OAuth 2.0

From scratch

What is OAuth 2

- OAuth 2 is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and DigitalOcean.
- It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account.
- OAuth 2 provides authorization flows for web and desktop applications, and mobile devices.

Roles

OAuth defines four roles:

- Resource Owner
- Client
- Resource Server
- Authorization Server

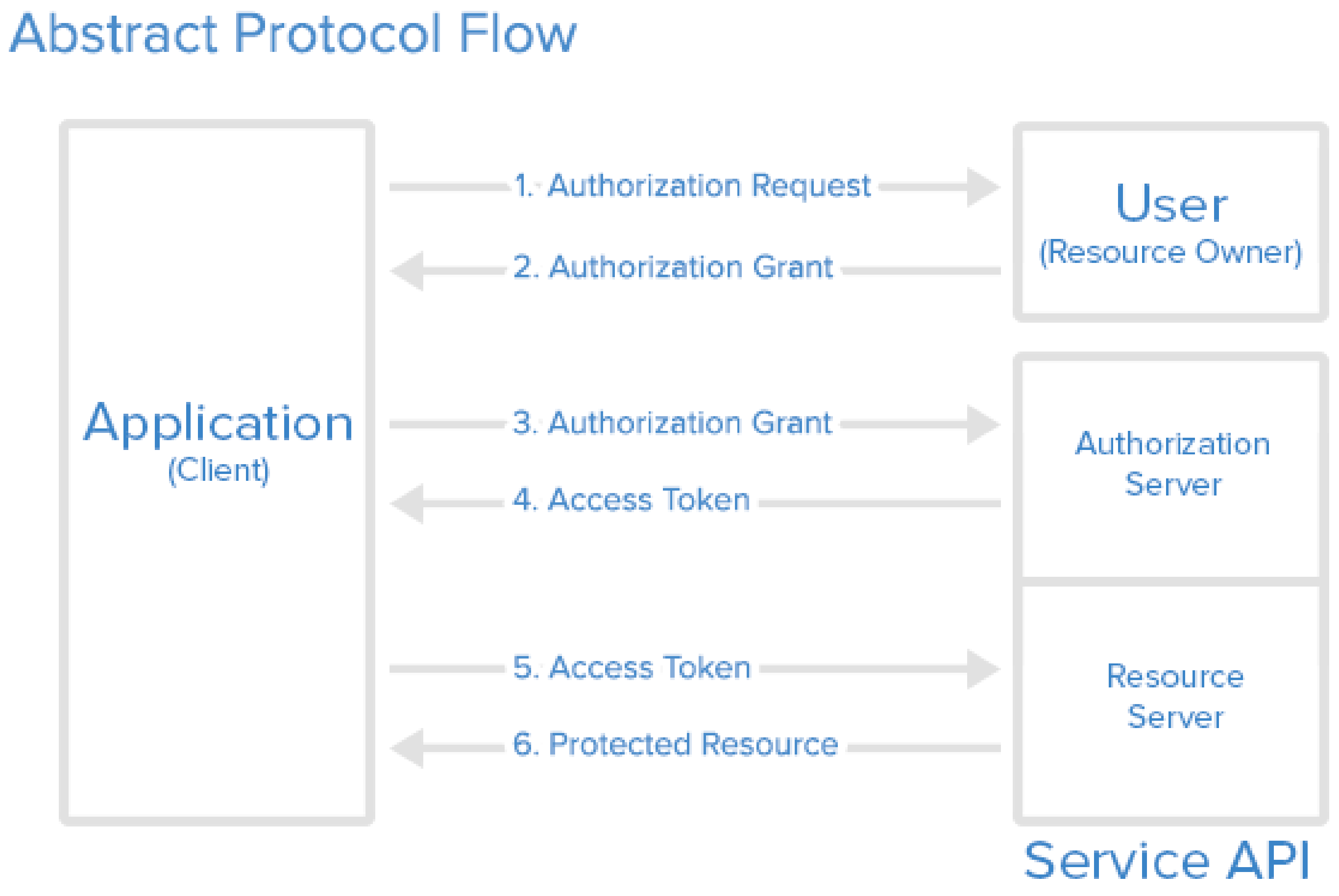
Roles in detail

- Resource Owner: User
  - The resource owner is the user who authorizes an application to access their account. The application’s access to the user’s account is limited to the “scope” of the authorization granted (e.g. read or write access).
- Resource / Authorization Server: API
  - The resource server hosts the protected user accounts, and the authorization server verifies the identity of the user then issues access tokens to the application.

From an application developer’s point of view, a service’s API may fulfil both the resource and authorization server roles. We will refer to both of these roles combined, as the Service /API role.

- Client: Application
  - The client is the application that wants to access the user’s account. Before it may do so, it must be authorized by the user, and the authorization must be validated by the API.

Protocol flow



Step-by-step

- The *application* requests authorization to access service resources from the *user*
- If the *user* authorized the request, the *application* receives an authorization grant
- The *application* requests an access token from the *authorization server* (API) by presenting authentication of its own identity, and the authorization grant
- If the application identity is authenticated and the authorization grant is valid, the *authorization server* (API) issues an access token to the application. Authorization is complete.
- The *application* requests the resource from the *resource server* (API) and presents the access token for authentication
- If the access token is valid, the *resource server* (API) serves the resource to the *application*
    - The actual flow of this process will differ depending on the authorization grant type in use.

Application Registration

- Before using OAuth with your application, you must register your application with the service. This is done through a registration form in the “developer” or “API” portion of the service’s website, where you will provide the following information (and probably details about your application):
    - Application Name
    - Application Website
    - Redirect orCallback URL
- The callback is where the service will redirect the user after they authorize (or deny) your application, and therefore the part of your application that will handle authorization codes or access tokens.

Client Secret

- Once your application is registered, the service will issue “client credentials” in the form of a *client identifier* and a *client secret*.
- The Client ID is a publicly exposed string that is used by the service API to identify the application, and is also used to build authorization URLs that are presented to users.
- The Client Secret is used to authenticate the identity of the application to the service API when the application requests to access a user’s account, and must be kept private between the application and the API.

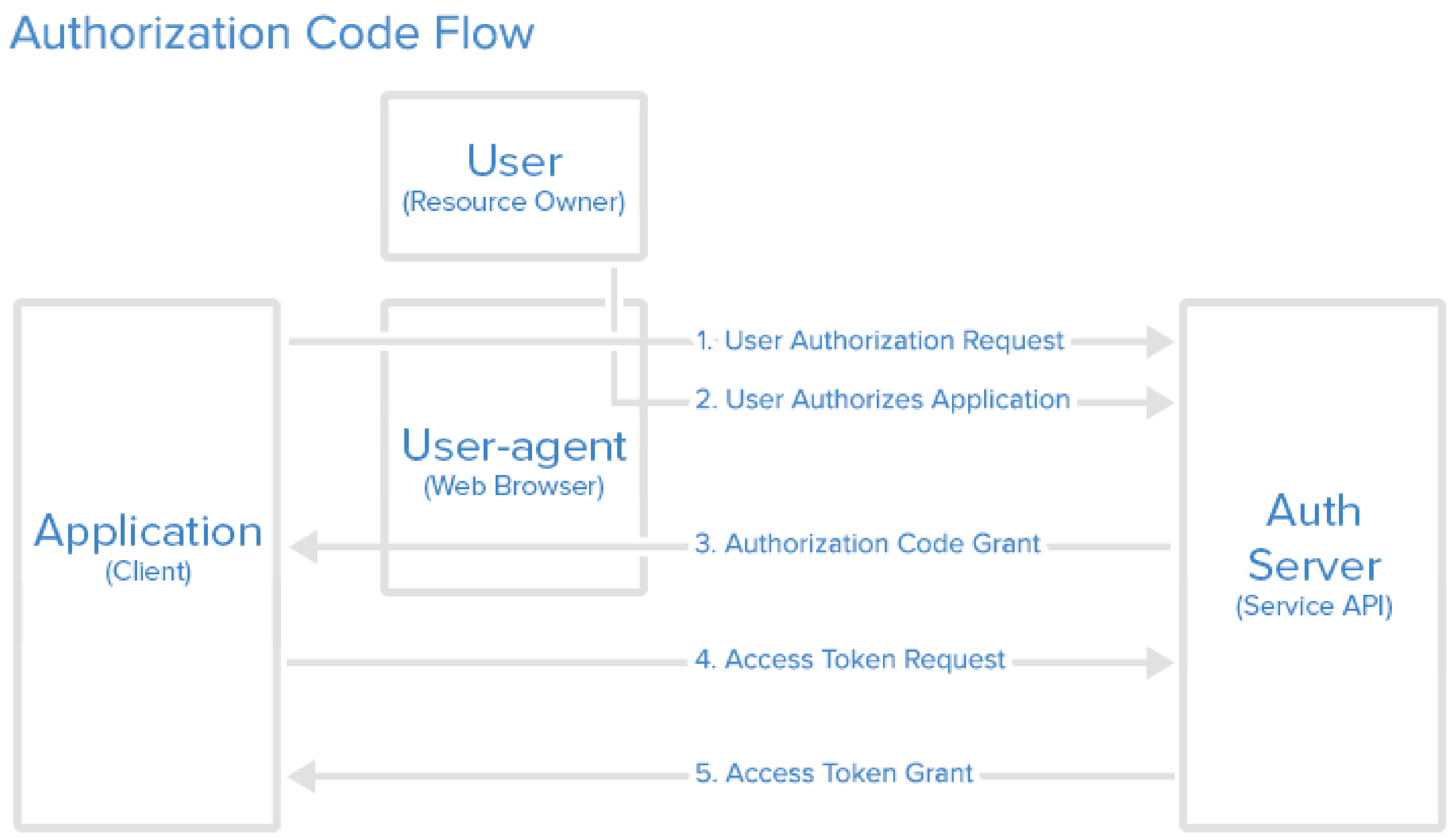
Authorization Grant

- In the *Abstract Protocol Flow* , the first four steps cover obtaining an authorization grant and access token.
- The authorization grant type depends on the method used by the application to request authorization, and on the grant types supported by the API.
- OAuth 2 defines four grant types, each of which is useful in different cases:
  - **Authorization Code**: used with server-side Applications
  - **Implicit**: used with Mobile Apps or Web Applications (applications that run on the user's device)
  - **Resource Owner Password Credentials**: used with trusted Applications, such as those owned by the service itself
  - **Client Credentials**: used with Applications API access

Grant Type: Authorization Code

- The **authorization code** grant type is the most commonly used because it is optimized for *server-side applications*, where source code is not publicly exposed, and *Client Secret* confidentiality can be maintained.
- This is a redirection-based flow, which means that the application must be capable of interacting with the *user-agent* (i.e. the user's web browser) and receiving API authorization codes that are routed through the user-agent.

Grant Type: Authorization Code



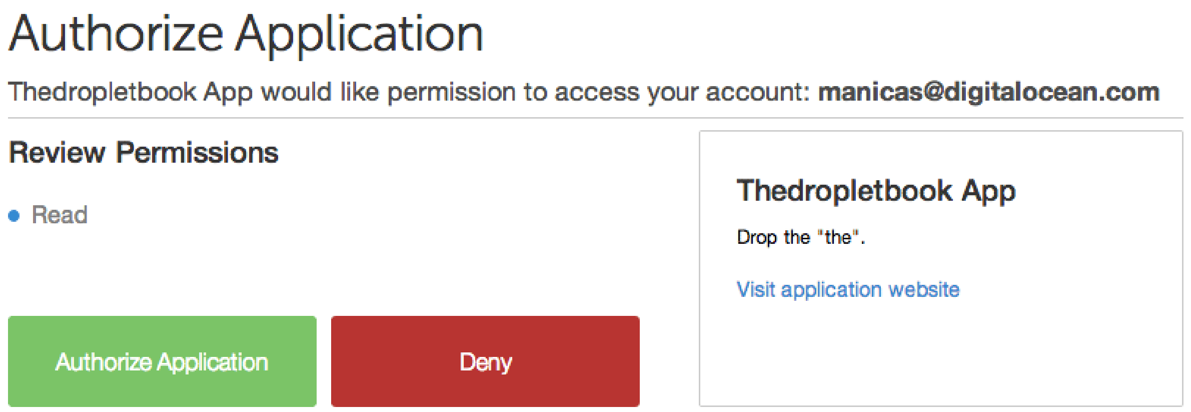
Step 1

- First, the user is given an authorization code link that looks like the following:
- https://cloud.digitalocean.com/v1/oauth/authorize?response_type=code&client_id=CLIENT_ID&redirect_uri=CALLBACK_URL&scope=read
- Here is an explanation of the link components:
  - https://cloud.digitalocean.com/v1/oauth/authorize: the API authorization endpoint
  - client_id=client_id: the application's client ID (how the API identifies the application)
  - redirect_uri=CALLBACK_URL: where the service redirects the user-agent after an authorization code is granted
  - response_type=code: specifies that your application is requesting an authorization code grant
  - scope=read: specifies the level of access that the application is requesting

Step 2

When the user clicks the link, she must first log in to the service, to authenticate her identity (unless she'salready logged in). Then she will be prompted by the service to *authorize* or *deny* the application access to their account.

Here is an example of the authorize application prompt:



Authorize Application

Thedropletbook App would like permission to access your account: **manicas@digitalocean.com**

Review Permissions

- Read

Thedropletbook App

Drop the "the".

Visit application website

Step 3

If the user clicks “Authorize Application”, the service redirects the user-agent to the application redirect URI, which was specified during the client registration, along with an *authorization code*.

The redirect would look something like this (assuming the application is “dropletbook.com”):

```
https://dropletbook.com/callback?code=AUTHORIZATION_CODE
```

Step 4

The application requests an access token from the API, by passing the authorization code along with authentication details, including the *client secret*, to the API token endpoint.

Here is an example HTTP POST request to DigitalOcean’s token endpoint:

```
https://cloud.digitalocean.com/v1/oauth/token?client_id=
CLIENT_ID&client_secret=CLIENT_S
```

Step 5

- If the authorization is valid, the API will send a response containing the access token (and optionally, a refresh token) to the application. The entire response will look something like this:

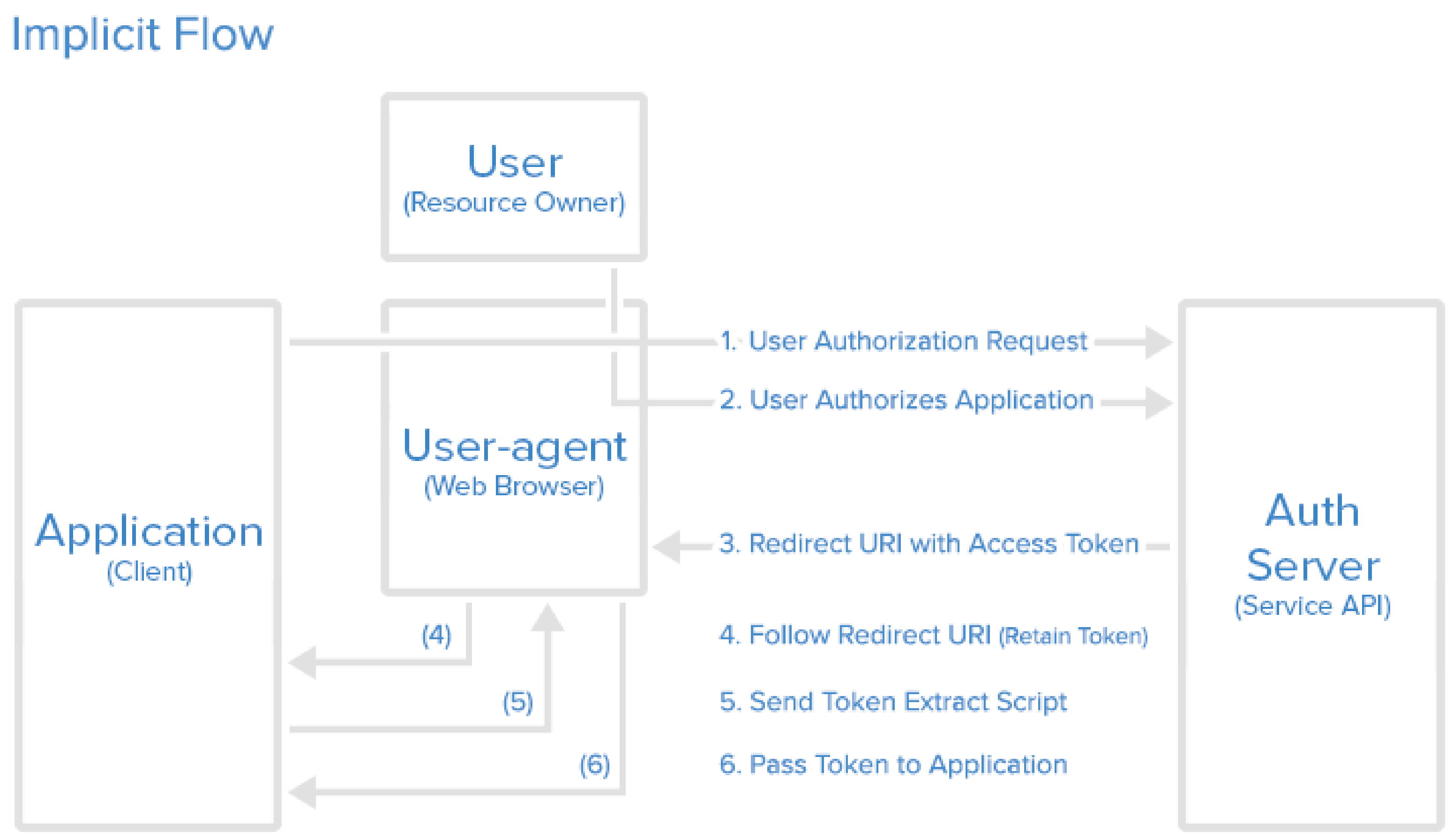
```
{"access_token":"ACCESS_TOKEN","token_type":"bearer","exp
ires_in":2592000,"refresh_token":"REFRESH_TOKEN","scope":
"read","uid":100101,"info":{"name":" E.
Dam","email":"edam@mac.com"}}
```

- Now the application is authorized. It may use the token to access the user’s account via the service API, limited to the scope of access, until the token expires or is revoked.
  - If a refresh token was issued, it may be used to request new access tokens if the original token has expired.

Grant Type: Implicit

- The **implicit** grant type is used for mobile apps and web applications (i.e. applications that run in a web browser), where the *client secret* confidentiality is not guaranteed.
- The implicit grant flow basically works as follows: the user is asked to authorize the application, then the authorization server passes the access token back to the user-agent, which passes it to the application.
- Comments:
  - The implicit grant type is also redirection-based but the access token is given to the user-agent to forward to the application, so it may be exposed to the user and other applications on the user's device.
  - Also, this flow does not authenticate the identity of the application, and relies on the redirect URI (that was registered with the service) to serve this purpose.
  - The implicit grant type does not support refresh tokens.

The flow


Implicit Flow
User
(Resource Owner)
User-agent
(Web Browser)
Application
(Client)
Auth
Server
(Service API)
1. User Authorization Request
2. User Authorizes Application
3. Redirect URI with Access Token
4. Follow Redirect URI (Retain Token)
5. Send Token Extract Script
6. Pass Token to Application
(4)
(5)
(6)

Grant Type: Resource Owner Password Credentials

- With the **resource owner password credentials** grant type, the user provides their service credentials (username and password) directly to the application, which uses the credentials to obtain an access token from the service.
- This grant type should only be enabled on the authorization server if other flows are not viable. Also, it should only be used if the application is trusted by the user (e.g. it is owned by the service, or the user’s desktop OS).

The flow

- After the user gives their credentials to the application, the application will then request an access token from the authorization server. The POST request might look something like this:

- `https://oauth.example.com/token?grant_type=password&username=USERNAME&password=PASSWORD&client_id=CLIENT_ID`

- If the user credentials check out, the authorization server returns an access token to the application.

Grant Type: Client Credentials

- The **client credentials** grant type provides an application a way to access its own service account.
- Examples of when this might be useful include if an application wants to update its registered description or redirect URI, or access other data stored in its service account via the API.

The flow

- The application requests an access token by sending its credentials, its client ID and client secret, to the authorization server. An example POST request might look like the following:
- https://oauth.example.com/token?grant_type=client_credentials&client_id=CLIENT_ID&client_secret=CLIENT_SECRET
-
- If the application credentials check out, the authorization server returns an access token to the application.