウドがどういうものなのかも明確には理解していない状況でした。そこで、まずはクラウド・Azure ってなんなのかというところから学んでいくことに。

クラウドとは、「クラウドコンピューティング」の略称で、ユーザーが自分の端末に持っていないアプリやサービスをインターネット等のネットワークを経由して必要な時に利用できる形態のことを指します。

上記の通り、自分の端末にソフトやアプリをインストールしなくても必要になった時に随時利用できるのも、ハードウェアの物理的スペースやシステム構築の時間などの導入コストを削減できるというメリットがあります。また、インターネットに繋がってさえいれば、どの端末からでも同じデータやサービスを利用できるというのも大きな特徴です。

クラウドサービスは、ソフトウェアを提供する SaaS（Software as a Service）、開発環境（プラットフォーム）を提供する PaaS（Platform as a Service）、インフラ機能を提供する HaaS（Hardware as a Service)/IaaS（Infrastructure as a Service)の3種類に分類されます。

クラウドサービスを利用する際には「ポータル」と呼ばれるサイトにブラウザでアクセスし、このポータル上で利用するサービス種別（開発環境含む）や課金情報などを管理します。

そんなクラウドサービスの一つが Azure（アジュール）。正式には Microsoft Azure と言い、その名の通り Microsoft社が提供しています。Azure を利用するにはアカウント登録が必要となります。

では、大まかにクラウドと Azure について理解したところで、本題の Azure 上で実際にプログラムを作っていきます。

作成するアプリの前提

今回は、Azure SQL Database に登録されているデータをログに出力するプログラムを Node.js で実装します。実行した際に、DB から取得したデータがログに出力できれば OK です。

なお、前述の通り Azure で開発を行うにはあらかじめ Azure アカウントが必要です。また、今回

実装・構築する SQL Server や関数アプリなどのリソースは新たに作成するリソースグループ内でまとめて管理するようにします。

※この記事は2020年3月時点の情報を基に作成しており、今後のバージョンアップ等により動作しなくなる場合があります。

関数アプリの作成

まずは関数アプリを作成します。今回は Microsoft が提供しているイベントドリブン型のサーバーレスコンピューティングプラットフォームである [Azure Functions](#) というクラウドサービスの機能を使用して関数アプリを作成します。Azure のポータルにサインイン後、メニューから「関数アプリ」→「追加」の順にクリックします。

Microsoft Azure

リソース、サービス、ドキュメントの検索 (G+/)

ホーム > 関数アプリ > 関数アプリ

関数アプリ

基本 ホスト中 監視 タグ 確認および作成

関数アプリを作成すると、関数を論理ユニットとしてグループ化できるため、リソースの管理、デプロイ、共有が容易になります。関数を使用すると、最初に VM を作成したり、Web アプリケーションを公開したりすることなく、サーバーレス環境でコードを実行できます。

プロジェクトの詳細

デプロイされているリソースとコストを管理するサブスクリプションを選択します。フォルダーのようなリソース グループを使用して、すべてのリソースを整理し、管理します。

サブスクリプション * ⓘ

リソース グループ * ⓘ

[新規作成](#)

インスタンスの詳細

関数アプリ名 * .azurewebsites.net

公開 * ☒ コード ☐ Docker コンテナ

ランタイム スタック *

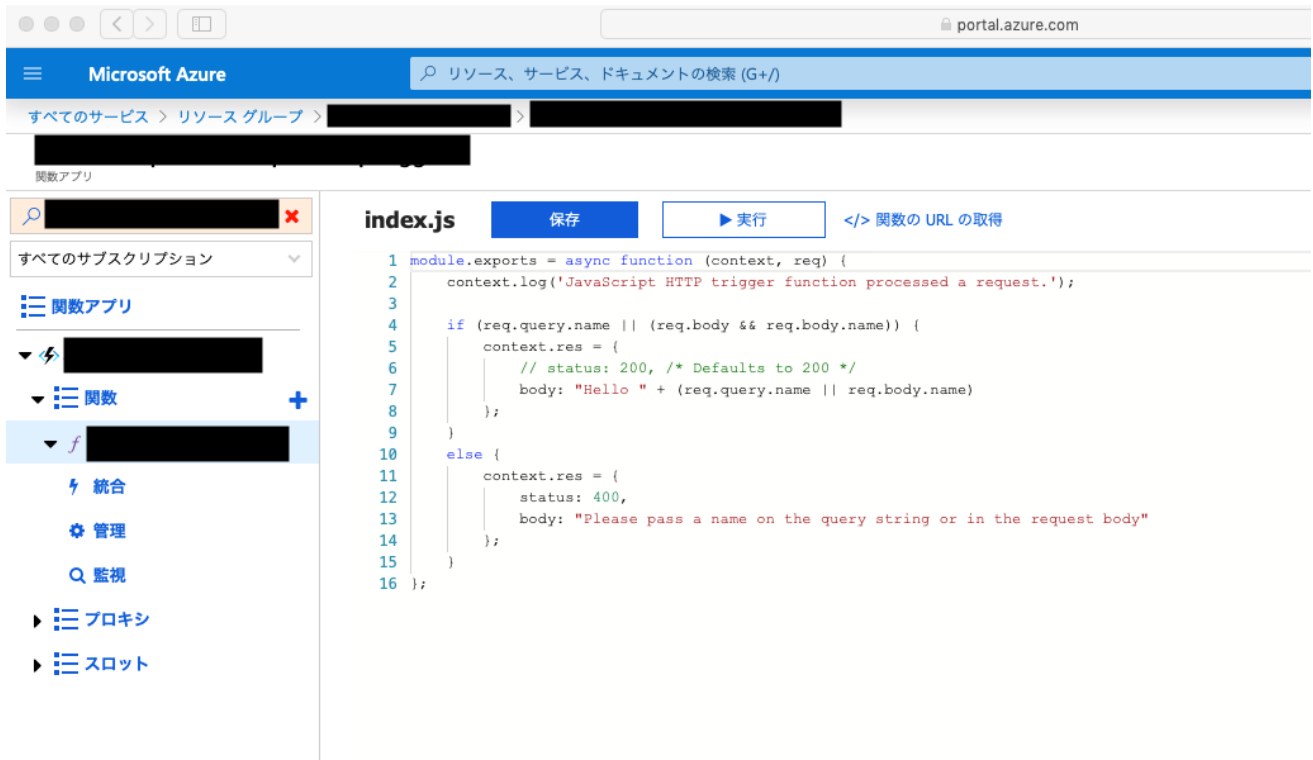
バージョン *

地域 *

[確認および作成](#) < 前へ 次: ホスト中 >

アプリ名やリソースグループなどの必要項目を入力して、「確認および作成」をクリックします。そうすると関数アプリが新規追加されるので、アプリのページに移動して関数名の右にある「+」をクリックして関数を追加します。

今回の関数は、実行トリガーを `HttpTrigger`、承認レベルを `Function` で作成しましょう。



試しに、自動で作成されたこのコードをここで一度実行してみましょう。「関数の URL の取得」をクリックして表示される URL をコピーし後ろに「`&name=testUser`」を追加してブラウザで実行すると、以下のような画面になります。



なお URL に注目すると「?code=」で始まる文字列が含まれています。これは先ほどの関数作成時に承認レベルを「function」に設定したことで、関数実行のための認証キーとして生成された文字列です。この文字列が1文字でも異なっていると関数を実行できません。

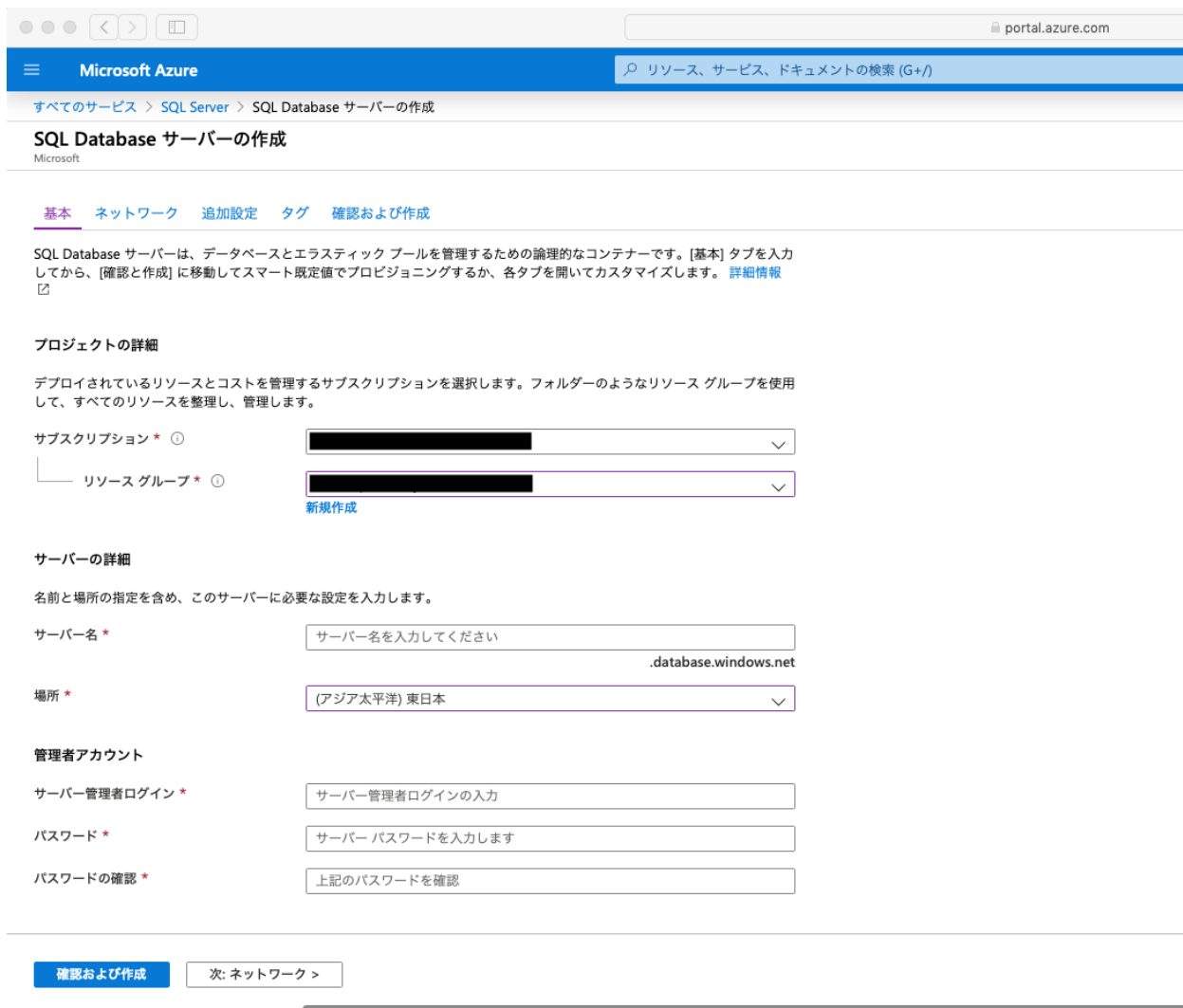
認証キーなしでの実行可能な関数としたい場合は、承認レベルを **anonymous** にすることで、認証なしでの実行が可能となります。ただし、URL から「code=」の文字列が消えているので、URL に追加する文字列を「&name=testUser」から「?name=testUser」に変更する必要があります。

SQL データベースの作成

続いて、関数のアクセス先となるデータベースを作成・設定します。今回は **Azure** が提供しているサービスを利用して新たに **SQL Server** のテーブルを作成しますが、事前に用意したテーブルやあらかじめ構築してある **SQL Server** のテーブルを利用しても問題ありません。

まずは **Azure** 上に **SQL Server** を構築します。

Azure ポータルのメニューから「すべてのサービス」→「SQL Server」→「追加」の順にクリックしていくと作成画面に遷移します。



Microsoft Azure

リソース、サービス、ドキュメントの検索 (G+/)

すべてのサービス > SQL Server > SQL Database サーバーの作成

SQL Database サーバーの作成

Microsoft

基本 ネットワーク 追加設定 タグ 確認および作成

SQL Database サーバーは、データベースとエラスティック プールを管理するための論理的なコンテナです。[基本] タブを入力してから、[確認と作成] に移動してスマート既定値でプロビジョニングするか、各タブを開いてカスタマイズします。 [詳細情報](#)

プロジェクトの詳細

デプロイされているリソースとコストを管理するサブスクリプションを選択します。フォルダーのようなリソース グループを使用して、すべてのリソースを整理し、管理します。

サブスクリプション * ①

リソース グループ * ① [新規作成](#)

サーバーの詳細

名前と場所の指定を含め、このサーバーに必要な設定を入力します。

サーバー名 * .database.windows.net

場所 * (アジア太平洋) 東日本

管理者アカウント

サーバー管理者ログイン *

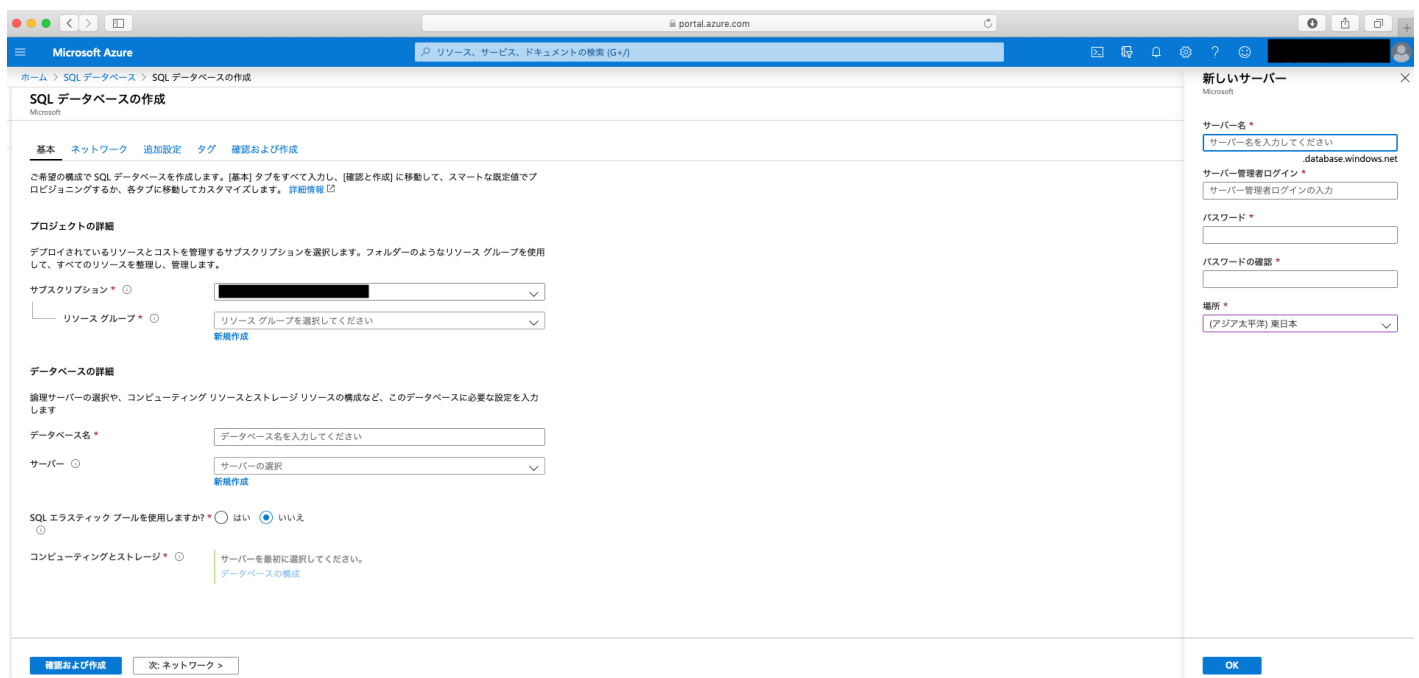
パスワード *

パスワードの確認 *

[確認および作成](#) [次: ネットワーク >](#)

必要事項を入力し、確認画面で「作成」をクリックで作成されます。

これで SQL Server が作成されましたが、データベースとテーブルの作成も必要です。再びメニューを開き「SQLデータベース」→「追加」の順にクリックしてデータベースの作成を行います。



Microsoft Azure

リソース、サービス、ドキュメントの検索 (G+/)

ホーム > SQL データベース > SQL データベースの作成

SQL データベースの作成

Microsoft

基本 ネットワーク 追加設定 タグ 確認および作成

ご希望の構成で SQL データベースを作成します。[基本] タブをすべて入力し、[確認と作成] に移動して、スマートな既定値でプロビジョニングするか、各タブに移動してカスタマイズします。 [詳細情報](#)

プロジェクトの詳細

デプロイされているリソースとコストを管理するサブスクリプションを選択します。フォルダーのようなリソース グループを使用して、すべてのリソースを整理し、管理します。

サブスクリプション * ①

リソース グループ * ① [新規作成](#)

データベースの詳細

論理サーバーの選択や、コンピューティング リソースとストレージ リソースの構成など、このデータベースに必要な設定を入力します

データベース名 *

サーバー ① [新規作成](#)

SQL エラスティック プールを使用しますか? * ☐ はい ☒ いいえ

コンピューティングとストレージ * ① [サーバーを最初に選択してください。](#)
[データベースの構成](#)

[確認および作成](#) [次: ネットワーク >](#)

新しいサーバー

Microsoft

サーバー名 * .database.windows.net

サーバー管理者ログイン *

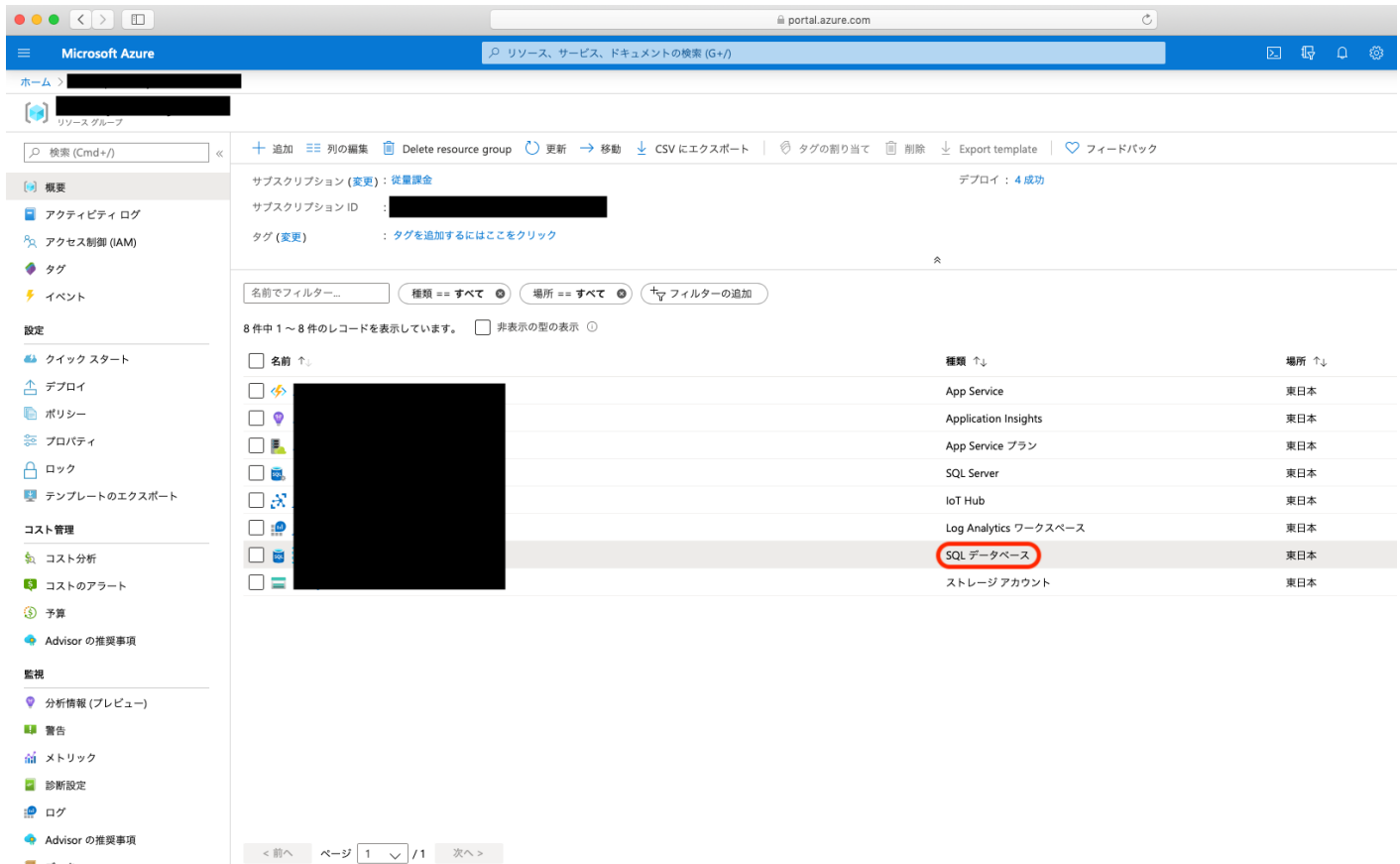
パスワード *

パスワードの確認 *

場所 * (アジア太平洋) 東日本

[OK](#)

SQL Serverの新規追加時と同様に、必要事項を入力後「作成」をクリックしてデータベースを作成します。リソースグループとSQL Serverは、どちらも作成したものを設定しましょう。



これで SQL Server 内にデータベースも作成できました。

あとはコマンド、もしくは SQL Server Management System などのツールを利用して、テーブル作成とデータの登録をしましょう。

今回はサンプルとして、ID・名前(name)・年齢(age)・所属(department)・入社日(hire_date)をカラムとして持つ従業員テーブルEmployeesを作成しました。データもいくつか追加しておきます。

DB アクセス処理の実装

Functions が作成できたので、ここから DB アクセスに関わる部分の実装を進めていきます。

Node.js と SQL Server は直接通信を行うことができないため、Node.js と SQL Server との通信を行う tedious というパッケージをインストールする必要があります。

【注釈】

ここから先の作業を行うにあたり、以下のドキュメントと Web ページに記載の内容を基にしています。ただし、本稿の内容に合わせてソースコードは一部変更しています。

- ・ [Node.js を使用してデータベースを照会する | Microsoft Docs](#)
- ・ [Azure Functions から SQLServerに接続する\(node.js編\) – Qiita](#)

まずはコマンドにて、作成した Functions のディレクトリ（index.js が含まれるディレクトリ）に移動した後、以下のコマンドを入力します。

```
npm init -y  
npm install tedious
```

このコマンドで tedious のインストールが行われ、Node.js のプログラムから SQL Server への接続が可能となります。

では、いよいよ SQLデータベースとの通信を行う部分の処理を実装していきます。関数アプリの index.js を開き、以下のコードを追加しましょう。

```
var Connection = require('tedious').Connection;  
var TedRequest = require('tedious').Request;  
var TYPES = require('tedious').TYPES;  
  
function scan_table(context, q_name){  
  
    //データベース接続情報  
    var config = {  
        server: 'サーバー名',  
        authentication: {  
            type: 'default',  
            options: {  
                userName: 'ユーザ名',  
                password: 'パスワード'  
            }  
        },  
        options: {  

```

```
        encrypt: true,
        database: 'データベース名',
    }
};

var query = "SELECT [ID],[name], [age], [department], [hire_date] "+
    "FROM [dbo].[Employees] "+
    "WHERE [name] = @WHO";

//CreateConnection
var connection = new Connection(config);

console.log("try connection...");

//接続
connection.on('connect', function(err) {
    if(err){
        //Error Handling
        console.log(err.stack);
    } else {
        console.log("connection start");
        // SQL実行
        results = executeStatement(query);
    }
});

connection.on('end', function(err){
    console.log("connection End");
});

//SQL実行の実体
function executeStatement(query) {

    //応答を詰め込むバッファ
    var arr = [];
    var con_cnt = 5; //取得する列数

    console.log("executeStatement start");
    queryrequest = new TedRequest(query, function(err){
        if (err) {
            console.log(err.stack);
        } else {
            //応答を返す
            console.log(arr[0][0]+arr[0][1]+arr[0][2] +arr[0][3] +arr[0][4]);
        }
    });
};
```



```
//検索条件の指定
queryrequest.addParameter('WHO', TYPES.NVarChar, q_name);

queryrequest.on('row', function(columns) {
    var record = new Array(con_cnt);
    var i = 0;
    columns.forEach( function(column) {
        record[i++] = column.value;
        console.log(column.metadata.colName + ": " + column.value);
    });
    arr.push(record);
});

queryrequest.on('requestCompleted', function(rowCount, more){
    console.log(arr.length + ' rows found');
    console.log(rowCount + ' rows returned');
    connection.close();
});

connection.execSql(queryrequest);

return;
}
};
```

index.js hosted with ❤ by GitHub

[view raw](#)

上記コードは SQL Server のテーブルから取得したデータをログに出力する `scan_table` 関数です。
`query` 文や DB 接続情報などは、実際の環境に合わせて適宜修正してください。
最後に、今追加した関数を呼び出す処理を追記すれば、実装完了です。デフォルトで生成されていたコードを以下の内容に変更します。

```
module.exports = async function (context, req) {
    context.log('JavaScript HTTP trigger function processed a request.');
```



```
    if (req.query.name || (req.body && req.body.name)) {
        context.res = {
            // status: 200, /* Defaults to 200 */
            body: "Hello " + (req.query.name || req.body.name)
        };

        //データベースアクセスと応答
        scan_table(context, req.query.name);
    }
}
```

```
}  
else {  
  context.res = {  
    status: 400,  
    body: "Please pass a name on the query string or in the request body"  
  };  
}  
};
```

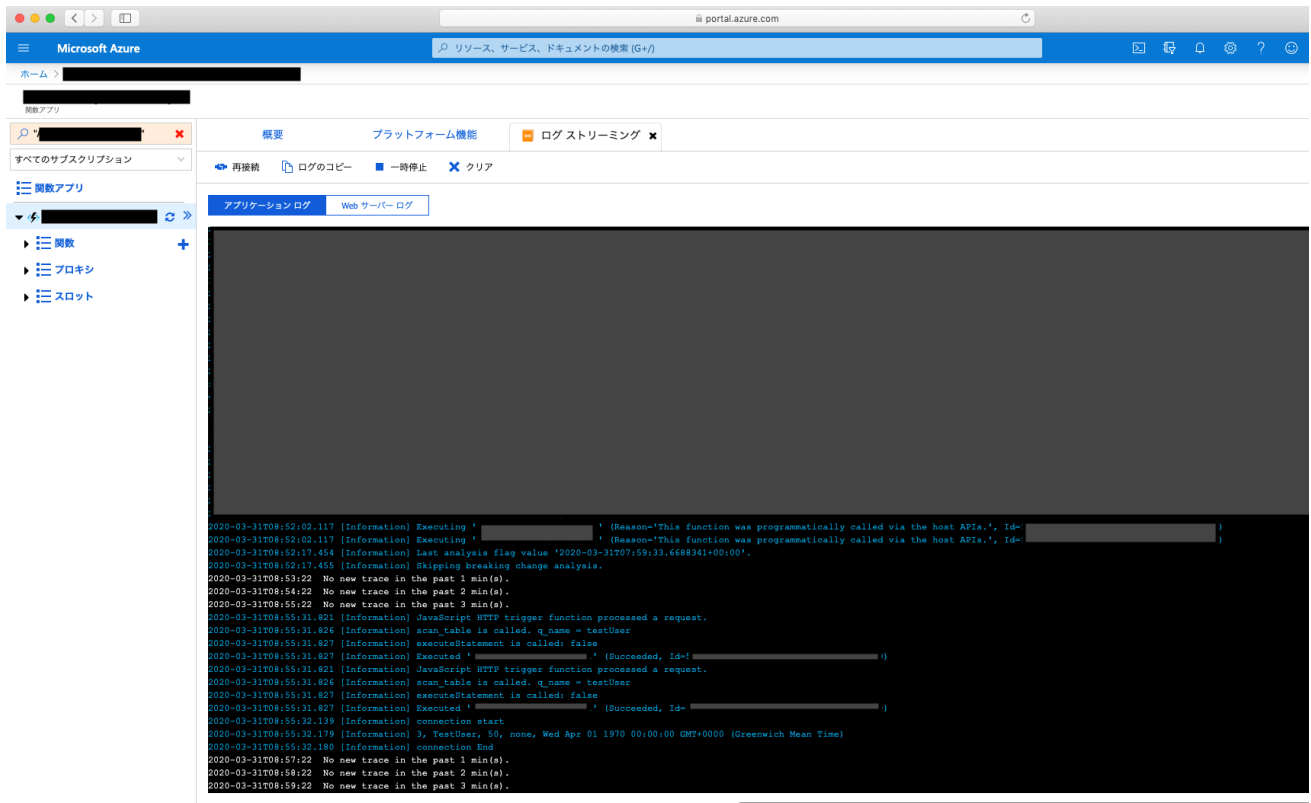
indexadd.js hosted with ❤ by GitHub

[view raw](#)

11行目で、先ほど作成した `scan_table()` 関数の呼び出し処理を追加しています。これで、SQL Server のデータベースからデータ取得とログ出力が実行されるようになります。



「プラットフォーム機能」から「ログストリーミング」を起動して、「アプリケーションログ」に取得したデータがログ出力されていれば成功です。関数のエディタ下部の「ログ」欄には出力されないので、ご注意ください。



終わりに

今回は Node.js による Azure Functions での DB アクセス処理をしました。

Azure では他にも IoT や機械学習を行うためのサービスも提供しているので、ご興味がある方は是非使ってみてはいかがでしょうか。

それでは、今回はこの辺りで失礼します。