日日是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2023年1月13日金曜日

Google Indexing APIを呼び出す

Oracle APEXのアプリケーションより Google Indexing APIを呼び出す方法を調べてみました。

ポーランドのPretius社が公開している以下のブログ記事を参考にしています。

https://pretius.com/blog/google-workspace-integration-oracle-database-apex/#integration-service-account

Pretius社はOracle APEX界隈ではとても知名度の高い会社です。昨年（2022年）は、ポーランドのソフトウェアの業界団体（SoDA - Software Development Association Poland）より、“An IT project that supported Ukraine”というカテゴリにて、Oracle APEXを使って作成したアプリケーションにより表彰されています。

https://pretius.com/blog/why-low-code-how-apex-helped-refugees/

Googleが提供しているAPIの認証を通す方法はいくつかあります。Indexing APIは一番設定が簡単な（すなわち安全性が一番低い）APIキーによる認証はできません。

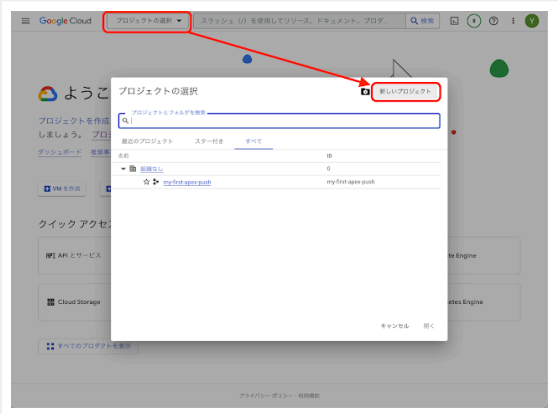
そのため、サービス・アカウントを使ってGoogle Indexing APIを呼び出すことにしました。

Google側の準備

Google Cloudのコンソールは使用経験がほとんど無いため、以下の記載は参考程度と考えてください。

最初にGoogle Cloudのコンソールを開きます。

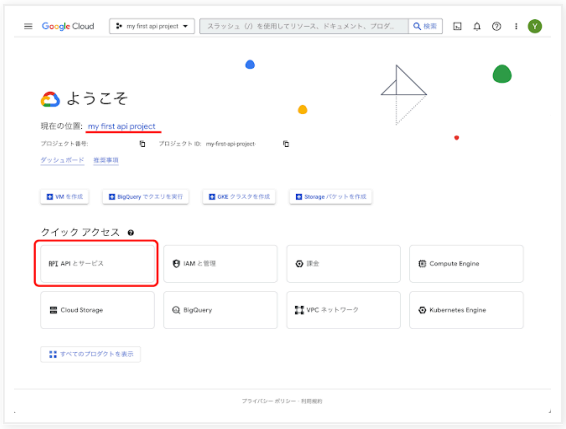
プロジェクトの選択をクリックし、開いたダイアログ上の**新しいプロジェクト**をクリックします。



プロジェクト名は任意ですが、ここでは**my first api project**としました。**作成**をクリックします。

プロジェクトが作成されます。先ほどの**プロジェクトの選択**のダイアログより、作成されたプロジェクトを開きます。

クイックアクセス（または左上のハンバーガー・メニューを開いて）より**APIとサービス**を開きます。

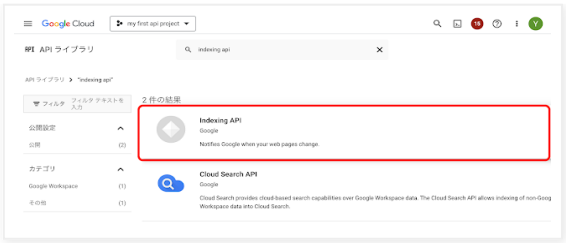


APIとサービスの画面の、**＋APIとサービスの有効化**をクリックします。

プロジェクト作成直後で15ほど利用可能なAPIとサービスが有効化されていました。使用する予定は無いため、すべて無効化しています。

今回使用する予定の**indexing api**を検索します。

検索された結果より、**Indexing API**を選択します。



Indexing APIを**有効**にします。

Indexing APIが有効になります。ウィザード形式で認証情報を作成するためのボタン**認証情報を作成**がありますが、認証情報はサービス・アカウントとキーを使うことに決めているので、ウィザードは使用しません。

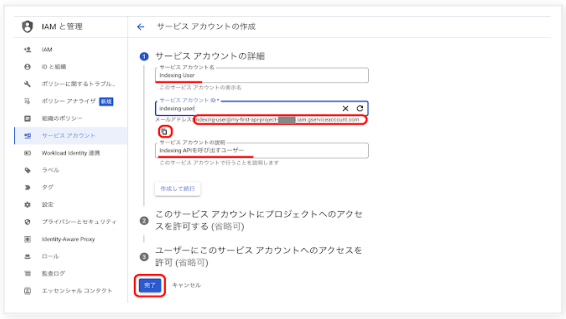
左のメニューより**認証情報**を開きます。

認証情報の画面より**＋認証情報を作成**をクリックし、**サービスアカウント**を選択します。

サービスアカウント名、**サービスアカウントID**、**サービスアカウントの説明**を設定します。任意の値を設定します。

これからの作業で使用するのは、サービスアカウントIDの**メールアドレス**です。クリップボードにコピーして、保存しておきます。

完了をクリックします。



サービスアカウントが作成されます。**鉛筆アイコン**（または**サービスアカウント名**）を**クリック**し、編集を開始します。

キー・タブを開き、**鍵を追加**より**新しい鍵の作成**を実行します。

キーのタイプとして**P12**（PKCS#12形式）を選択し、作成を実行します。この鍵はPL/SQLのパッケージDBMS_CRYPTOで使用するため、PKCS#1またはPKCS#8形式に変更します。JSONが推奨となっていますが、openssl コマンドを使用するため扱いやすいP12を選択しています。

作成をクリックします。

秘密鍵が手元のPCにダウンロードされます。openssl で形式変換する際に、秘密鍵のパスワードを聞かれるため、表示されている**秘密鍵のパスワード**をクリップボードに**コピー**し、保存しておきます。

閉じるをクリックします。

ダウンロードされたPKCS#12形式のファイルをPKCS#8形式に変換します。openssl pkcs12コマンドを使用します。

openssl pkcs12 -in ダウンロードされたファイル -nocerts -nodes -out 出力ファイル

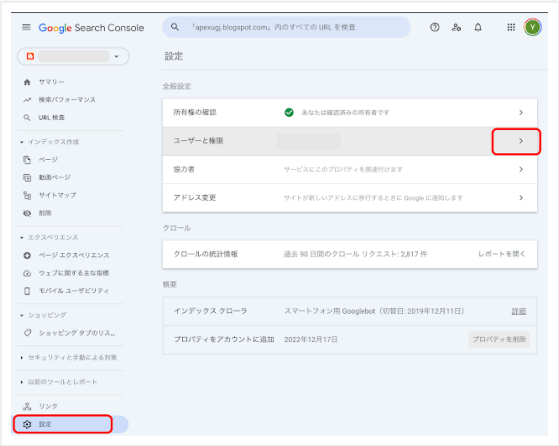
Enter Import Password:には**秘密鍵のパスワード**を入力します。

以下の例では、秘密鍵を**my-api.key**というファイルに出力しています。

```
% openssl pkcs12 -in my-first-api-project-*******.p12 -nocerts -nodes -out my-api.key
Enter Import Password: 秘密鍵のパスワード
MAC verified OK
%
```

Google Cloudのコンソールでの作業は、以上で終了です。

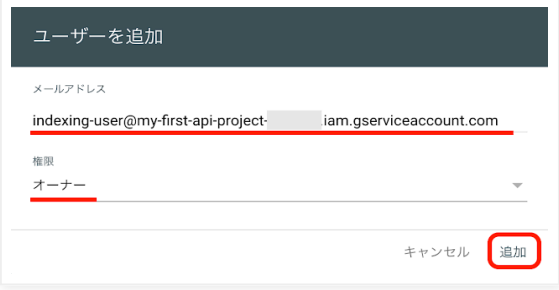
Google Search Consoleを開き、**設定**から**ユーザーと権限**を開きます。



ユーザーを追加を実行します。

メールアドレスとして、作成済みの**サービスアカウントのメールアドレス**を指定します。**権限**は**オーナー**を割り当てます。

追加をクリックします。



ユーザーの追加が確認されたら、Google Search Consoleでの作業は完了です。

Autonomous Databaseでの作業に移ります。

Oracle Database 19cより追加されたDBMS_CRYPTO.SIGNを使用します。Autonomous Databaseでは利用可能です。

JSON Web Tokenを作成するファンクションと作成したJWTを使ってトークンを取得するために使用するパッケージ**UTIL_GOOGLE_API**を作成します。

コードは以下になります。データベースで実行します。

```
create or replace package util_google_api
as
    C_TOKEN_URL constant varchar2(160) :=
        'https://oauth2.googleapis.com/token?grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-typ
    /*
     * Googleのプロジェクトに登録したサービス・アカウントのキーより
     * JWTを生成する。
     */
    function generate_jwt(
        p_secret    in varchar2
        ,p_iss      in varchar2 default null
        ,p_sub      in varchar2 default null
        ,p_scope    in varchar2 default null
        ,p_aud      in varchar2 default null
        ,p_iat      in timestamp default current_timestamp
        ,p_duration in number   default 3600 -- 秒で指定する
    )
    return varchar2;

    /*
     * JWTを使ってBearerトークンを取得する。
     */
    function get_token(
        p_jwt in varchar2
        ,p_credential_static_id in varchar2 default null
    )
    return varchar2;
end util_google_api;
/
```

```
create or replace package body util_google_api
as
    /*
     * OracleのTIMESTAMP型のデータをUNIX時間に変換する。
     */
    function unixtime(p_timestamp in timestamp)
    return pls_integer
    is
        l_date date;
        l_epoc number;
    begin
        l_date := sys_extract_utc(p_timestamp);
        l_epoc := l_date - date'1970-01-01';
        return l_epoc * 24 * 60 * 60;
    end unixtime;

    /* BASE64のデコード */
    function from_base64(t in varchar2) return varchar2 is
    begin
        return utl_raw.cast_to_varchar2(utl_encode.base64_decode(utl_raw.cast_to_raw(t)));
    end from_base64;

    /* BASE64へのエンコード  - RAWより */
    function to_base64_from_raw(t in raw) return varchar2 is
        l_base64 varchar2(32767);
    begin
        l_base64 := utl_raw.cast_to_varchar2(utl_encode.base64_encode(t));
        l_base64 := replace(l_base64, chr(13)||chr(10), '');
        return l_base64;
    end to_base64_from_raw;

    /* BASE64へのエンコード  - VARCHAR2 */
    function to_base64(t in varchar2) return varchar2 is
    begin
        return to_base64_from_raw(utl_raw.cast_to_raw(t));
    end to_base64;

    /* 秘密鍵を一行にする。 */
    function convert_to_single_line(
        p_string in varchar2
    )
    return varchar2
    as
    begin
        return regexp_replace(
            p_string
            ,'(-+((BEGIN|END) (RSA )?(PUBLIC|PRIVATE) KEY)-+\s?|\s)'
```

```
        ,''
    );
end convert_to_single_line;

/* JWTを生成する実装 */
function generate_jwt(
    p_secret    in varchar2
    ,p_iss      in varchar2
    ,p_sub      in varchar2
    ,p_scope    in varchar2
    ,p_aud      in varchar2
    ,p_iat      in timestamp
    ,p_duration in number   -- second
)
return varchar2
as
    l_iat            pls_integer;
    l_exp            pls_integer;
    l_header_json    json_object_t;
    l_header_str     varchar2(32767);
    l_header_base64  varchar2(32767); -- 1st part of JWT
    l_payload_json   json_object_t;
    l_payload_str    varchar2(32767);
    l_payload_base64 varchar2(32767); -- 2nd part of JWT
    l_data     varchar2(32767);
    l_hmac_raw raw(32767);
    l_hmac     varchar2(32767);       -- 3rd part of JWT
    l_jwt      varchar2(32767);
begin
    /* iatとexpとなる値を求める。 */
    l_iat := unixtime(p_iat);
    l_exp := l_iat + p_duration;

    /* ヘッダーを手作業で作成し、BASE64でエンコードする。 */
    l_header_json := json_object_t();
    l_header_json.put('alg','RS256');
    l_header_json.put('typ','JWT');
    l_header_str := l_header_json.to_string();
    l_header_base64 := to_base64(l_header_str); -- ヘッダー

    /* ペイロードを手作業で作成し、BASE64でエンコードする。 */
    l_payload_json := json_object_t();
    if p_iss is not null then
        l_payload_json.put('iss', p_iss);
    end if;
    if p_sub is not null then
        l_payload_json.put('sub', p_sub);
```

```
    end if;
    if p_scope is not null then
        l_payload_json.put('scope', p_scope);
    end if;
    if p_aud is not null then
        l_payload_json.put('aud', p_aud);
    end if;
    l_payload_json.put('iat',   l_iat);
    l_payload_json.put('exp',   l_exp);
    l_payload_str := l_payload_json.to_string();
    l_payload_base64 := to_base64(l_payload_str); -- ペイロード

    -- シグネチャを手作業で作成する。
    l_data := l_header_base64 || '.' || l_payload_base64;
    l_hmac_raw := dbms_crypto.sign(
        src => utl_i18n.string_to_raw(l_data,'AL32UTF8'),
        prv_key => utl_i18n.string_to_raw(convert_to_single_line(p_secret),'AL32UTF8'),
        pubkey_alg => DBMS_CRYPTO.KEY_TYPE_RSA,
        sign_alg => DBMS_CRYPTO.SIGN_SHA256_RSA
    );
    l_hmac := to_base64_from_raw(l_hmac_raw);
    l_hmac := trim(translate(l_hmac, '+/=', '-_ ')); -- HMAC
    /* JSON Web Tokenを返す。 */
    l_jwt := l_header_base64 || '.' || l_payload_base64 || '.' || l_hmac;
    return l_jwt;
end generate_jwt;

/* トークンを取得する実装 */
function get_token(
    p_jwt in varchar2
    ,p_credential_static_id in varchar2
)
return varchar2
as
    l_request_url varchar2(32767);
    l_token_clob  clob;
    l_token_json  json_object_t;
    l_token       varchar2(32767);
begin
    l_request_url := C_TOKEN_URL || p_jwt;
    apex_web_service.clear_request_headers;
    apex_web_service.set_request_headers('Content-Length',0,p_reset => false);
    l_token_clob := apex_web_service.make_rest_request(
        p_url => l_request_url
        ,p_http_method => 'POST'
    );
    l_token_json := json_object_t(l_token_clob);
```

```
        l_token := l_token_json.get_string('token_type') || ' ' || l_token_json.get_string('acc
        /*
         * Web資格証明の静的IDが指定されている場合は、アップデートする。
         */
        if p_credential_static_id is not null then
            apex_credential.set_session_credentials(
                p_credential_static_id => p_credential_static_id
                ,p_username => 'Authorization'
                ,p_password => l_token
            );
        end if;
        return l_token;
    end get_token;
end util_google_api;
/
```

util_google_api.sql hosted with ❤️ by GitHub view raw

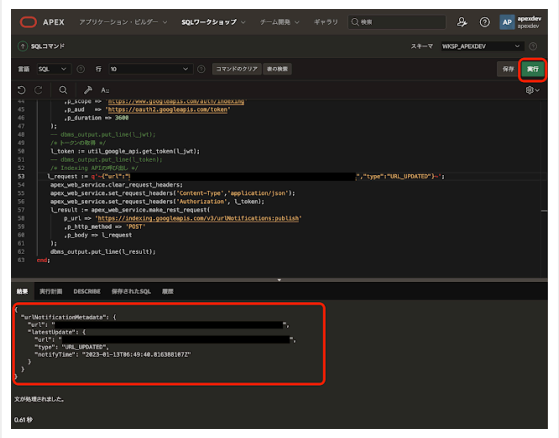
SQLコマンドよりGoogle Indexing APIを発行し、動作を確認します。秘密鍵やサービスアカウントのメールアドレスの部分は置き換えます。

```
declare
    l_jwt varchar2(32767);
    C_RSA_KEY constant varchar2(32767) := q'~
-----BEGIN PRIVATE KEY-----
opensslで生成した秘密鍵ファイルの内容をそのまま
貼り付ける。
-----END PRIVATE KEY-----
~';
    l_token_url    varchar2(4000);
    l_token_clob   clob;
    l_token_json   json_object_t;
    l_token        varchar2(32767);
    l_request clob;
    l_result  clob;
begin
    /* JWTの生成 */
    l_jwt := util_google_api.generate_jwt(
        p_secret => C_RSA_KEY
        ,p_iss   => 'サービスアカウントのメールアドレス'
        ,p_scope => 'https://www.googleapis.com/auth/indexing' -- Indexing APIのscope
        ,p_aud   => 'https://oauth2.googleapis.com/token'
    );
    -- dbms_output.put_line(l_jwt);
    /* トークンの取得 */
    l_token := util_google_api.get_token(l_jwt);
    -- dbms_output.put_line(l_token);
```

```
    /* Indexing APIの呼び出し */
    l_request := q'~{"url":"索引の作成を要求するURL","type":"URL_UPDATED"}~';
    apex_web_service.clear_request_headers;
    apex_web_service.set_request_headers('Content-Type','application/json',p_reset => false);
    apex_web_service.set_request_headers('Authorization', l_token,p_reset => false);
    l_result := apex_web_service.make_rest_request(
        p_url => 'https://indexing.googleapis.com/v3/urlNotifications:publish'
        ,p_http_method => 'POST'
        ,p_body => l_request
    );
    dbms_output.put_line(l_result);
end;
```

test-google-indexing-api.sql hosted with ❤️ by GitHub view raw

JSONのレスポンスが正常に返されていれば、すべての作業は完了です。



以上で動作確認も完了です。

実際には、データベースのどこかに秘密鍵を安全に保存する必要もあり、また、生成したJWTや取得したトークンをキャッシュしておくといった実装も必要になるかと思います。

本記事は以上になります。

完

Yuji N. 時刻: 16:30

共有

‹ ホーム ›

ウェブ バージョンを表示

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示