

OAuth 2 Grant Types

Open Authorization (OAuth 2) is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service.

Further, OAuth 2 provides authorization flows for web and desktop applications, and mobile devices.

In this post, I will be navigating you all through all different OAuth Grant Types with a story guide.

**OAuth 2 Roles**

* Resource Owner (A.K.A User)
* Client: An application making protected resource requests on behalf of the Resource Owner and with its authorization
* Authorization Server: The server issuing Access Tokens to the Client after successfully authenticating the Resource Owner and obtaining authorization
* Resource Server: The server hosting the protected resources

**OAuth 2 Grant Types**

In total, there are five different grant type flows defined and described to perform authorizations tasks. Those are …

* Authorization Code Grant Type
* Implicit Grant Type
* Resource Owner Credentials Grant Type
* Client Credentials Grant Type
* Refresh Token Grant

**Authorization Code Grant Type**

The Authorization Code Grant Type is the most commonly used grant type to authorize theClient to access protected data from a Resource Server.

Image for post

**The Story …** Athiththan tries to sign-up himself with the *Medium*platform using his Facebook account (because he already has an active FB profile) without totally registering with the platform from scratch. Therefore, he chose the Sign-Up using Facebook option.

*The Medium platform has already been a registered member with the Facebook server and has a unique CLIENT ID and a CLIENT SECRET to introduce and authenticate with Facebook’s Authorization Server.*

So, when Athiththan chose the Sign-Up using Facebook option, the *Medium* sent a GET request with all related data as **Query Parameters** to the Authorization Endpoint.

GET (Authorization Endpoint) ?response\_type=code  
 &client\_id={CLIENT ID}  
 &redirect\_uri={REDIRECT URI}  
 &scope={SCOPES}  
 &state={STATE}

As a response, the Authorization Server sent a temporaryAuthorization Code back to the Redirect URI which was defined in the GET request.

LOCATION : {REDIRECT URI}  
 ?code={AUTHORIZATION CODE}  
 &state={STATE}

The *Medium*stored the temporary Authorization Code and made a POST request to the Token Endpoint which occupied on the Authorization Server.

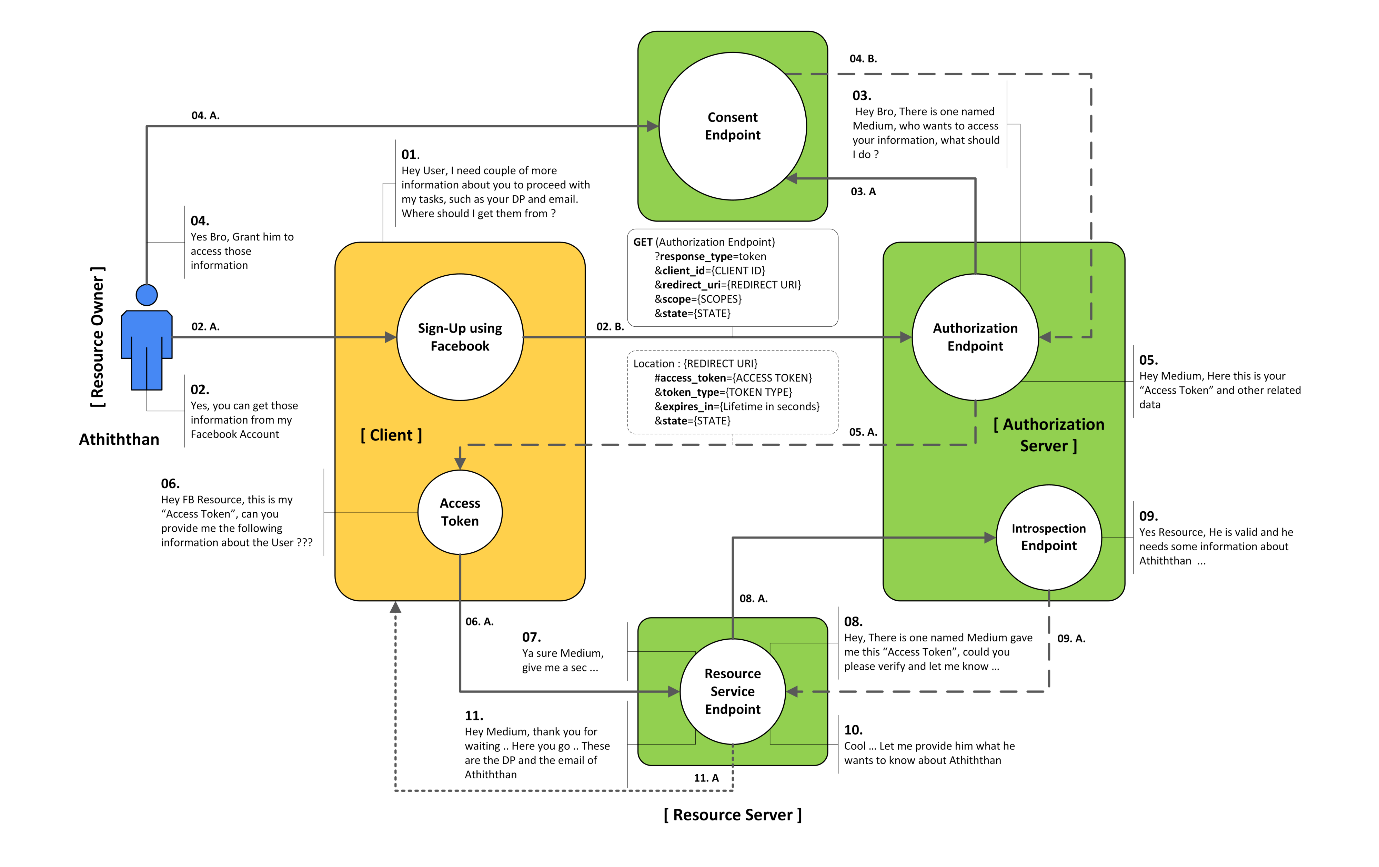
POST (Token Endpoint)  
Authorization : Basic <BASE64({CLIENT ID}:{CLIENT SECRET})> grant\_type=authorization\_code  
 &code={AUTHORIZATION CODE}  
 &redirect\_uri={REDIRECT URI}

As a response, the Authorization Server returned a JSON response embedding the access\_token and other related data.

{  
 "access\_token" : {ACCESS TOKEN},  
 "token\_type" : {TOKEN TYPE >> Bearer},  
 "refresh\_token" : {REFRESH TOKEN},  
 "expires\_in" : {Lifetime in seconds}  
}

**Implicit Grant Type**

The Implicit Grant Type is intended to be used by user-agent based clients (Example: SPA), which can’t keep a client secret because all of the application code and storage is easily accessible.



**The Story …**Athiththan tries to sign-up himself with the *Medium*platform using his Facebook account (because he already has an active FB profile) without totally registering with the platform from scratch. Therefore, he chose the Sign-Up using Facebook option. [*blah blah blah …*]

*The Medium platform has already been a registered member with the Facebook server and has a unique*CLIENT ID*and a*CLIENT SECRET*to introduce and authenticate with*Facebook’s Authorization Server*.*

So, when Athiththan chose the Sign-Up using Facebook option, the *Medium* transmitted a GET request with all related data as **Query Parameters** to the Authorization Endpoint.

\*\* But this time the *Medium*wants to get the access token straightaway without acquiring a temporary Authorization Code in order to provide and retrieve an access token. (Assumption: the *Medium*page is a Single Page Application (SPA))

GET (Authorization Endpoint)  
 ?response\_type=token  
 &redirect\_uri={REDIRECT URI}  
 &scope={SCOPES}  
 &state={STATE}

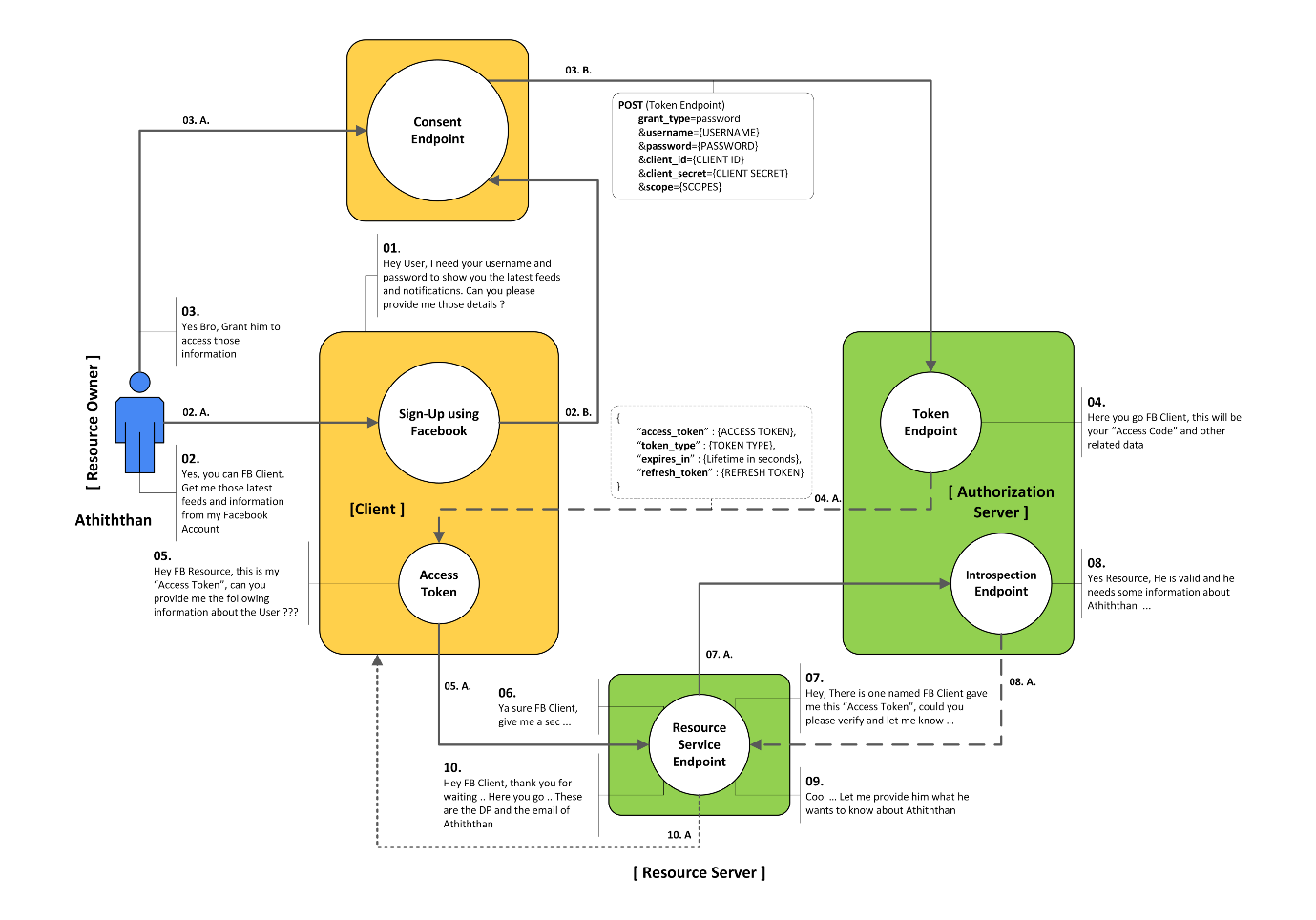
As a response, the Authorization Server shipped back the actualAccess Token back to the Redirect URI which was defined in the GET request as **Query Parameters**.

LOCATION : {REDIRECT URI}  
 #access\_token={ACCESS TOKEN}  
 &token\_type={TOKEN TYPE >> Bearer}  
 &expires\_in={EXPIRES IN}  
 &state={STATE}

**Resource Owner Credentials Grant Type**

The Resource Owner Credentials Grant Type uses the username and the password credentials of a Resource Owner (user) to authorize and access protected data from a Resource Server.

This grant is great for trusted first party clients on both the web and in native device applications.



**The Story …**Athiththan got annoyed trying signing-up with the *Medium*platform using his Facebook account. So, he started downloading the native Facebook application on his smartphone and tried to log-in using his Facebook username and password credentials.

*The Facebook Client application has already been a registered member (First Party & trusted) with the Facebook server and has a unique*CLIENT ID*and a*CLIENT SECRET*to introduce and authenticate with*Facebook’s Authorization Server*.*

So, when Athiththan chose to log-in with Facebook, the application displayed an embedded Consent Page to Athiththan (Resource Owner / User) to sign-in and to give permissions to the *Facebook Client*to access related protected resources from the Resource Server. When he logged in using his username & password of the Facebook profile, the *Facebook Client application*mailed a POST request to the Token Endpoint of the Facebook's Authorization Server as below …

POST (Token Endpoint)  
Authorization : Basic <BASE64({CLIENT ID}:{CLIENT SECRET})> grant\_type=password  
 &username={USERNAME}  
 &password={PASSWORD}  
 &scope={SCOPES}

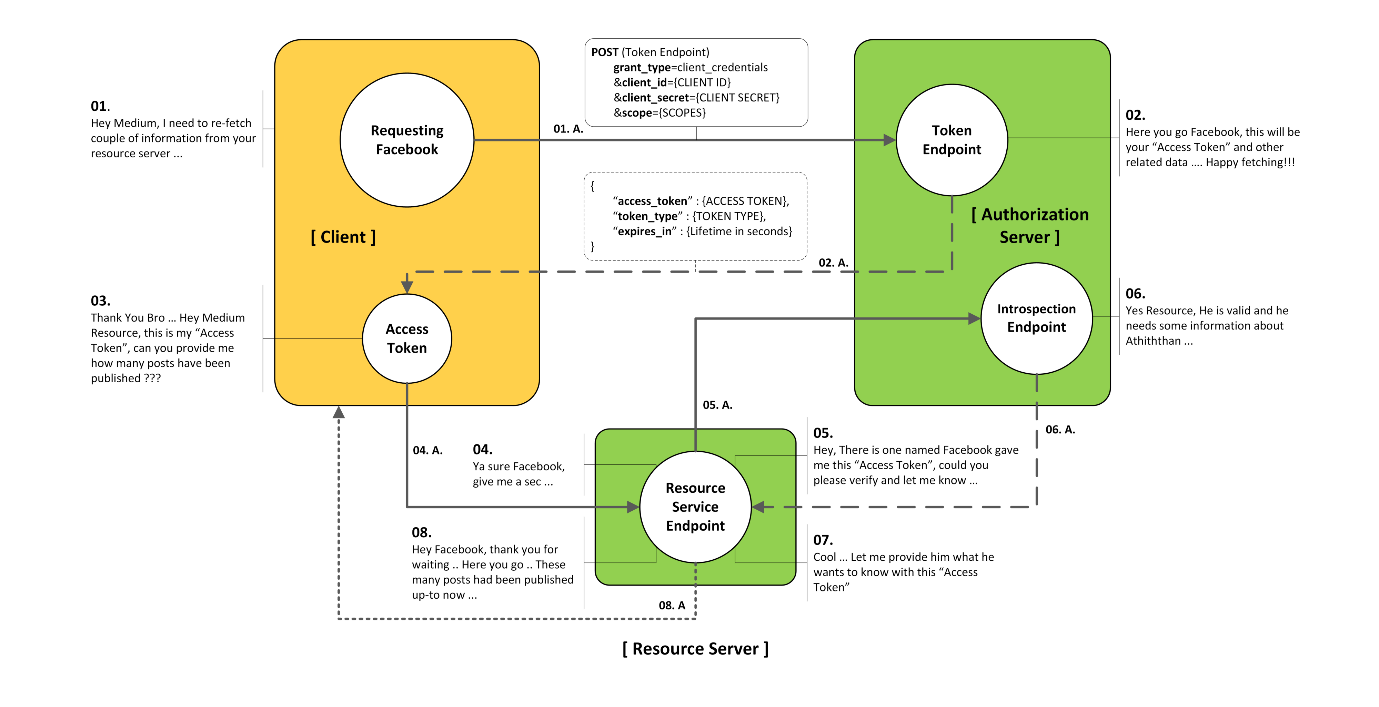
As a response, the Authorization Server delivered a JSON embedding the access\_token and other related data.

{  
 "access\_token" : {ACCESS TOKEN},  
 "token\_type" : {TOKEN TYPE >> Bearer},  
 "expires\_in" : {Lifetime in seconds},  
 "refresh\_token" : {REFRESH TOKEN}  
}

**Client Credentials Grant Type**

The Client Credentials Grant Type uses the client\_id and the client\_secret credentials of a Client to authorize and access protected data from a Resource Server.

This grant flow is suitable for machine-to-machine authentication where a specific user’s permission to access data is not required.



**The Story …**Athiththan finally got himself signed up successfully with the *Facebook*platform using its native application. Assume that the Facebook application has a new feature to fetch the number of posts published in the Medium platform without any required permissions.

*Facebook has already been a registered member of the Medium server, and has a unique CLIENT ID and a CLIENT SECRET to introduce and authenticate with Medium's Authorization Server.*

So, after a successful attempt of logging in to the *Facebook*account using the native application, the *Facebook*application itself started to communicate with the Medium's Authorization Server (without any consent) by sending a POST request to the Token Endpoint as below …

POST (Token Endpoint)  
Authorization : Basic <BASE64({CLIENT ID}:{CLIENT\_SECRET})> grant\_type=client\_credentials  
 &scope={SCOPES}

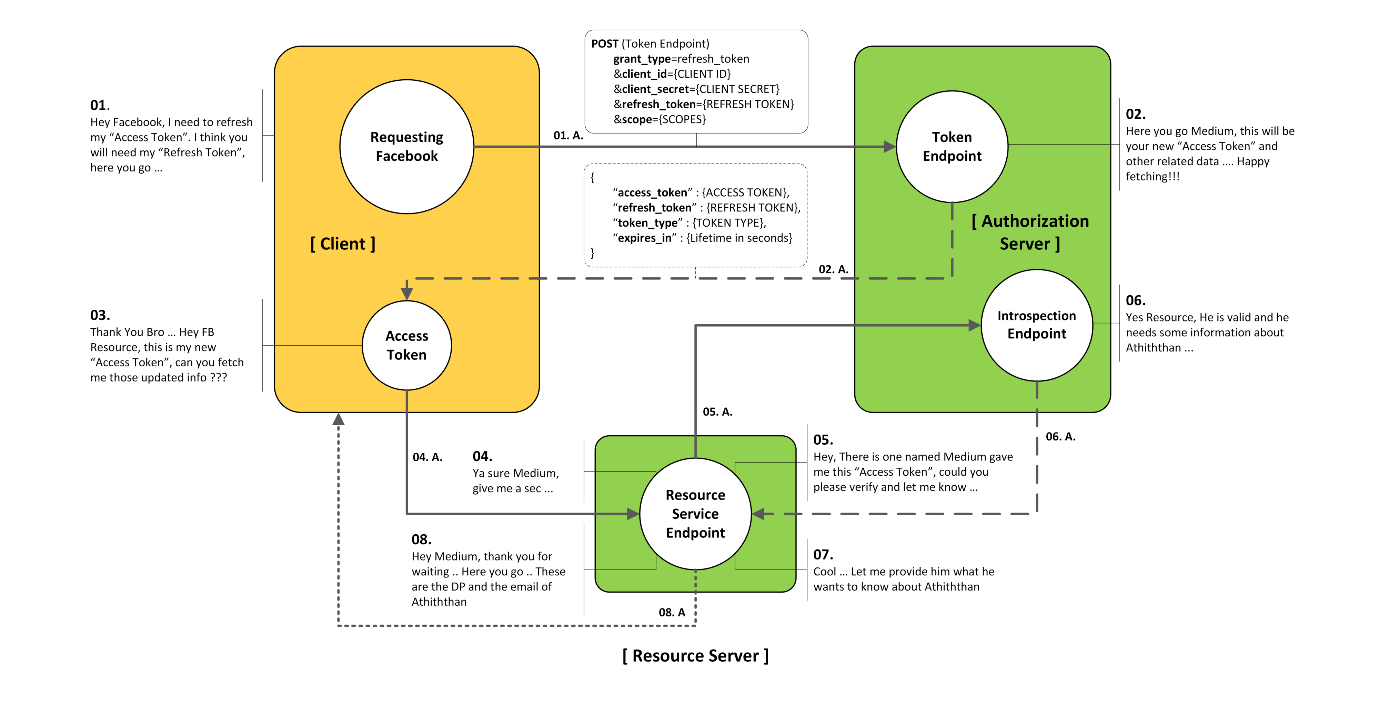
As a response, the Authorization Server conveyed a JSON embedding the access\_token and other related data.

\*\* But, this time, the response will not contain any refresh\_token, which used to gain new access\_token from the Authorization Server.

{  
 "access\_token" : {ACCESS TOKEN},  
 "expires\_in" : {Lifetime in seconds},  
 "token\_type" : {TOKEN TYPE >> Bearer}  
}

**Refresh Token Grant**

The Refresh Token Grant Flow is specially used to gain new access\_token from the Authorization Server by providing the refresh\_token to the Token Endpoint.



**The Story …**(If you can remember, Athiththan got into *Facebook*successfully using his username and password via its smartphone native application. After that scene and he spent around 15 minutes on browsing and viewing newer feeds and stories.) After a day, Athiththan took his smartphone and opened the Facebook's native application to browse through newer feeds.

The native application sent a request to the Resource Server to acquire all newer feeds and stories to display in front. But, at that time, the access token which used to make the request got expired and the Resource Server blocked itself from responding back to the native application.

Athiththan is waiting and starring at the native application to view the newer feeds, at the same time the native application got a false response because of its expired access token. Therefore, the native application made a request to the Token Endpoint conveying the acquired refresh token with its request as below …

POST (Token Endpoint)  
Authorization : Basic <BASE64({CLIENT ID}:{CLIENT SECRET})> grant\_type=refresh\_token  
 refresh\_token={REFRESH TOKEN}  
 scope={SCOPES}

As a response, the Authorization Server responded back with a JSON embedding a newaccess\_token and other tokens to get verified with the Resource Server and to peruse data.

{  
 "access\_token" : {ACCESS TOKEN},  
 "refresh\_token" : {REFRESH TOKEN},  
 "expires\_in" : {Lifetime in seconds},  
 "token\_type" : {TOKEN TYPE >> Bearer}  
}