How do I authorise an app (web or installed) without user intervention?

Asked 7 years, 4 months ago Active 9 months ago Viewed 47k times

73

66

Let's say that I have a web app that needs to access Drive files in a background service. It will either own the files it is accessing, or be run in a Google Account with which the owner has shared the documents.

I understand that my app needs a refresh token, but **I don't want to write the code to obtain that** since I'll only ever do it once.

NB. This is NOT using a Service Account. The app will be run under a conventional Google account. Service Account is a valid approach in some situations. However the technique of using Oauth Playground to simulate the app can save a bunch of redundant effort, and applies to any APIs for which sharing to a Service Account is unsupported.

google-api google-drive-api google-oauth gmail-api

Share Follow edited Jan 27 '19 at 9:45 asked Nov 4 '13 at 11:37

pinoyyid
18.7k 11 49 100

2 Answers

Active Oldest Votes

151

✓

This can be done with the Oauth2 Playground at https://developers.google.com/oauthplayground

Steps:-

1. Create the Google Account (eg. my.drive.app@gmail.com) - Or skip this step if you are using an existing account.
2. Use the API console to register the mydriveapp (https://console.developers.google.com/apis/credentials/oauthclient?project=mydriveapp or just https://console.developers.google.com/apis/)
3. Create a new set of credentials. `Credentials/Create Credentials/OAuth Client Id` then select `Web application`
4. Include https://developers.google.com/oauthplayground as a valid redirect URI
5. Note the client ID (web app) and Client Secret
6. Login as my.drive.app@gmail.com
7. Go to Oauth2 playground
8. In Settings (gear icon), set

- OAuth flow: Server-side
- Access type: Offline
- Use your own OAuth credentials: TICK
- Client Id and Client Secret: from step 5

9. Click Step 1 and choose Drive API v3 https://www.googleapis.com/auth/drive (having said that, this technique also works for any of the Google APIs listed)
10. Click Authorize APIs. You will be prompted to choose your Google account and confirm access
11. Click Step 2 and "Exchange authorization code for tokens"
12. Copy the returned Refresh token and paste it into your app, source code or in to some form of storage from where your app can retrieve it.

Your app can now run unattended, and use the Refresh Token as described https://developers.google.com/accounts/docs/OAuth2WebServer#offline to obtain an Access Token.

NB. Be aware that the refresh token can be expired by Google which will mean that you need to repeat steps 5 onwards to get a new refresh token. The symptom of this will be a Invalid Grant returned when you try to use the refresh token.

NB2. This technique works well if you want a web app which access your own (and **only** your own) Drive account, without bothering to write the authorization code which would only ever be run once. Just skip step 1, and replace "my.drive.app" with your own email address in step 6. make sure you are aware of the security implications if the Refresh Token gets stolen.

See Woody's comment below where he links to this Google video https://www.youtube.com/watch?v=hfWe1gPCnzc

. . .

Here is a quick JavaScript routine that shows how to use the Refresh Token from the OAuth Playground to list some Drive files. You can simply copy-paste it into Chrome dev console, or run it with node. Of course provide your own credentials (the ones below are all fake).

```
  grant_type=refresh_token&client_id=${encodeURIComponent(client_id)}&client_secret=$


    let refresh_request = {

        body: post_body,
        method: "POST",
        headers: new Headers({
            'Content-Type': 'application/x-www-form-urlencoded'
        })
    }

    // post to the refresh endpoint, parse the json response and use the access
token to call files.list
    fetch(refresh_url, refresh_request).then( response => {
            return(response.json());
        }).then( response_json =>  {
            console.log(response_json);
            files_list(response_json.access_token);
    });
    });
}

// a quick and dirty function to list some Drive files using the newly acquired
access token
function files_list (access_token) {
    const drive_url = "https://www.googleapis.com/drive/v3/files";
    let drive_request = {
        method: "GET",
        headers: new Headers({
            Authorization: "Bearer "+access_token
        })
    }
    fetch(drive_url, drive_request).then( response => {
        return(response.json());
    }).then( list =>  {
        console.log("Found a file called "+list.files[0].name);
    });
}

get_access_token_using_saved_refresh_token();
```

Share Follow

edited Jun 11 '20 at 2:46
study
4,951 2 31 42

answered Nov 4 '13 at 11:37
pinoyyid
18.7k 11 49 100

6 Note that my research indicates that refresh tokens are 'long lived' and are not expired by Google, but can be revoked on the API Console. Also, Google has a short 4 minute video on how to get a refresh token from Playground: youtube.com/watch?v=hfWe1gPCnzc – woody Dec 13 '13 at 21:11

Good find on the video. I've seen the refresh token expire when I do iterative testing on a given Google account. Google's response was along the lines of "we only allow a finite number of extant refresh tokens, and will expire the older ones. There may be other unusual circumstances which also expire refresh tokens, so your app needs to deal with that as a possibility". – pinoyyid Dec 13 '13 at 21:26

This sounds perfect, but I still get The OAuth 2.0 access token has expired, and a refresh token is not available. Even though I get a successful response back from using the refresh token. cl.ly/image/011i1V1O2m1k. I'm passing this JSON into $client->setAccessToken – Sean Clark Feb 19 '15 at 15:15

So I got that working by making a cURL call to googleapis.com/oauth2/v3/token then adding my refresh token into that response and using that as my setAccessToken. Is there any built in method to get an access token via a refresh token? – Sean Clark Feb 19 '15 at 15:27

This is supposed to work `$client->refreshToken($refresh_token);` but it returns NULL. cURL does work though. – Sean Clark Feb 19 '15 at 15:42

@SeanClark you should ask those as separate questions as they may be specific to the php sdk. personally I avoid the SDKs and program the http calls directly as I find it more reliable and easier to debug problems – pinoyyid Feb 19 '15 at 16:36

2 in reality its much cleaner if you also code a separate setup page that generates the refresh token by asking permissions to an admin. the admin uses the page to deploy the app or reconfigure later. using the oauth playground is just a quick way to avoid writting such admin page. – Zig Mandel Aug 11 '15 at 14:22

13 in reality it's much cleaner if you don't code anything at all. Why waste hours if not days figuring out OAuth, wrangling the leaky abstractions of the Google libraries, all for an app that you will only run once? That's not clean, it's borderline insane. – pinoyyid Aug 11 '15 at 15:43

1 Where do I perform step 3? Also V2 is not showing up. Will try this with Drive V3 – Dan Jan 21 '17 at 17:26

this no longer works. It is not possible to add the playground under "Domain verification" and attempting to add it in the verification URL will result in a 400 "redirect_uri_mismatch" error. – fommil Apr 1 '17 at 9:20

1 @fommil Absolute nonsense! Not only does it work, I've updated the question with a JS snippet to prove it. Feel free to follow the steps more carefully. Domain verification is only required for webhook notifications, *not* OAuth. – pinoyyid Apr 1 '17 at 17:16

@pinoyyid ok, so maybe domain verification is not relevant. But step 4 no longer exists. I am simply not asked at this point to provide a `redirect_uri` (what you're calling a callback). Did you try this yourself to confirm? I selected "Other" between 3 and 4 because I am writing a headless server. Perhaps you're specifically talking about a web app or an android app or something. – fommil Apr 2 '17 at 10:34

@fommil I followed the instructions yesterday and they work *exactly* as stated - INCLUDING STEP 4! The result is a Refresh Token which can be used in ANY app - web, embedded, installed, HEADLESS SERVER, IoT. Instead of choosing "other" you should have chosen "Web Application". IT DOES NOT MATTER THAT YOU DON'T THINK YOU'RE WRITING A WEB APP - JUST CHOOSE "Web Application". I've updated the answer to reflect the latest jargon changes in the dev console. – pinoyyid Apr 2 '17 at 17:49

6 this is the best answer ever. i'd give it ten votes if i could. you just made the oauth insanity bearable. thanks! – mga Dec 19 '17 at 15:33

Thanks for the great instructions! The `access_token` generated from Google Playground worked just fine. I was able to view all the files in my Google Drive account. – webworm Sep 13 '18 at 20:41

@pinoyyid - Would this be considered a "hack" or work around? Should we be concerned that Google will block this method even though it works quite well for many use cases? – webworm Sep 14 '18 at 16:43

@webworm it's not a hack and the YT video linked above is from Google. There is no method to block as it's standard OAuth. The worst thing that could happen is Google take down the OAuth Playground website, but I can't think of any reason why that would be desirable as it's a useful aid to G developers. – pinoyyid Sep 14 '18 at 22:11

I have used camel google sheet endpoint, with this we are able to generate the refresh and access token, but it expires in 3600s. is there any solution to increase the timeout github.com/apache/camel/blob/master/components/… – Elango Mani Jan 8 '19 at 9:48

I found something like this anyone tried creating a service account for G suit for accessing google API's: support.google.com/a/answer/7378726?hl=en – Elango Mani Jan 8 '19 at 10:04

@ElangoMani the Access Token will always expire after 1 hour for security. This is why the Refresh Token exists to allow your app to generate new Access Tokens whenever they are needed. As an aside, welcome to StackOverflow. SO exists to ask questions and provide answers. It's not a discussion forum. Please post your questions as questions instead of comments. – pinoyyid Jan 8 '19 at 10:58

I always get: "The redirect URI in the request, developers.google.com/oauthplayground, does not match the ones authorized for the OAuth client." – m02ph3u5 Mar 13 '19 at 17:23

@m02ph3u5 post as a new question including screenshots of the OAuth playground and what redirect URI's you have defined in the API console (step 4 above). Note the URI must be an **exact** match, including the `https://` – pinoyyid Mar 13 '19 at 17:38

Some people might have problem using pinoyyid's method, just to make sure the redirect uri **is** developers.google.com/oauthplayground and **not** developers.google.com/oauthplayground , even though it was just a forward slash, the auth will determine that as different URI. – Mahdiar Naufal Sep 11 '19 at 19:27

@pinoyyid what did you mean by RT could be expired by Google ? Does this mean that once we generated an RT it will not expire until google decides to shut it down ? Thanks – str028 Nov 19 '19 at 7:23

@str028 under normal circumstances the RT will not expire. – pinoyyid Nov 19 '19 at 22:33

Let me add an alternative route to pinoyyid's excellent answer (which didn't work for me -

6

popping redirect errors).

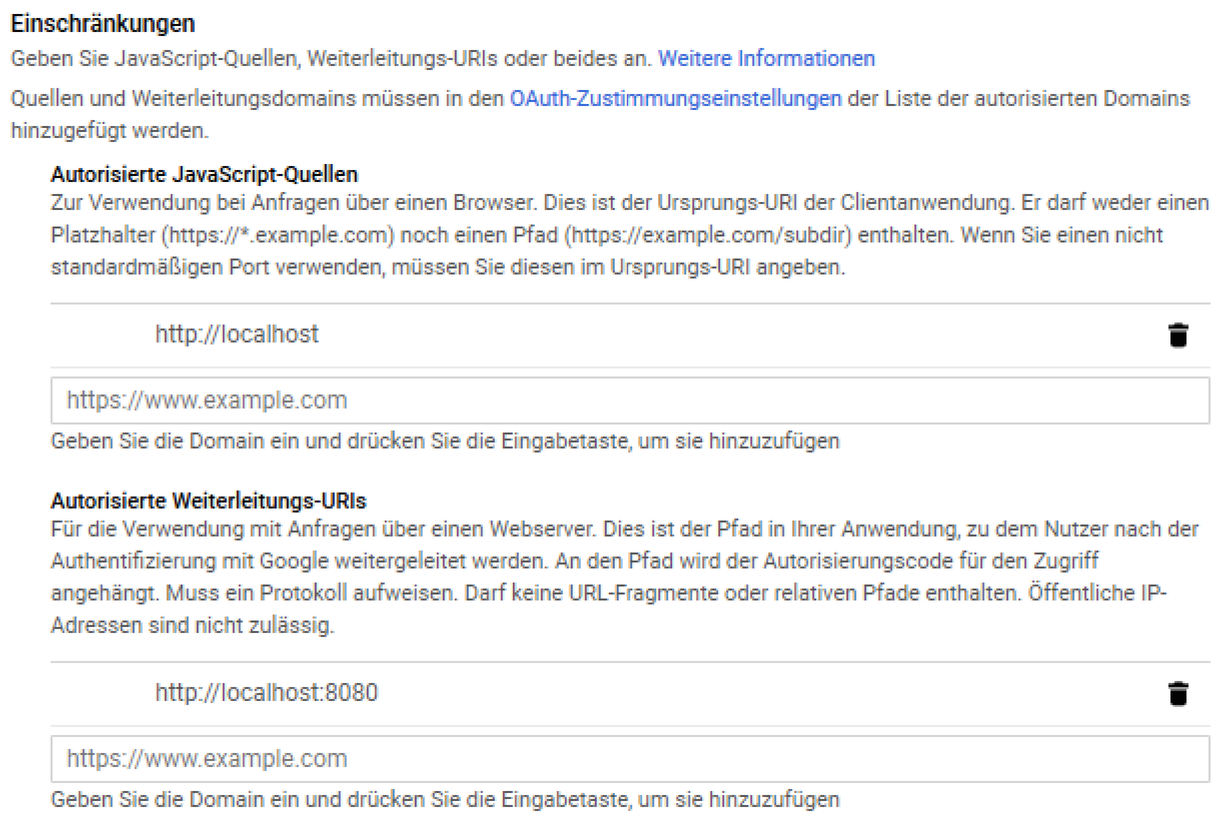
Instead of using the OAuthPlayground you can also use the HTTP REST API directly. So the difference to pinoyyid's answer is that we'll do things locally. Follow steps 1-3 from pinoyyid's answer. I'll quote them:

1. Create the Google Account (eg. my.drive.app@gmail.com) - Or skip this step if you are using an existing account.
2. Use the API console to register the mydriveapp (https://console.developers.google.com/apis/credentials/oauthclient?project=mydriveapp or just https://console.developers.google.com/apis/)
3. Create a new set of credentials (NB OAuth Client ID not Service Account Key and then choose "Web Application" from the selection)

Now, instead of the playground, add the following to your credentials:

Authorized JavaScript Sources: http://localhost (I don't know if this is required but just do it.)
Authorized Redirect URIs: http://localhost:8080

Screenshot (in German):



Einschränkungen

Geben Sie JavaScript-Quellen, Weiterleitungs-URIs oder beides an. Weitere Informationen

Quellen und Weiterleitungsdomains müssen in den OAuth-Zustimmungseinstellungen der Liste der autorisierten Domains hinzugefügt werden.

Autorisierte JavaScript-Quellen

Zur Verwendung bei Anfragen über einen Browser. Dies ist der Ursprungs-URI der Clientanwendung. Er darf weder einen Platzhalter (https://*.example.com) noch einen Pfad (https://example.com/subdir) enthalten. Wenn Sie einen nicht standardmäßigen Port verwenden, müssen Sie diesen im Ursprungs-URI angeben.

http://localhost

https://www.example.com

Geben Sie die Domain ein und drücken Sie die Eingabetaste, um sie hinzuzufügen

Autorisierte Weiterleitungs-URIs

Für die Verwendung mit Anfragen über einen Webserver. Dies ist der Pfad in Ihrer Anwendung, zu dem Nutzer nach der Authentifizierung mit Google weitergeleitet werden. An den Pfad wird der Autorisierungscode für den Zugriff angehängt. Muss ein Protokoll aufweisen. Darf keine URL-Fragmente oder relativen Pfade enthalten. Öffentliche IP-Adressen sind nicht zulässig.

http://localhost:8080

https://www.example.com

Geben Sie die Domain ein und drücken Sie die Eingabetaste, um sie hinzuzufügen

Make sure to **actually save** your changes via the blue button below!

Now you'll probably want to use a GUI to build your HTTP requests. I used Insomnia but you can go with Postman or plain cURL. I recommend Insomnia for it allows you to go through the consent screens easily.

Build a new *GET* request with the following parameters:

```
URL: https://accounts.google.com/o/oauth2/v2/auth
Query Param: redirect_uri=http://localhost:8080
Query Param: prompt=consent
Query Param: response_type=code
Query Param: client_id=<your client id from OAuth credentials>
Query Param: scope=<your chosen scopes, e.g.
https://www.googleapis.com/auth/drive.file>
Query Param: access_type=offline
```

If your tool of choice doesn't handle URL encoding automagically make sure to get it right yourself.

Before you fire your request set up a webserver to listen on `http://localhost:8080` . If you have node and npm installed run `npm i express` , then create an `index.js` :

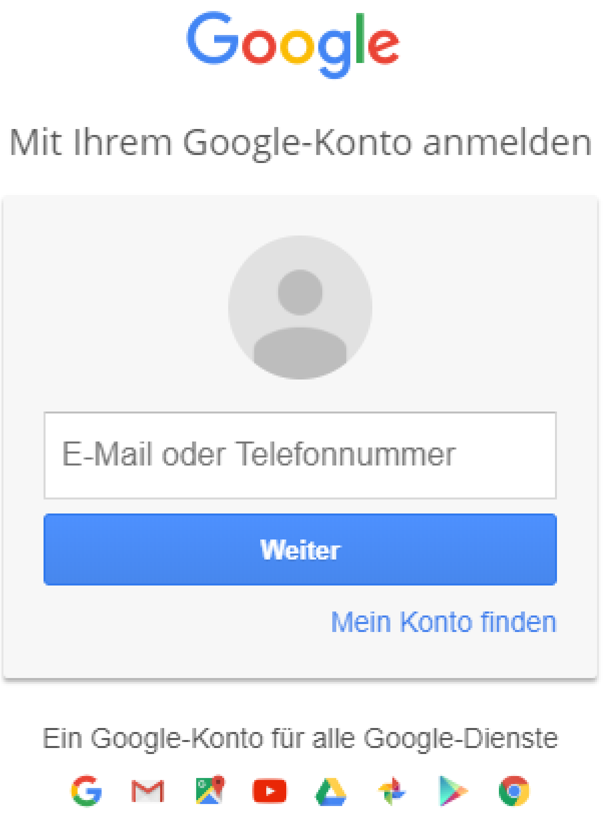
```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('ok');
  console.log(req)
});

app.listen(8080, function () {
  console.log('Listening on port 8080!');
});
```

And run the server via `node index.js` . I recommend to either not log the whole `req` object or to run `node index.js | less` for the full output will be huge.
There are very simple solutions for other languages, too. E.g. use PHP's built in web server on 8080 `php -S localhost:8080` .

Now fire your request (in Insomnia) and you should be prompted with the login:


Google
Mit Ihrem Google-Konto anmelden
E-Mail oder Telefonnummer
Weiter
Mein Konto finden
Ein Google-Konto für alle Google-Dienste

Log in with your email and password and confirm the consent screen (should contain your chosen scopes).

Go back to your terminal and check the output. If you logged the whole thing scroll down (e.g. pgdown in less) until you see a line with `code=4/...` .

Copy that code; it is your authorization code that you'll want to exchange for an access and refresh token. Don't copy too much - if there's an ampersand `&` do not copy it or anything after. `&` delimits query parameters. We just want the `code` .

Now set up a HTTP POST request pointing to `https://www.googleapis.com/oauth2/v4/token` as *form URL encoded*. In Insomnia you can just click that - in other tools you might have to set the header yourself to `Content-Type: application/x-www-form-urlencoded` .

Add the following parameters:

```
code=<the authorization code from the last step>
client_id=<your client ID again>
client_secret=<your client secret from the OAuth credentials>
redirect_uri=http://localhost:8080
grant_type=authorization_code
```

Again, make sure that the encoding is correct.

Fire your request and check the output from your server. In the response you should see a JSON object:

```
{
  "access_token": "xxxx",
  "expires_in": 3600,
  "refresh_token": "1/xxxx",
  "scope": "https://www.googleapis.com/auth/drive.file",
  "token_type": "Bearer"
}
```

You can use the `access_token` right away but it'll only be valid for one hour. Note the refresh token. This is the one you can always* exchange for a new access token.

`*` You will have to repeat the procedure if the user changes his password, revokes access, is inactive for 6 months etc.

Happy *OAuthing*!

Share Follow

answered Mar 14 '19 at 14:06
m02ph3u5
2,672 6 33 45

I've just double checked my instructions and they work fine. If you're having a problem it means you've made a mistake at 3,4 or 8. – pinoyyid Mar 15 '19 at 1:56

Probably but I couldn't spot the mistake. Non-playground path can be useful if the playground is down/unreachable, too. – m02ph3u5 Mar 15 '19 at 10:10