

OAuth 2.0

Sumário

- Motivação para a *framework* OAuth 2.0
- Fluxos de autorização na *framework* OAuth 2.0
- Exemplo com o Google OAuth 2.0 playground
- Relação com OpenID Connect

Motivação para a *framework* OAuth 2.0

- “A Alice quer encomendar a impressão de algumas fotos que tem no site myphotos.com usando o serviço de impressão do site printonline.com”
- No cenário tradicional, o cliente (printonline.com) teria de se autenticar no servidor de recursos (myphotos.com) usando as credenciais da Alice
 - Problemas: Posse de credenciais, Acesso total às fotos, Impossibilidade de revogar acesso
- A *framework* OAuth 2.0 tem como objectivo delegar autorização condicionada
 - O dono de recurso autoriza o acesso temporário a um conjunto pré-determinado de recursos

Papéis

- Proprietário do recurso (*Resource owner*)
 - Uma entidade capaz de conceder acesso a um recurso protegido
 - Quando o proprietário de recurso é uma pessoa, é referenciado como um *end-user*
- Servidor de recurso (*Resource server*)
 - O servidor que hospeda recursos protegidos, capaz de aceitar e responder a requisições de recursos protegidos usando *access tokens*
- Cliente (*Client*)
 - Uma aplicação que requisita recursos protegidos através do proprietário de recurso e com sua autorização
- Servidor de autorização (*Authorization server*)
 - O servidor que emite *access tokens* ao cliente depois de autenticar com sucesso o proprietário de recurso e obter autorização

Aplicações cliente

- A *framework* OAuth tem como objectivo ser utilizada por diferentes classes de aplicações
 - Aplicações web clássicas, a correr num servidor HTTP
 - Aplicações web a correr maioritariamente no browser com tecnologias JavaScript
 - Aplicações nativas, em particular as aplicações móveis
- Os clientes têm de se registar no servidor de autorização, sendo-lhes atribuído um `client_id`. Para alguns é também atribuído um `client_secret`, usado pelo cliente no servidor de autorização
- Dois tipos de clientes, em função de conseguir guardar ou não o `client_secret`
 - Confidenciais
 - Públicos

Registo de clientes

- Apenas clientes registados podem aceder a recursos controlados pelo servidor de autorização
 - Imposição de limites
 - Auditoria
- A forma como o registo é feito depende do fornecedor
 - Ex: O responsável pelo cliente faz o registo através de um formulário do servidor de recursos

Exemplo de registo de cliente na aplicação GitHub

- <https://github.com/settings/applications/new>

Register a new OAuth application

Application name *

Name should not begin with 'GitHub' or 'Gist'

The full URL to your application homepage.

Application description

Application description is optional

This is displayed to all users of your application.

Authorization callback URL *

Your application's callback URL. Read our [OAuth documentation](#) for more information.

☐ **Enable Device Flow**

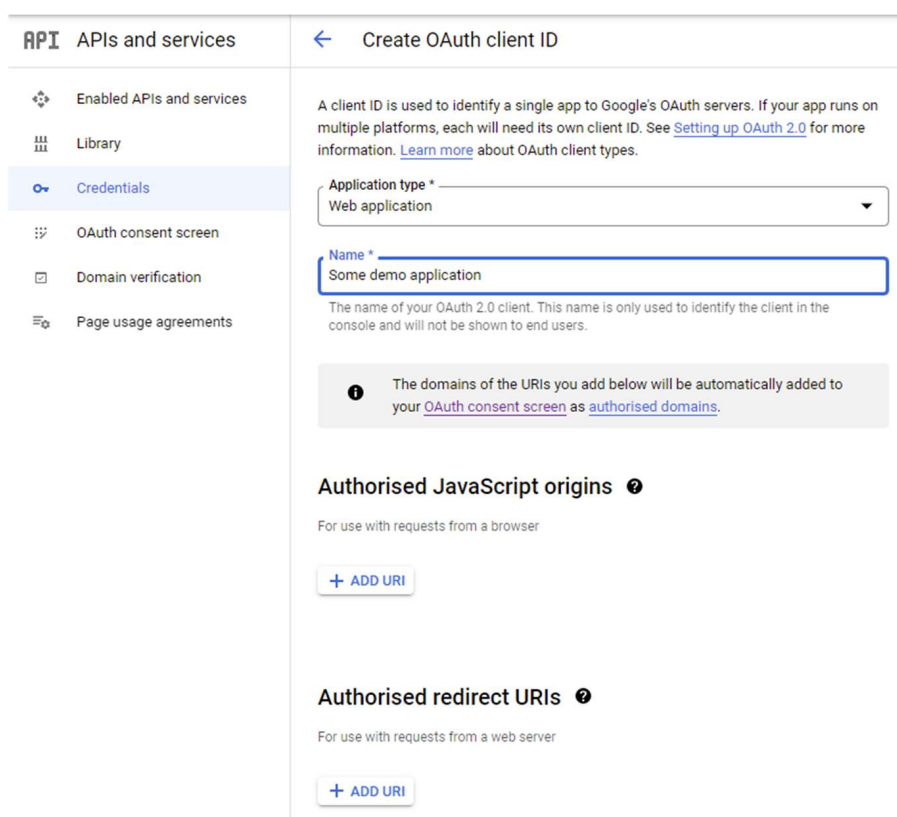
Allow this OAuth App to authorize users via the Device Flow.
Read the [Device Flow documentation](#) for more information.

Register application

Cancel

Exemplo de registo de cliente para acesso a APIs Google

- <https://console.cloud.google.com/apis/credentials>



APIs and services

- Enabled APIs and services
- Library
- Credentials**
- OAuth consent screen
- Domain verification
- Page usage agreements

Create OAuth client ID

A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, each will need its own client ID. See [Setting up OAuth 2.0](#) for more information. [Learn more](#) about OAuth client types.

Application type *
Web application

Name *
Some demo application

The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.

i The domains of the URIs you add below will be automatically added to your [OAuth consent screen](#) as [authorised domains](#).

Authorised JavaScript origins **i**

For use with requests from a browser

+ ADD URI

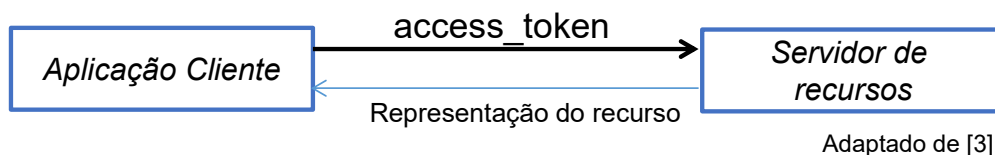
Authorised redirect URIs **i**

For use with requests from a web server

+ ADD URI

Acesso a recursos protegidos (I)

- O *access token* representa uma credencial de acesso
 - Os clientes acedem aos recursos protegidos indicando um *access token*
 - O servidor de recursos usa-o para aplicar políticas de acesso
 - São *strings* opacas para os clientes



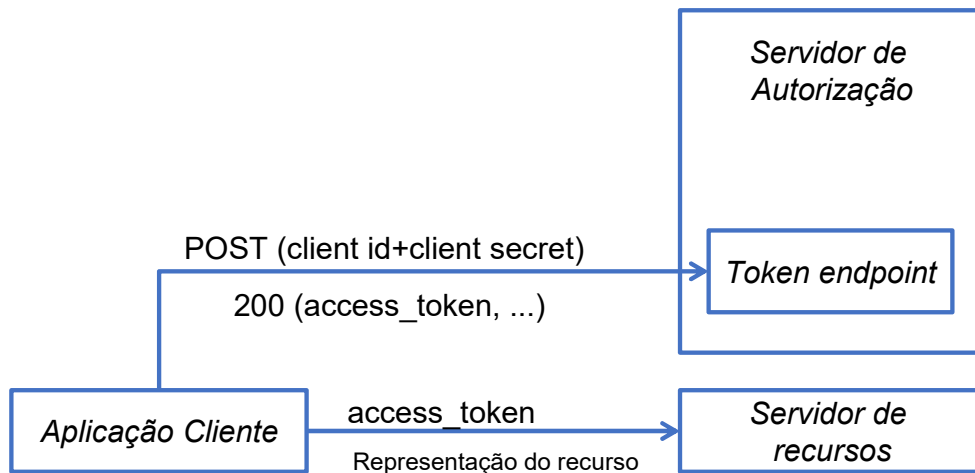
- Exemplo de acesso a um recurso usando autenticação HTTP

```
GET https://si.example.pt/resource HTTP/1.1
Authorization: Bearer the.access.token
```

- A *framework* prevê quatro cenários (*grant flows*) para a obtenção de um *access token*
 - Client Credentials, Resource owner password, Authorization code, Implicit

Acesso a recursos protegidos (II)

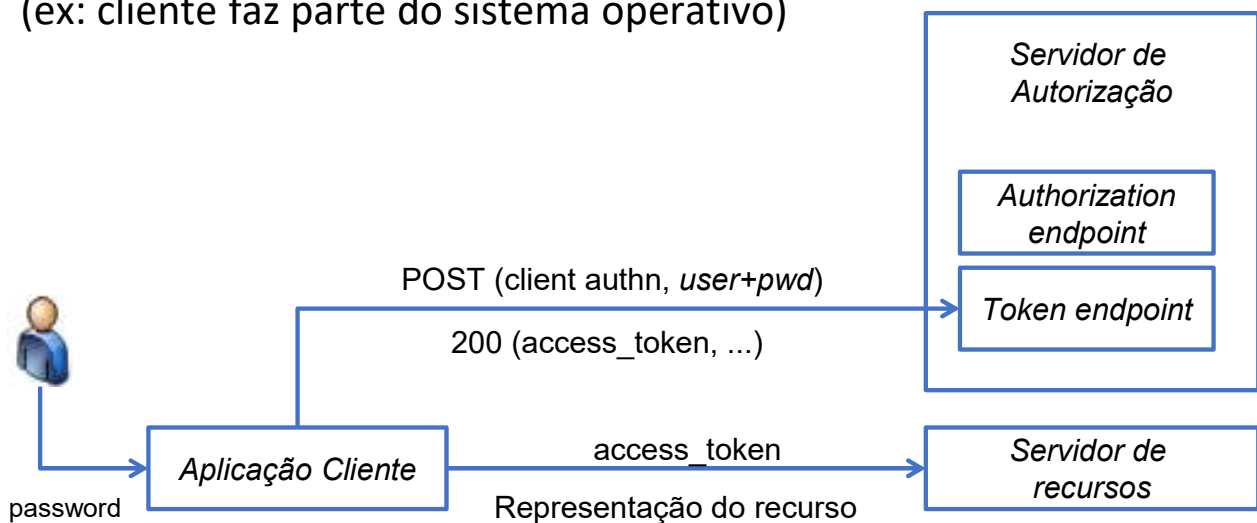
- *Client credentials grant flow* – Autorização é dada somente com base nas credenciais do cliente (client_id e client_secret), ou seja, não está envolvido nenhum utilizador



Adaptado de [3]

Acesso a recursos protegidos (III)

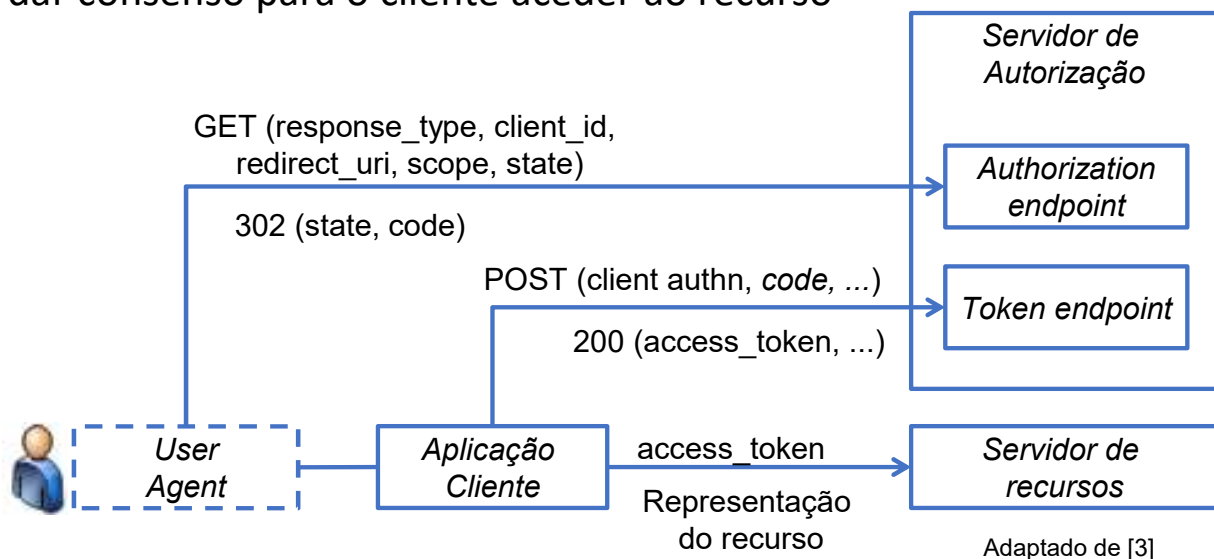
- *Resource owner password credential grant* – Autorização é dada com base na *password* do utilizador
 - A aplicação cliente não precisa de persistir a *password*
 - Deve apenas ser usado quando há um nível elevado de confiança entre o dono de recursos e a aplicação cliente (ex: cliente faz parte do sistema operativo)



Adaptado de [3]

Acesso a recursos protegidos (IV)

- *Authorization code grant* – código de autorização é obtido pelo dono dos recursos e entregue ao cliente para que este obtenha o *access_token*
 - A *password* do dono dos recursos não é visível para o cliente
 - Não define forma do dono de recursos se autenticar e dar consento para o cliente aceder ao recurso



Mensagem para obtenção do `access_token`

- O *access token* é pedido através de um POST para o *token endpoint*.
 - Parâmetros do pedido seguem no corpo de uma mensagem HTTP cujo *content type* é `application/x-www-form-urlencoded`
 - Cada cenário prevê diferentes parametrizações
- Nos clientes confidenciais é usado o esquema *Basic Authentication* do HTTP
 - `client_id` e `client_secret` são o *username* e *password*, respectivamente

Access tokens: exemplos de pedidos

- Pedido com *Resource Owner Password Credentials Grant*

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=password&username=johndoe&password=A3ddj3w
```

- Pedido com *Authorization Code Grant*

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=Sp1xl0BeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

Access tokens: exemplo de resposta

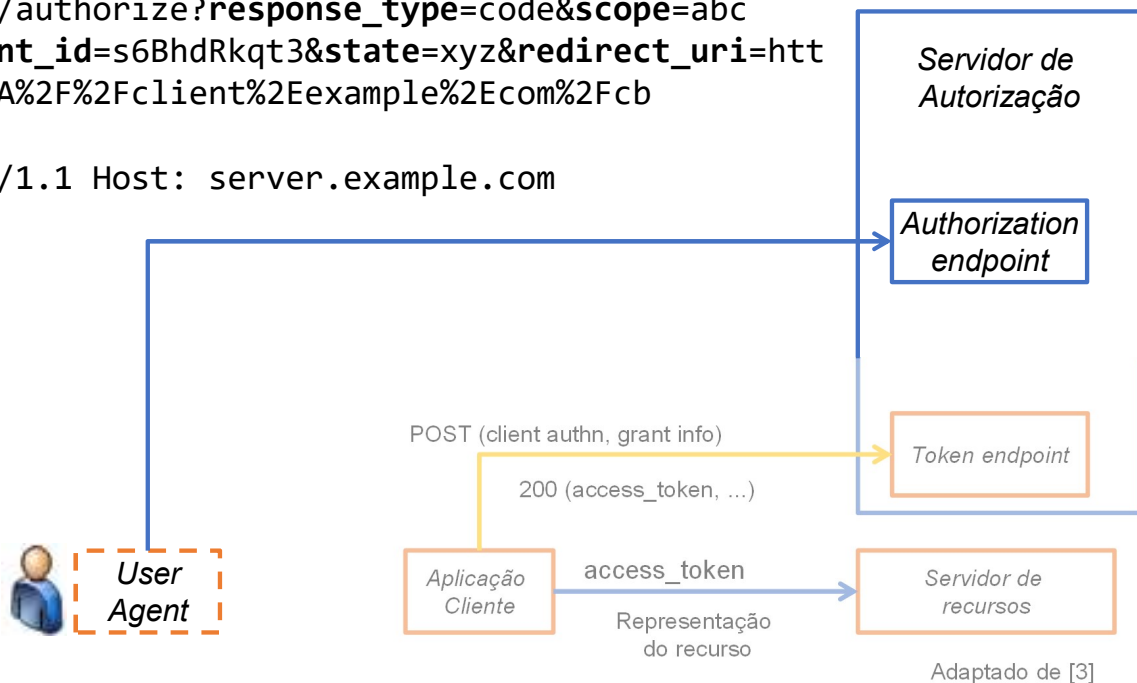
```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
```

```
{
  "access_token": "2YotnFZFEjr1zCsicMWpAA",
  "token_type": "example",
  "expires_in": 3600,
  "refresh_token": "tGzv3J0kF0XG5Qx2TlKWIA",
  "example_parameter": "example_value"
}
```

Detalhes sobre o *Authorization code grant*

GET /authorize?response_type=code&scope=abc
client_id=s6BhdRkqt3&state=xyz&redirect_uri=ht
ps%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb

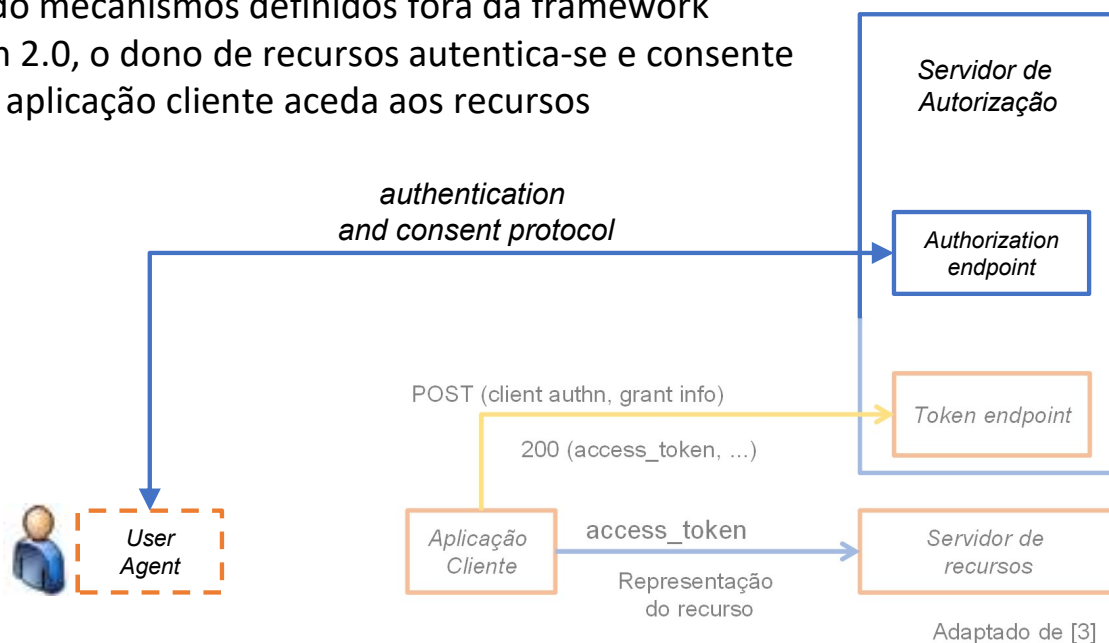
HTTP/1.1 Host: server.example.com



Adaptado de [3]

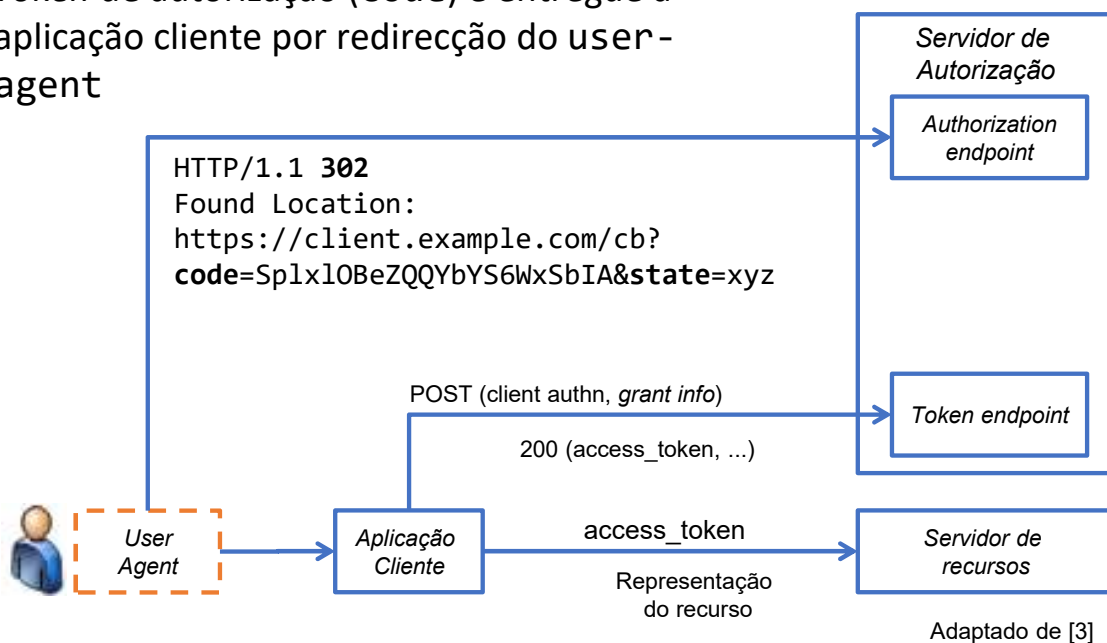
Detalhes sobre o *Authorization code grant*

- Usando mecanismos definidos fora da framework OAuth 2.0, o dono de recursos autentica-se e consente que a aplicação cliente acesse os recursos



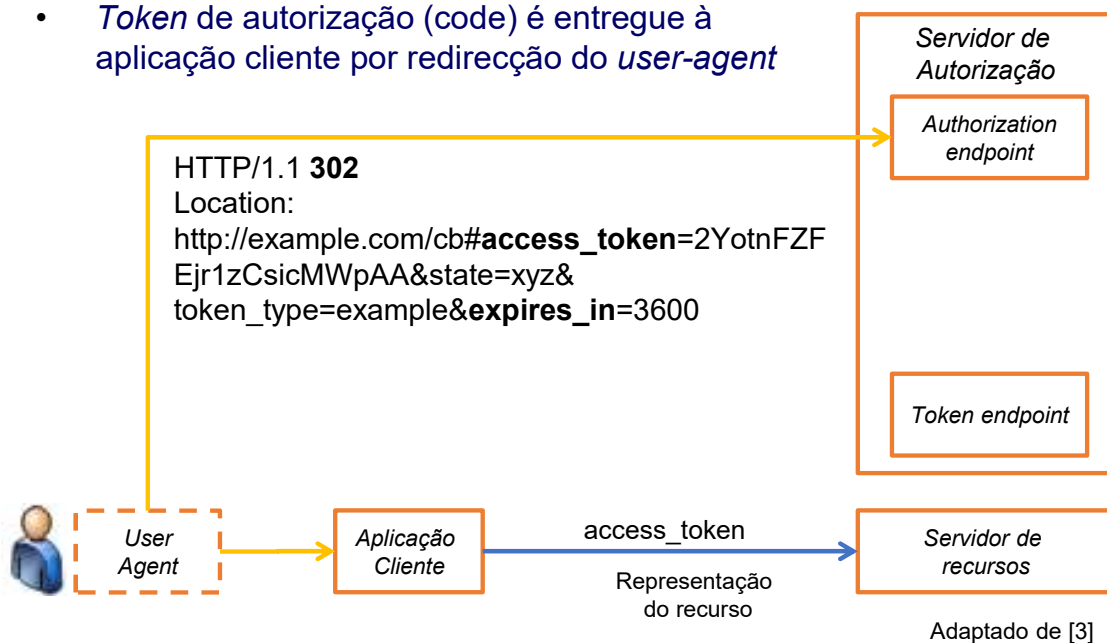
Detalhes sobre o *Authorization code grant*

- *Token* de autorização (code) é entregue à aplicação cliente por redirecção do user-agent



Detalhes sobre o *Implicit grant*

- *Token* de autorização (code) é entregue à aplicação cliente por redirecção do *user-agent*



Scopes

- Representam o tipo de autorização que está a ser pedido a determinado recurso
- Cada *scope* é uma string. Cada pedido de autorização contém zero ou mais *scopes*.
- Exemplos
 - (Google tasks) `https://www.googleapis.com/auth/tasks`
 - (GitHub) `user:email+public_repo`

```
GET /authorize?  
response_type=code&  
client_id=s6BhdRkqt3&  
state=xyz&  
redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb&  
scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Ftasks  
HTTP/1.1 Host: server.example.com
```

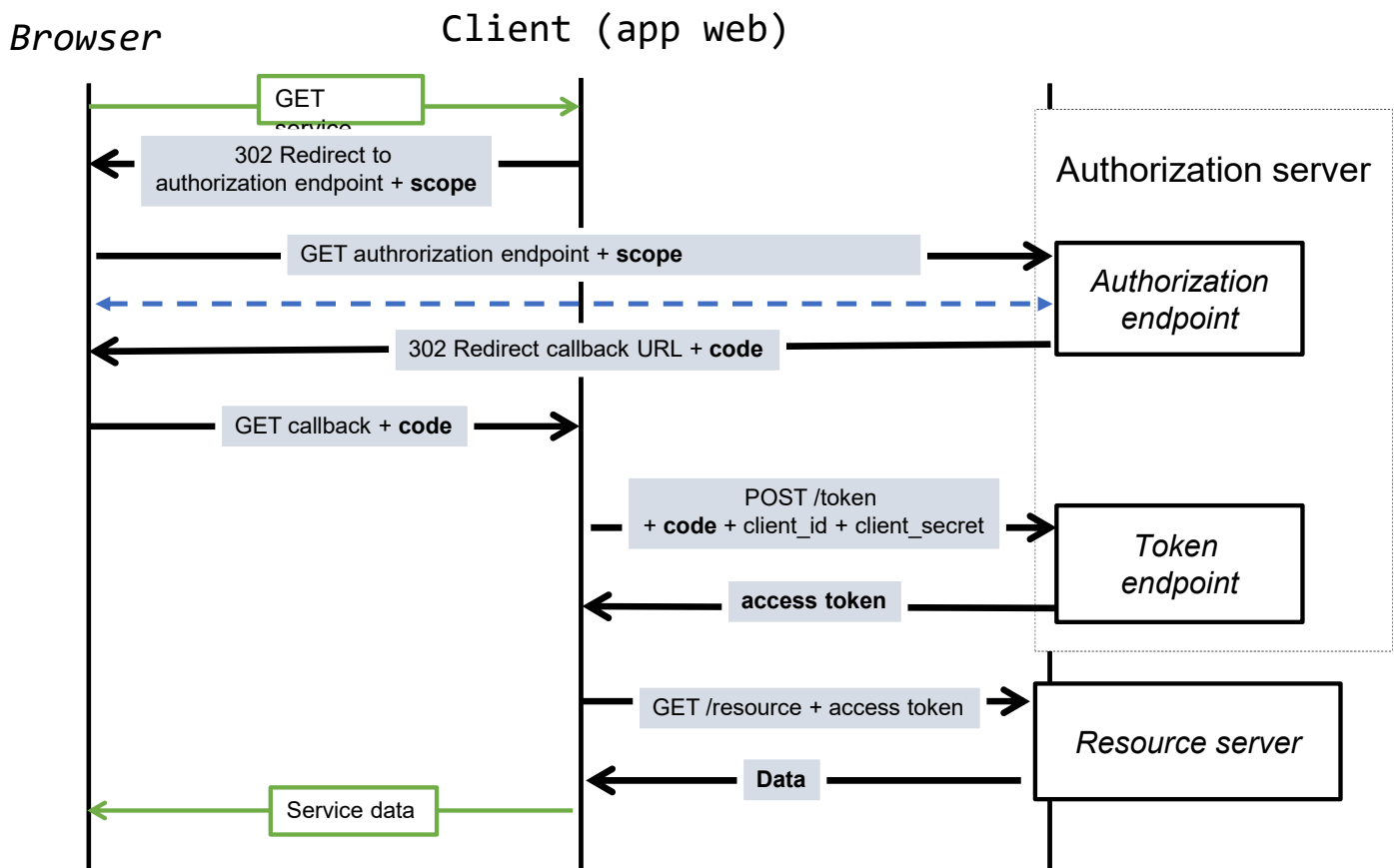
Access token e Refresh token

- O *Access token* representa as credenciais para aceder a recursos protegidos
 - *String* opaca para os clientes, representando *scopes* específicos e duração de acesso
 - É emitido pelo servidor de autorização e validado pelo servidor de recursos. Ambos os servidores de recursos e de autorização precisam de conhecer a estrutura interna do token e o que representa
 - Tem a vantagem do servidor de recursos não precisar de saber lidar com um conjunto diversificado de mecanismos de autenticação
 - Formato, estrutura e métodos de utilização podem variar
- Opcionalmente, o *Authorization Server* entrega também um *refresh token* usado para pedir novas credenciais de acesso

Front channel e Back channel

- *Front channel*
 - Termo usado para designar o canal de comunicação client <-> *Authorization endpoint*, via redirecção do *user-agent*
 - Em caso de erro a resposta tem sempre de ser entregue via *redirect* (não podem ser usados os códigos de erro)
 - O *client_secret* nunca passa pelo *front channel*
 - Usando o *front channel* como é que o cliente estabelece uma relação entre pedido e resposta?
 - Parâmetro *state* (ver [1] secção 10.12)
- *Back channel*
 - Termo usado para designar o canal de comunicação cliente <-> *Token endpoint*
 - Usa mensagem POST HTTP e os respectivos códigos de erro
 - HTTP Basic authentication com *username* (*client_id*) e *password* (*client_secret*)

Authorization code grant: outra visão



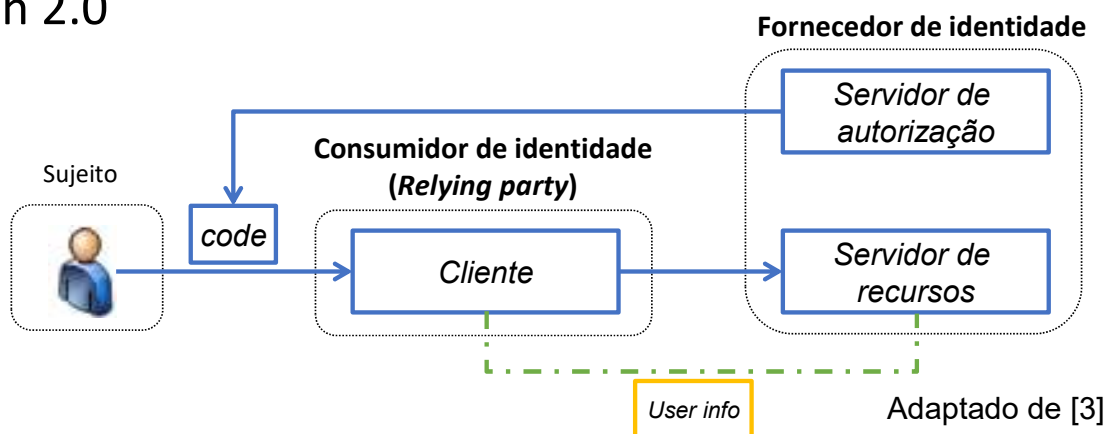
Ameaças e recomendações (alguns exemplos [2])

- Premissas gerais
 - Atacante tem acesso total à rede entre o utilizador e servidor de autorização.
 - Atacante não tem acesso à rede entre servidor de autorização e servidor de Recursos
 - O cliente e o servidor de autorização podem fazer conluio
- Servidor de autorização
 - Rotação de *refresh tokens* para que em caso de perda fiquem inválidos
 - Validação do *redirect_uri* para evitar falsos clientes
- Cliente
 - Não armazenar credenciais no código ou em recursos da aplicação
 - Guardar credenciais (*client secret*) em local seguro
 - Ligar o parâmetro **state** ao *user-agent* que fez o pedido

OPENID CONNECT

OpenID Connect

- O protocolo OpenID Connect (*core specification*) fornece dois serviços [4]
 - Autenticação do utilizador, através de um conjunto assinado de asserções (*id_token*)
 - Acesso a informação adicional sobre o utilizador indicando o *access token*
- O OpenID Connect acrescenta uma camada de identidade ao OAuth 2.0



Obtenção do ID Token pela aplicação cliente

POST /token HTTP/1.1

...

grant_type=authorization_code&code=Sp1x10BeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb

HTTP/1.1 200 OK

Content-Type: application/json

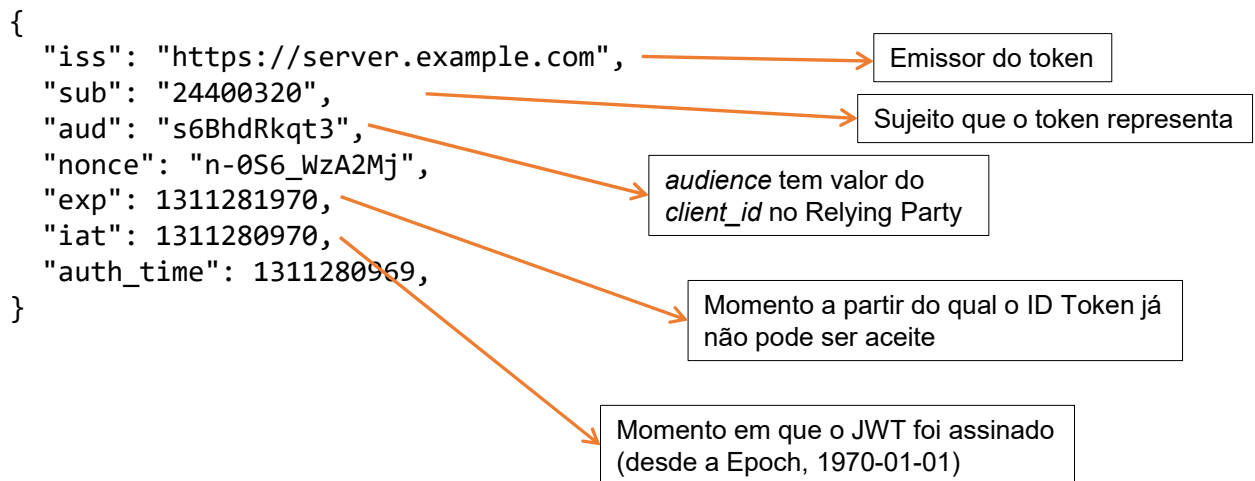
Cache-Control: no-store

Pragma: no-cache

```
{  
  "access_token": "S1AV32hkKG",  
  "token_type": "Bearer",  
  "refresh_token": "8xLOxBtZp8",  
  "expires_in": 3600,  
  "id_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXLTJ5IiwiaWF0IjoxNTE2MzY0MDAifQ.eyJ1IjoiYXNjaWQiLCJ2IjoiZWogImZc...A5NzfQ.XUVr...rvKMzqg"  
}
```

ID Token

- Um ID token é um conjunto de asserções sobre um utilizador autenticado
- JSON Web Token assinado pelo fornecedor de identidade



Recurso UserInfo

- A informação sobre um utilizador autenticado pode ser obtida através do UserInfo Endpoint
- Representada através de um objecto JSON
 - se assinada/cifrada será um JWT [5]
- Exemplo com *UserInfo endpoint*
<https://www.googleapis.com/oauth2/v3/userinfo>

```
{
  "family_name": "Surname",
  "name": "Alice",
  "picture": "...",
  "email": "alice@gmail.com",
  "gender": "female",
  "link": "https://plus.google.com/...",
  "given_name": "Alice",
  "id": "100...2243139"
}
```

Referências

- [1] The OAuth 2.0 Authorization Protocol, RFC 6749 (2012),
<http://tools.ietf.org/html/rfc6749>

- [2] OAuth 2.0 Threat Model and Security Considerations, RFC 6819 (2013) -
<https://tools.ietf.org/html/rfc6819>

- [3] Designing Evolvable Web APIs with ASP.NET (2014), Glenn Block, Pablo Cibrari, Pedro Félix, Howard Dierking, Darrel Miller.

- [4] OpenID Connect, https://openid.net/specs/openid-connect-core-1_0.html (2014)
- [5] JSON Web Token (JWT), <https://tools.ietf.org/html/rfc7519> (2015)