A beginners guide to Google OAuth and Google APIs

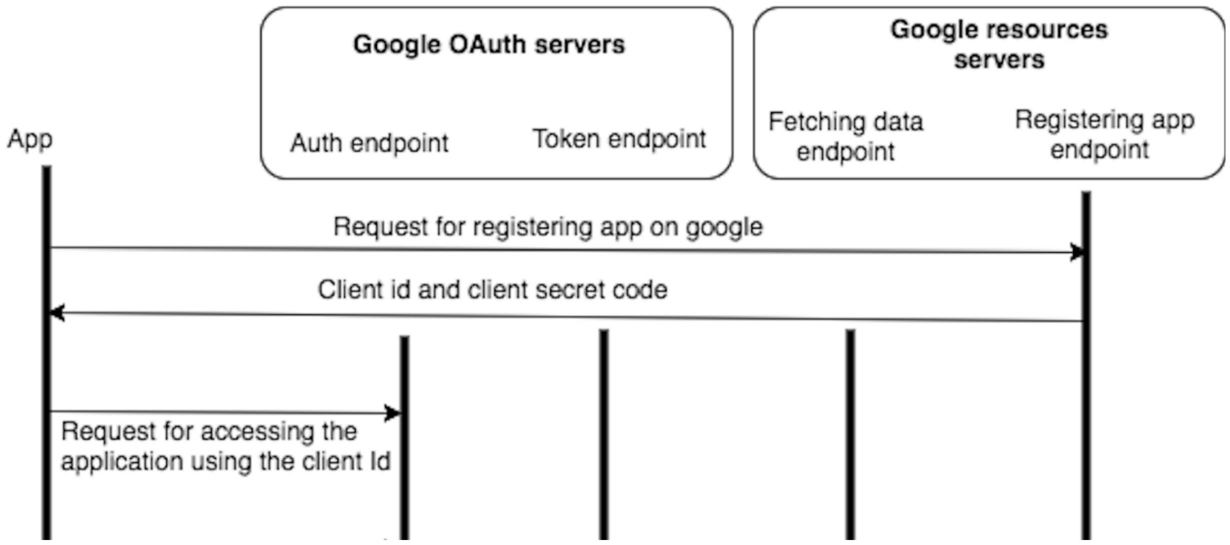
Ashok Yogi Follow
Jan 8, 2017 · 6 min read

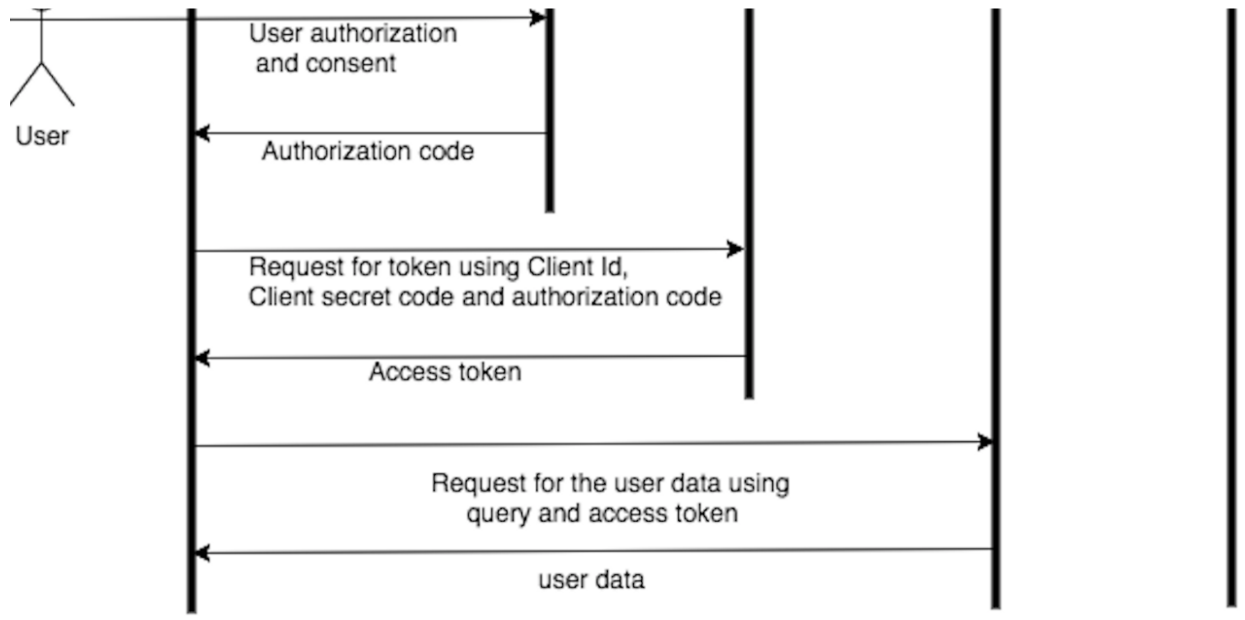
Starting something new is always an exciting task, but it becomes a nightmare when you can't find any good article written specifically for a naive.

In the last couple of days I stumbled upon infinite articles pertaining to Google OAuth and Google APIs but all my efforts to find a lucid one which focuses on the implementation went in vain. This is my attempt to pen down my learnings gained from few sleepless nights after strolling through endless articles, tutorials and of course our beloved Stack Overflow.

The guide is meant for technical audience and the emphasis would be on the implementation. I will use Google Spreadsheet as the application for demonstrating the use of Google APIs. The code is implemented in Python.

TL;DR: This is the diagram which show the requests made to the OAuth and Google servers and responses received to authorize your app to access the data from google APIs.


Google OAuth servers
Google resources servers
App
Auth endpoint
Token endpoint
Fetching data endpoint
Registering app endpoint
Request for registering app on google
Client id and client secret code
Request for accessing the application using the client Id


User authorization and consent
User
Authorization code
Request for token using Client Id, Client secret code and authorization code
Access token
Request for the user data using query and access token
user data

Definitions:

a) OAuth: OAuth is an open standard for authorization, commonly used as a way for Internet users to authorize websites or applications to access their information on other websites but without giving them the passwords. In other words, OAuth provides your App access to your personal data without exposing user's password.

b) OAuth 2.0: It is the latest version of OAuth and is the most widely used one. The key thing to note is that OAuth 2.0 is a completely new protocol and is **not** backward compatible with the previous versions. A lot of things have been changed from OAuth 1.0 to OAuth 2.0 but I won't cover these changes in this article. Also, whenever I write just *"OAuth"*, I'm actually talking about OAuth 2.0 — as it is most likely what you will be using.

c) Client Id: A unique string representing the registration information provided by the client. It is the identifier for your application.

d) Client secret: It is the client password that is used to authenticate with the authentication server (Google server in this case), that authenticates the client.

e) Access token: A token with a limited lifetime, issued to the client after authorization which state the scopes and privileges for a client.

f) Refresh token: Refresh token is used to obtain new access tokens when the previous one gets expired.

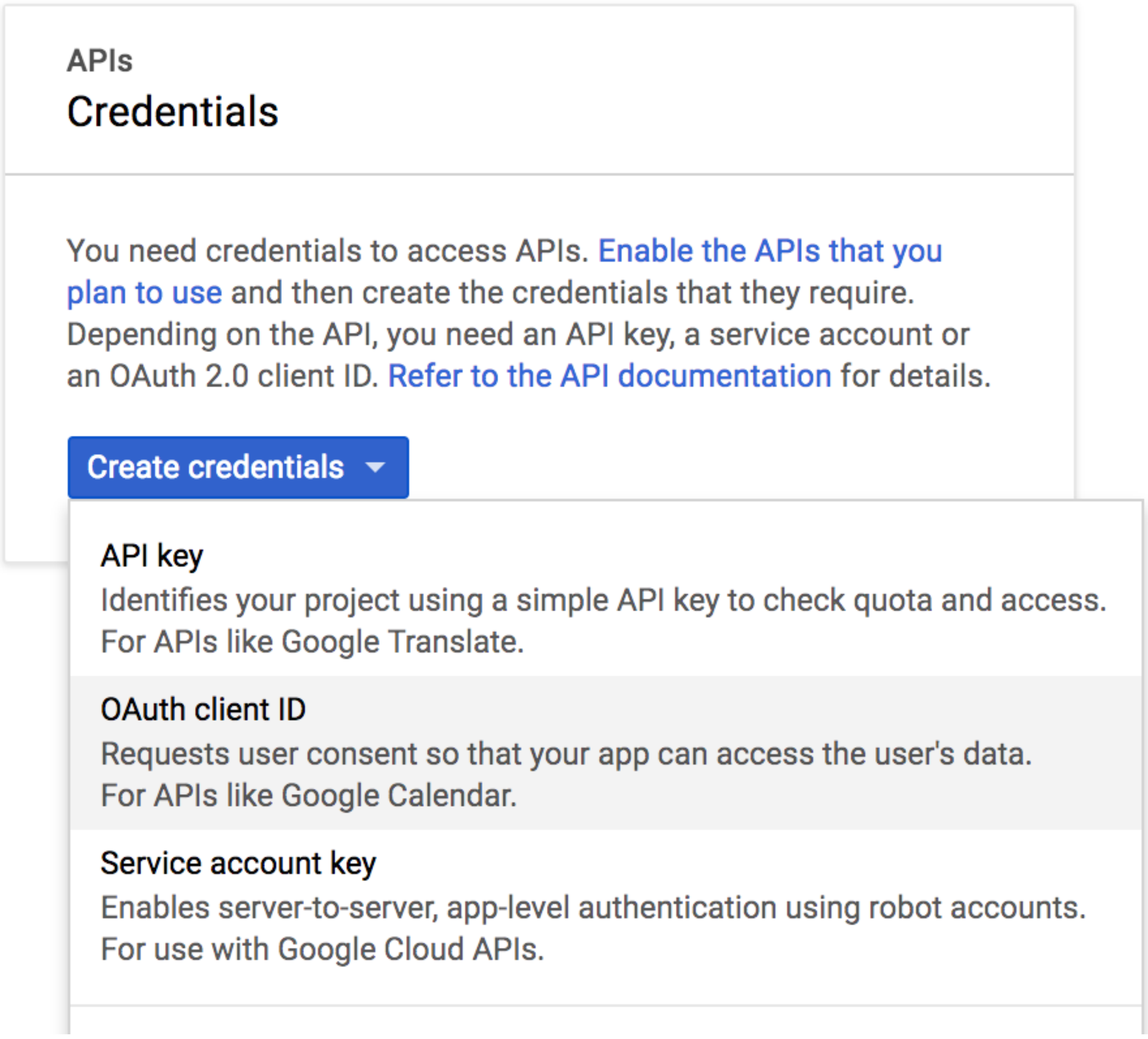
So let's get started!

Steps:

a) Obtain OAuth 2.0 credentials from the Google API Console: This is the first step which requires you to register your App on the Google console to obtain OAuth 2.0 credentials.

Fist of all, open the Google console link and create an new project by opening the "*Credentials*" tab.

Get the credentials by selecting *"Create Credentials"* -> *"OAuth Client Id"*



APIs

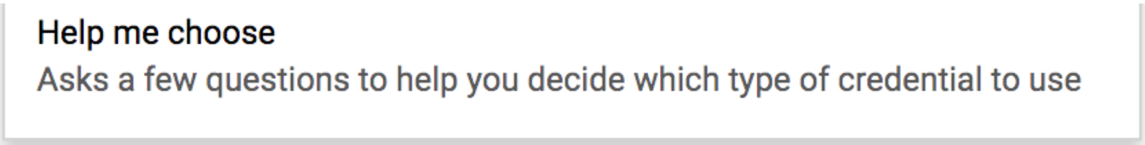
Credentials

You need credentials to access APIs. Enable the APIs that you plan to use and then create the credentials that they require. Depending on the API, you need an API key, a service account or an OAuth 2.0 client ID. Refer to the API documentation for details.

API key
Identifies your project using a simple API key to check quota and access. For APIs like Google Translate.

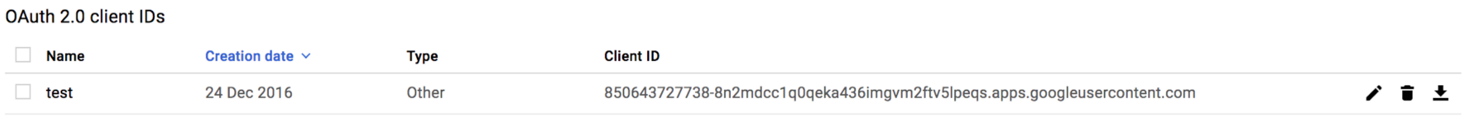
OAuth client ID
Requests user consent so that your app can access the user's data. For APIs like Google Calendar.

Service account key
Enables server-to-server, app-level authentication using robot accounts. For use with Google Cloud APIs.



Help me choose
Asks a few questions to help you decide which type of credential to use

Set the product name on the OAuth consent screen and then select the Application as *“Other”*. Download the credentials by clicking the download icon on the OAuth 2.0 client IDs section



OAuth 2.0 client IDs

Name	Creation date	Type	Client ID
test	24 Dec 2016	Other	850643727738-8n2mdcc1q0qeka436imgvm2ftv5lpeqs.apps.googleusercontent.com

The credentials will be a json document containing **client Id** and **client secret** code(refer to the definitions).

```
{
  "installed": {
    "client_id": "850643727738-
8n2mdcc1q0qeka436imgvm2ftv5lpeqs.apps.googleusercontent.com",
    "project_id": "test-153513",
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://accounts.google.com/o/oauth2/token",
    "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
    "client_secret": "Your client secret code",
    "redirect_uris": [
      "urn:ietf:wg:oauth:2.0:oob",
      "http://localhost"
    ]
  }
}
```

b) The next step is to use this Client Id and Client secret code to obtain the authorization code. The easiest way to do this is to use The oauth2client library which is the Google APIs Client Library for Python. It handles all steps of the OAuth 2.0 protocol required for making API calls.

```
import httplib2
```

```
from oauth2client.client import flow_from_clientsecrets
from oauth2client.file import Storage
from oauth2client.tools import run_flow

CLIENT_SECRET = 'client_secret.json'
SCOPE = 'https://www.googleapis.com/auth/spreadsheets.readonly'
STORAGE = Storage('credentials.storage')

# Start the OAuth flow to retrieve credentials
def authorize_credentials():
# Fetch credentials from storage
    credentials = STORAGE.get()
# If the credentials doesn't exist in the storage location then run
the flow
    if credentials is None or credentials.invalid:
        flow = flow_from_clientsecrets(CLIENT_SECRET, scope=SCOPE)
        http = httplib2.Http()
        credentials = run_flow(flow, STORAGE, http=http)
    return credentials

credentials = authorize_credentials()
```

Executing this code will open your browser where you have to provide permission for your app to access your spreadsheet data.

So what's going on?

```
CLIENT_SECRET = 'client_secret.json'
```

First, we declare the path of our credentials which we got from **step a**.

```
SCOPE = 'https://www.googleapis.com/auth/spreadsheets.readonly'
```

Next, we define the scope of the token obtained. In this example we have provided the scope of reading from the google spreadsheet.

```
STORAGE = Storage('credentials.storage')
credentials = STORAGE.get()
```

Now, we create a storage object which loads “credentials.storage”. If it doesn’t exists, it gets created automatically. The credentials is stored in the storage path so that we don’t need to authorize the client with each and every request.

```
if credentials is None or credentials.invalid:
        flow = flow_from_clientsecrets(CLIENT_SECRET, scope=SCOPE)
        http = httplib2.Http()
        credentials = run_flow(flow, STORAGE, http=http)
    return credentials
```

Now, we fetch the credentials from the storage object. Since, this is the first time we are authorizing the client, the credentials will be *None*. So we define a *flow* object using the *flow_from_clientsecrets* method in the *oauth2client.client*. *Flow* object has functions to re-direct the application to the browser so that the application can be authorized to access the data by the user. As soon as the application is authorized, the credentials get generated and stored in the *storage* object and henceforth, we won’t require *flow* object to authorize the application.

The credentials generated gets stored as a json file.

```
{
  "_module": "oauth2client.client",
  "scopes": [
    "https://www.googleapis.com/auth/spreadsheets.readonly"
  ],
  "token_expiry": "2016-12-24T18:58:11Z",
  "id_token": null,
  "access_token": "my_access_token",
  "token_uri": "https://accounts.google.com/o/oauth2/token",
  "invalid": false,
  "token_response": {
    "access_token": "my_access_token",
    "token_type": "Bearer",
    "expires_in": 3600,
    "refresh_token": "my_refresh_token"
  },
  "client_id": "850643727738-
8n2mdcc1q0qeka436imgvm2ftv5lpeqs.apps.googleusercontent.com",
  "token_info_uri": "https://www.googleapis.com/oauth2/v3/tokeninfo",
  "client_secret": "My client secret",
  "revoke_uri": "https://accounts.google.com/o/oauth2/revoke",
  "_class": "OAuth2Credentials",
  "refresh_token": "my_refresh_token",
  "user_agent": null
}
```

The access token in this credentials file can now be used to access the data from the spreadsheet.

```
def get_google_sheet(spreadsheetId):
    credentials = authorize_credentials()
    http = credentials.authorize(httplib2.Http())
    discoveryUrl = ('https://sheets.googleapis.com/$discovery/rest?
version=v4')
    service = discovery.build('sheets', 'v4', http=http,
discoveryServiceUrl=discoveryUrl)

    rangeName = 'A:J'
    result = service.spreadsheets().values().get(
        spreadsheetId=spreadsheetId, range=rangeName).execute()
    values = result.get('values', [])
    return values
```

We use the credentials obtained earlier to authorise an httplib2.Http object which the client library will use to issue HTTP requests. Finally we create the API service object which we can use to interact with the API.

The spreadsheetId is the unique Id given to each spreadsheet.

The rangeName = 'A:J' specifies the columns in the spreadsheet which we want to read.

Congratulations! You have successfully authorized your app and read the data from google spreadsheet. But don't get too excited. You won't be able to read the data after a specific amount of time(well, 1 hour in this case). This is because the access token gets expired and you have to re-generate the access token by using the refresh token.

```
from urllib import urlencode
from urllib2 import Request , urlopen
import json
def refresh_access_token():
# You can also read these values from the json file
    client_id = "my_client_id"
    client_secret = "my_client_secret_code"
    refresh_token = "my_refresh_token"
    request = Request('https://accounts.google.com/o/oauth2/token',
    data=urlencode({
      'grant_type':    'refresh_token',
      'client_id':     client_id,
      'client_secret': client_secret,
      'refresh_token': refresh_token
    }),
    headers={
      'Content-Type': 'application/x-www-form-urlencoded',
      'Accept': 'application/json'
    }
    )
    response = json.load(urlopen(request))
    return response['access_token']

access_token = refresh_access_token()
```

For refreshing the access token, you can just make a post call to '**https://accounts.google.com/o/oauth2/token**' along with the **grant_type, client_id, client_secret and refresh_token** in the request body.

The access token obtained can be used to create a credential object instead of the earlier approach of fetching the credential object from the storage object.

```
def get_credentials(access_token):
    user_agent = "Google Sheets API for Python"
    revoke_uri = "https://accounts.google.com/o/oauth2/revoke"
    credentials = oauth2client.client.AccessTokenCredentials(
                                    access_token=access_token,
                                    user_agent=user_agent,
                                    revoke_uri=revoke_uri)
    return credentials
```

The refresh token never expires so you can use it any number of times.

Now your application will be able to authorise itself, connect to your api of choice and interact with it via Google's client library.

It took me a while to navigate Google's documentation and a little bit of trial and error to get this working properly, but hopefully this article will allow you to skip all that pain and get right into the actual building of your application.

Oauth Google Spreadsheets Python Google Api