**Oauth2 Grant Types**

There are four type of oauth2 grant type which are given below

OAuth (Open Authorization) is a simple way to publish and interact with protected data.

It is an open standard for token-based authentication and authorization on the Internet. It allows an end user's account information to be used by third-party services, such as Facebook, without exposing the user's password.

When using OAuth2, grant type is the way an application gets the access token. Following are the grant types according to OAuth2 specification-

1: Authorization Code

2: Client Credential

3: Client Credential with Resource Owner Password (password)

4: Implicit

**1: Authorization Code** -

* 1. Getting The Authorization Code :

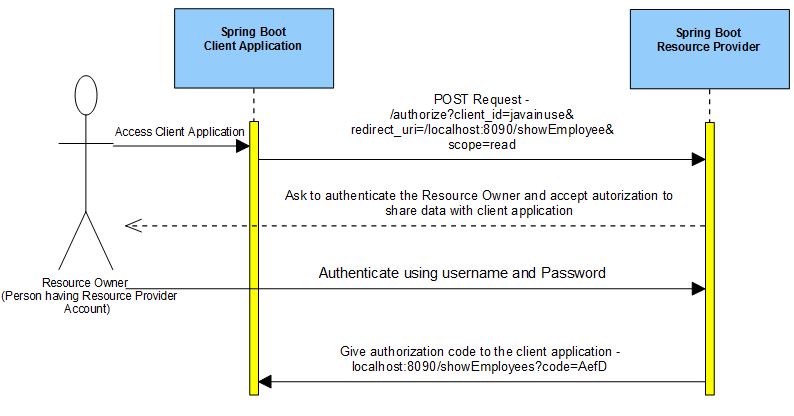
We will be implementing the Client Application and the Resource Server. The flow we will be implementing is as follows -

The Resource Owner will ask the Client Application to get some data from the Resource Server.

The Resource Server asks the Resource Owner to authenticate itself and as for authorization to share data.

After successful authentication the Resource Server shares an authorization code with the client application.

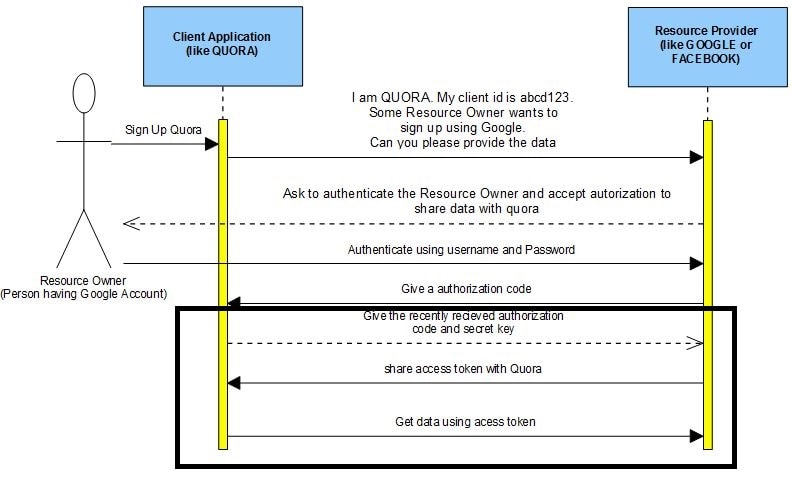
In the given diagram we will see how the client get authorization code and later use that code to get access key and use that key to accessing restricted resource.



The Client Application using the Authorization code and Secret key ask for the Access Token from the Resource Server.

The Resource Server shares the Access Token with the Client Application.

Using the shared Access Token the Client Application can now get the required JSON data from the Resource Server.



**2 : Client Credentials -**

The Client Credentials Grant involves machine to machine authentication. Oauth usually consists of following actors -

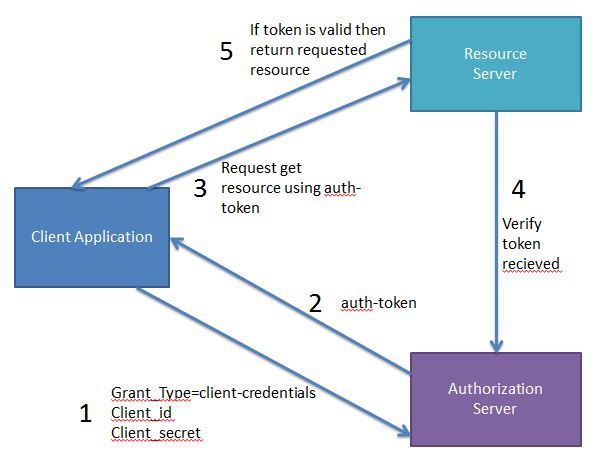
Resource Owner(User) - An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

Client Application - The machine that needs to be authenticated.

Authorization Server - The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization

Resource Server - The resource server is the OAuth 2.0 term for your API server. The resource server handles authenticated requests after the application has obtained an access token.

In case of Client credentials grant type the user has no role to play. As previously stated it is machine to machine communication. This is typically used by clients to access resources about themselves rather than to access a user's resources.



This type of Authentication does not involve any end-user. Unlike Authorization Grant where the end user had to authenticate himself using Authorization Server like Gmail, here the machine it self authenticates itself to access a protected resource.

Trivago server will be accessing several third party APIs to show search results. Machine to machine authentication will be done by the Trivago server to access the third party API's to get the hotel data. Suppose it wants search data from makemytrip.com, so Trivago Server will authenticate itself by calling makemytrip's authorization server to get access token and then using this token access the makemytrip resource server to get the search result. So here-

Client Application(Trivago Server) - Trivago Server which will need to get some reources from MakeMyTrip.com.

Authorization Server(MakeMyTrip Authorization Server)- MakeMyTrip Authorization Server. Here Trivago should have already registered itself to the MakeMyTrip Authorization Server so that it can be authenticated and issued token.

Resource Server(MakeMyTrip Resource Server) - MakeMyTrip application will then use the token it recieved from the Authorization Server to get resource from the MakeMyTrip Resource Server.

MakeMyTrip ResourceServer will verify if the token recieved is valid by calling the Authorization server which issued it. If its valif it will return the requested resource

So 2 calls are required to be made by the client application to get the resource-

Call to the Authorization Server to get the token.

Parameter Value

grant\_type (required) client\_credentials

client\_id(required) The client id

client\_secret(required) The client secret key

After getting the token from the authorization server, the client application then needs to use this for getting resource from the resource server.

1. **: Password** –

In Password grant Oauth usually consists of following actors –

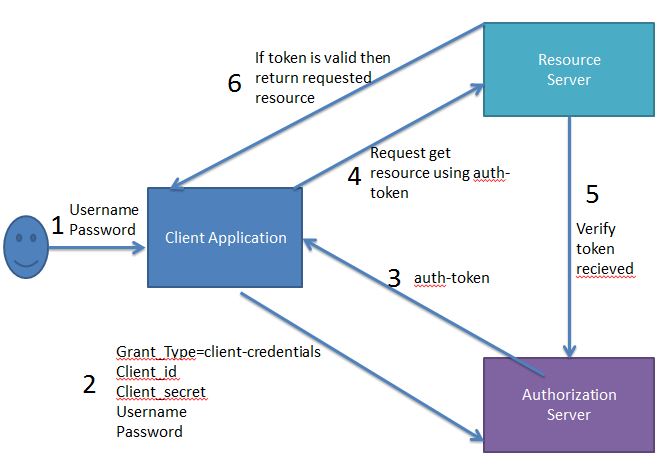
Resource Owner(User) - An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

Client Application - The machine that needs to be authenticated.

Authorization Server - The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization

Resource Server - The resource server is the OAuth 2.0 term for your API server. The resource server handles authenticated requests after the application has obtained an access token.

In case of Password grant type the user triggers the client to get some resource. While doing so it passes the username and password to the client. The client then communicates with the authorization server using the provided username, password and also its own clientId and clientSecret to get the access token. Using this access token it then gets the required resource from the resource server.



So 2 calls are required to be made by the client application to get the resource-

Call to the Authorization Server to get the token.

Parameter Value

grant\_type (required) client\_credentials

client\_id(required) The client id

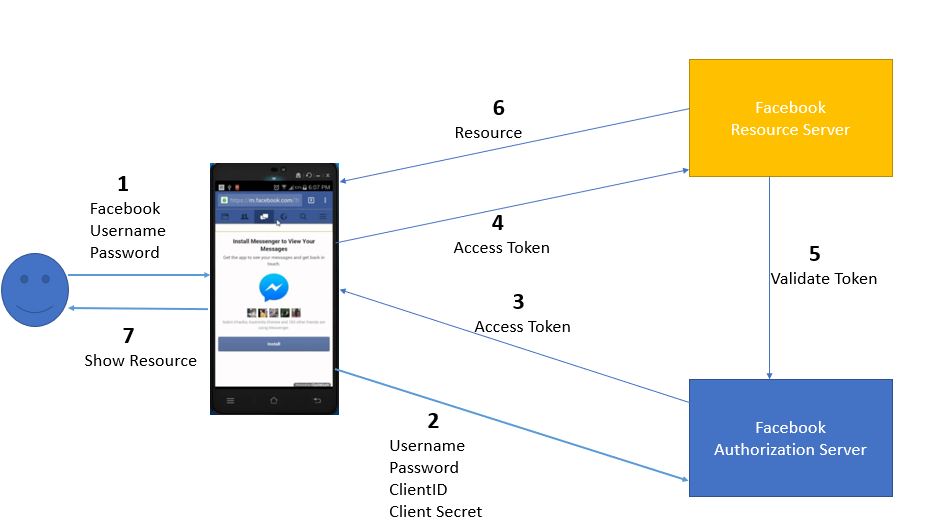
client\_secret(required) The client secret key

username(required) The username of the user

password(required) The password of the user

After getting the token from the authorization server, the client application then needs to use this for getting resource from the resource server.

The real life example of Password grant will be you doing a login to you facebook account using its mobile application. Here the user will have to specify the facebook credentials to the app. Also the app will be having its own client id and client secret.



1. **: Implicit Grant –**

The Implicit Grant Type is a way for a single-page JavaScript app to get an access token without an intermediate code exchange step. It was originally created for use by JavaScript apps (which don’t have a way to safely store secrets) but is only recommended in specific situations.

The Implicit Grant starts out by building a link and directing the user’s browser to that URL. At a high level, the flow has the following steps:

The application opens a browser to send the user to the OAuth server

The user sees the authorization prompt and approves the app’s request

The user is redirected back to the application with an access token in the URL fragment.

**Get The User Permission**

OAuth is all about enabling users to grant limited access to applications. The application first needs to decide which permissions it is requesting, then send the user to a browser to get their permission. To begin the Implicit flow, the application constructs a URL like the following and directs the browser to that URL.

https://authorization-server.com/auth

?response\_type=token

&client\_id=29352910282374239857

&redirect\_uri=https%3A%2F%2Fexample-app.com%2Fcallback

&scope=create+delete

&state=xcoiv98y3md22vwsuye3kch

Here’s each query parameter explained:

response\_type=token - This tells the authorization server that the application is initiating the Implicit flow. Note the difference from the Authorization Code flow where this value is set to code.

client\_id - The public identifier for the application, obtained when the developer first registered the application.

redirect\_uri - Tells the authorization server where to send the user back to after they approve the request.

scope - One or more space-separated strings indicating which permissions the application is requesting. The specific OAuth API you’re using will define the scopes that it supports.

state - The application generates a random string and includes it in the request. It should then check that the same value is returned after the user authorizes the app. This is used to prevent CSRF attacks.

When the user visits this URL, the authorization server will present them with a prompt asking if they would like to authorize this application’s request.

**Redirect Back to the Application**

If the user approves the request, the authorization server will redirect the browser back to the redirect\_uri specified by the application, adding a token and state to the fragment part of the URL.

For example, the user will be redirected back to a URL such as

https://example-app.com/redirect

#access\_token=g0ZGZmNj4mOWIjNTk2Pw1Tk4ZTYyZGI3

&token\_type=Bearer

&expires\_in=600

&state=xcoVv98y2kd44vuqwye3kcq

Note the two major differences between this and the Authorization Code flow: the access token is returned instead of the temporary code, and both values are returned in the URL fragment (after the #) instead of in the query string. By doing this, the server ensures that the app will be able to access the value from the URL, but the browser won’t send the access token in the HTTP request back to the server.

The state value will be the same value that the application initially set in the request. The application is expected to check that the state in the redirect matches the state it originally set. This protects against CSRF and other related attacks.

The server will also indicate the lifetime of the access token before it expires. This is usually a very short amount of time, along the lines of 5 to 10 minutes, because of the additional risk in returning the token in the URL itself.

This token is ready to go! There is no additional step before the app can start using it!