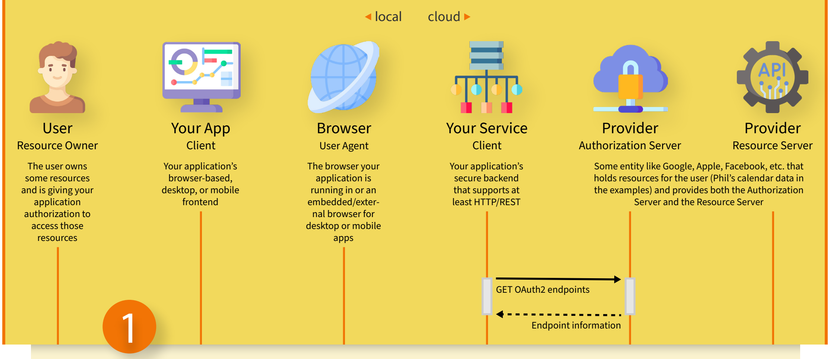
OAuth 2.0 **Authorization Code** grant type

Three-legged OAuth fully visualized



Optionally, get OAuth endpoints from OpenID Connect

Providers that support OpenID Connect must support a well-known discovery endpoint that returns the latest version of their OAuth2 and OpenID endpoints. If your Provider supports this it's recommended you get the most current authorization, token, and other endpoints this way instead of hard coding them.

Example

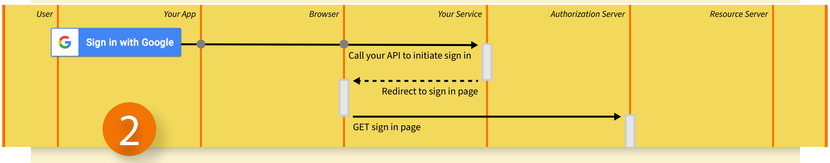
HTTP Request

GET https://accounts.google.com/.well-known/openid-configuration

HTTP Response

200 OK

```
{
  "issuer": "https://accounts.google.com",
  "authorization_endpoint": "https://accounts.google.com/o/oauth2/v2/auth",
  "device_authorization_endpoint": "https://oauth2.googleapis.com/device/code",
  "token_endpoint": "https://oauth2.googleapis.com/token",
  "userinfo_endpoint": "https://openidconnect.googleapis.com/v1/userinfo"
  and so on...
```



Send the Provider an authorization request

The sequence shown is an opinionated version of how to accomplish this. By calling a known API endpoint in your service to generate the authorization URL and then redirecting the browser there, you keep all responsibility for OAuth URL construction in one place and protect the code that generates the **`state`** value. Other options include attaching the URL directly to a sign-in button, generating it using JavaScript, etc.

The URL itself uses the **`authorization_endpoint`** from step ❶ along with various parameters, some of which are standard, and some of which vary depending on the Provider. See your Provider's documentation for construction of this URL. A few common, important ones are discussed below:

`response_type` must be **`code`** to indicate the Authorization Code grant type

`client_id` must be the public client id for your App that you received from the Provider, likely through some web-based developer console.

`state` should be used and be non-guessable to prevent CSRF attacks.

`scope` should be used to describe only the resources your App needs thereby allowing the Provider to present this information to the user for authorization, as well as limiting the blast radius of a stolen token.

Example

HTTP Request

GET https://www.myapp.com/googlelogin

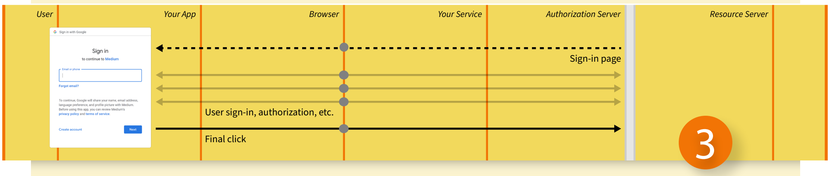
HTTP Response

302 Found

Location: https://accounts.google.com/o/oauth2/v2/auth?
  response_type=code&
  client_id=*your application's client id*&
  redirect_uri=https://www.myapp.com/redirect&
  scope=openid https://www.googleapis.com/auth/calendar.readonly&
  state=9047328547239572 9&
  access_type=offline&
  prompt=consent

HTTP Request

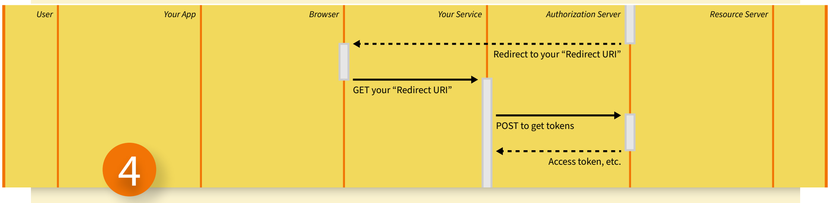
GET *same as Location above*



User signs in and authorizes your App

This process is opaque to your App and usually includes things like sign-in, two-factor authentication, authorization, etc. specific to the Provider.

Note that the user and browser participate in this interaction but your App does not.



Provider sends a code which you turn into token(s)

Once the user completes the Provider's authorization process, the Provider will redirect the browser back to your service using the "Redirect URI" you provided in step ❷ above.

The "Redirect URI" will include at least **`code`** and **`state`** query parameters. The **`state`** parameter should be verified to be the value you passed in step ❷ to prevent CSRF attacks. The **`code`** parameter should be used in a POST along with the private client secret for your App to retrieve an access token and other tokens. Once you have a valid access token it may be used to directly call the Provider's API (Resource Server). The additional tokens received are dependent on the scope parameter in the authorization request and possibly other factors.

The enhanced security of this flow comes from the fact that your service and the Provider have a private interaction (the third leg) not moderated by the browser. Neither your App's client secret nor the resulting tokens are exposed to the browser or any other part of the user's local system thereby preventing even a fully compromised system from obtaining them.

Read your Provider's documentation carefully. There are several valid methods of constructing the POST and each Provider has some nuances. Query parameter-based methods should be avoided if possible (although I use them in the example) to prevent logging of your App's client secret.

Example

HTTP Response

302 Found

Location: https://www.myapp.com/redirect?
  code=*some value*&
  state=9047328547239729

HTTP Request (from browser)

GET *same as Location above*

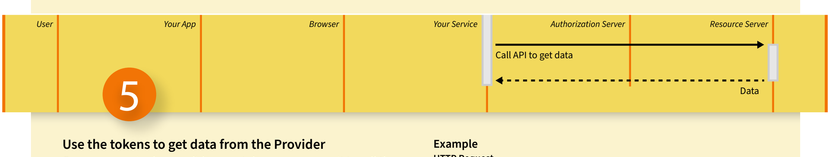
HTTP Request (from your service)

POST https://oauth2.googleapis.com/token?
  grant_type=authorization_code&
  client_id=*your application's client id*&
  client_secret=*your application's client secret*&
  code=*code from above*&
  redirect_uri=https://www.myapp.com/redirect

HTTP Response

200 OK

```
{
  "access_token" : "eyJhbG...",
  "refresh_token" : "OiJSUzI1...",
  "id_token" : "NiIsIng1dS...",
  "token_type" : "bearer",
  "expires_in" : 3599,
  "scope" : "openid https://www.googleapis.com/auth/calendar.readonly"
  and so on...
```



Use the tokens to get data from the Provider

Once your service has a valid access token you may use it to call the Provider APIs (Resource Server) immediately or in the future. The access token may be refreshed using a refresh token and user information may be obtained using an id token. Other tokens may be available for other tasks.

Note that the sequence diagram shows this happening immediately but it could happen at any point in the future as long as the access token is valid.

Example

HTTP Request

GET https://www.googleapis.com/calendar/v3/users/me/calendarList
  Authorization: Bearer *access_token retrieved above*

HTTP Response

200 OK

```
{
  "kind": "calendar#calendarList",
  "etag": "\"p32079o6aq36820o\"",
  "nextSyncToken": "CICk4MrQxoEDEhZwaGlsQpJv8XRyb3NmYW1pbHkuY29t",
  "items": [
  {
    "kind": "calendar#calendarListEntry",
    "etag": "\"1635346027013500 0\"",
    "id": "pqargb3t6kul4i9shs@group.calendar.google.com",
    "summary": "Phil's Calendar",
    "timeZone": "America/Chicago",
    and so on...
```