Consumption of an API via Orange Gateway OPE

OPE clients are allowed to:

- Request a token via the token server offered by OPE available via the following URL: https://ope.apibackbone.orange.com/oauth/v3/token
- Use this token to consume the API via OPE

1. Getting a token from OPE

A token is a proof of authentication and authorization allowing, for a short period of time, to consume an API without the need to authenticate. The format of the tokens issued by OPE is called JWT (JSON Web Token).

Oauth2 authentication with JWT tokens allows to answer the following points:

- Single Sign On: the client requests a token once and then uses it as long as its lifetime has not expired.
- Security of the authentication: the APIs consumed by the client do not receive the password or the secret of the client, but only the token. The APIs cannot usurp the identity of the client.

The operation is as follows:

1. The client must obtain a proof of authentication from the OPE token server. It thus requests a JWT token by transmitting in its request the `client_id` (identifier of the client) and the `client_secret` (password of authentification).
2. The OPE token server checks the authentication (`client_id` + `client_secret`). If so, then it will provide a token in its response.
3. The client will be able to use this token to consume the APIs by conveying it in the Authorization header of each call. In order not to penalize the execution time and not to unnecessarily load the OPE token server, the client must hide the token so that it is reused in each API call.
4. The token has a limited validity in time (60 minutes). The client must therefore request a new token periodically, before the current token expires.

Request :

The URL to obtain authentication tokens is the following:

- https://ope.apibackbone.orange.com/oauth/v3/token

Type	Field	Explanation	Example
Header	`Authorization`	`Basic` followed by `client_id:client_secret` encoded in base64	`Basic Y2xPLW[..]ZRTYUX`

Type	Field	Explanation	Example
Header	`Accept`	`application/json;charset=utf-8`	
Header	`Content-Type`	`application/x-www-form-urlencoded`	
Data	`grant_type`	Type of authorization. OPE currently only supports the grant_type `client_credentials`	`client_credentials`

Response :

Field	Explanation	Example
`access_token`	Used JWT token to authenticate the client to the service	`eyJ0eXA[..]VnNM`
`token_type`	Always the following string : `bearer`	`Bearer`
`expires_in`	Validity time of the access token in seconds (60 minutes)	3600

Example :

```
{
   "token_type": "Bearer",
   "access_token": "eyJ0eXA[..]VnNM",
   "expires_in": 3600
}
```

2. Call to the API through OPE

The request to the API must go through OPE via the following URLs:

- Partners
  - ProductOrdering
    - https://ope.apibackbone.orange.com/mba/productordering/v2/event
  - UserService
    - https://ope.apibackbone.orange.com/mba/userservice/v2/event

Request :

Type	Field	Explanation	Example
Header	`Authorization`	`Bearer` suivi du token JWT	`Bearer eyJ0eXA[..]VnNM`

Example via CURL

```
# Call the service through api gateway with OPE authentication data
curl --silent \
    --show-error \
    --url "${service_url}" \
    --header "Authorization: Bearer ${access_token}"
```

3. Example illustrating the implementation of the token renewal policy, cached in native PHP

```php
<?php // config.inc.php
    $OPE_TOKEN_SERVER_URL = 'https://ope.apibackbone.orange.com/oauth/v3/token
';
    $OPE_CLIENTID = 'cli-my_client-v1-prd';
    $OPE_CLIENTSECRET = 'abcdrefghijkl';
?>
<?php // opev2.inc.php
    /**
     * Check if the token is in cache.
     * If cache is empty, request a new token.
     */
    function check_or_renew_token() {
        $cache = read_cache();

        // If token is not in cache, trigger a request to renew it.
        if( !isset($cache) ) {
            // No OPE tokens found in cache, request for new tokens
            renew_tokens();
        } else {
            // OPE tokens found in cache, check if one of them is expired
            foreach($cache as $token) {
                $now = new DateTime();
                $exp = new DateTime();
                $exp->setTimestamp($token->{'expiration'});
                $diff = $now->diff($exp)->format('%R%H:%I:%S');
                if($diff[0] !== '+') {
                    // One of the tokens have expired, renew all tokens.
                    renew_tokens();
                    break;
                }
            }
        }
    }

    /**
     * Request OPE Token for new tokens,
     * by exchanging a secret. Store the tokens in cache.
     */
    function renew_tokens() {
        require_once 'api_call.inc.php';
        require_once 'config.inc.php';

        //store all tokens returned by api calls
        $tokens = array();
```

```
        $fields = array(
            'grant_type' => 'client_credentials',
            'client_id' => $OPE_CLIENTID,
            'client_secret' => $OPE_CLIENTSECRET
        );
        $fields = http_build_query($fields);
        $headers = array(
                'Content-Type: application/x-www-form-urlencoded',
                'Accept: application/json'
        );
        $method = 'POST';

        $resp = call_api($OPE_TOKEN_SERVER_URL, $method, $headers, $fields);

        if(empty($resp['code'])) {
                error_log('Couldn\'t contact OPE, abort');
                throw new Exception('Couldn\'t contact OPE, abort');
        }

        if(empty($resp['error'])) {
                $tokens[] = $resp['res'];
        }

        // Once all the token are fetched, store them in cache
        $json = array();
        foreach($tokens as $token) {
            // Compute when the token will expire (based on expires_in field o
f
            // the response) and store it in cache
            $seconds = $token['expires_in'];
            $now = new DateTime();
            $interval = new DateInterval('PT'.$seconds.'S');
            $expiration = $now->add($interval)->getTimestamp();

            $json[] = array(
                'expiration' => $expiration,
                'token' => $token['access_token']
            );
        }

        if(empty($json)) {
            error_log('Couldn\'t obtain token, don\'t write the cache');
            return false;
        }
        write_cache( json_encode($json) );
    }

    function write_cache($tokens) {
```

```
        file_put_contents("/tmp/tokens", $tokens);
    }

    /**
     * Return the cached tokens in a JSON format,
     * or null if nothing was found in cache
     */
    function read_cache() {
        $tokens = @file_get_contents("/tmp/tokens");

        if(empty($tokens)) {
            error_log('No cache file found');
            return NULL;
        }
        return json_decode($tokens);
    }

    /**
     * Get a token from cache
     */
    function get_token() {
        $cache = read_cache();
        if( !isset($cache) ) {
            return NULL;
        }

        foreach($cache as $token) {
            return $token->{'token'};
        }
        return NULL;
    }
?>
<?php //api_call.inc.php
    /**
     * Generic function to call an API
     * $url: url of the endpoint.
     * $method: HTTP method
     * $fields: post data in the request payload
     */
    function call_api($url, $method, $headers, $fields) {
        $curl = curl_init();
        curl_setopt($curl, CURLOPT_URL, $url);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, True);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, $method);
        curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, False);
        curl_setopt($curl, CURLOPT_POST, True);
        curl_setopt($curl, CURLOPT_POSTFIELDS, $fields);
        curl_setopt($curl, CURLOPT_HTTPHEADER, $headers);
```

```
        $response = curl_exec($curl);
        $httpcode = curl_getinfo($curl, CURLINFO_RESPONSE_CODE);
        curl_close($curl);

        $ret = array(
            'code'=>$httpcode,
            'res'=>json_decode($response, True),
            'error'=>($httpcode>=400)
        );
        return $ret;
    }
?>
```