# **OAuth – Open Authorization**

SAML is good which gives us Single sign on but the issue is it is across only 1 data center ,

Ex:- when student completed a course on Udemy and wants to publish a certificate on LinkedIn.

With OAuth 1 application can talk to another application

The OAuth 2.0 authorization framework enables third party application (like udemy ) to obtain limited access to HTTP Service like Linked in either on behalf of resource owner or by orchestrating an approval interaction between resource owner and HTTP service

Or by allowing third party application to obtain access on its own

Meanings

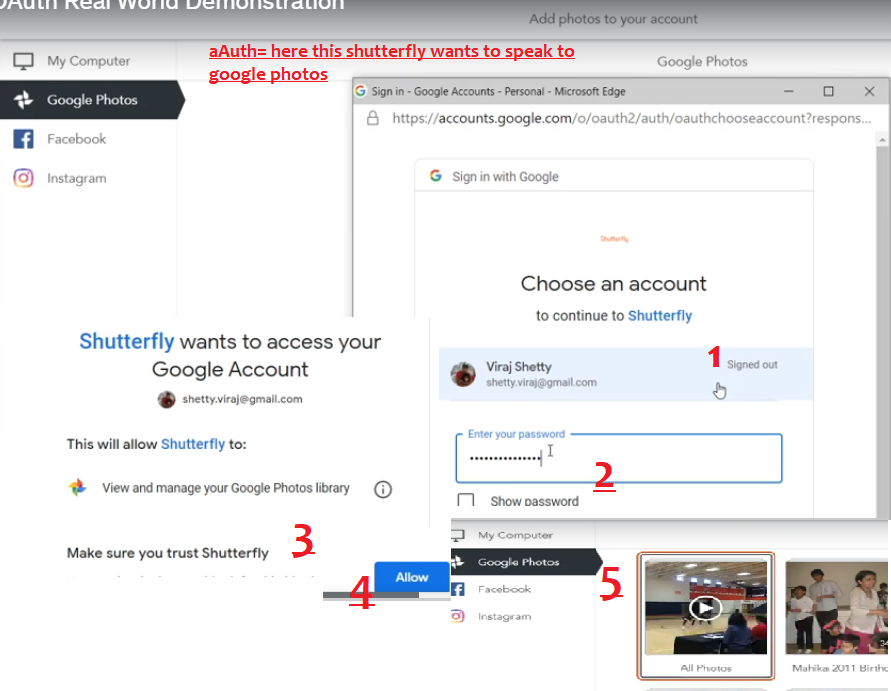
Client is somebody who receives a service from a professional group of people

Pani adigi cheyinchukune vadu client ante

Scopes

here it says shutterfly can view,manage,edit all your photos on google photos , so this view, manage, edit are the scopes what this application does.

Example-1-connect with Google photos



1. This shutterfly should talk to google photos api ( which is called resource server because our photos are there only ) and get the photos , hence shutter fly application is talking to other google photos application to get the photos

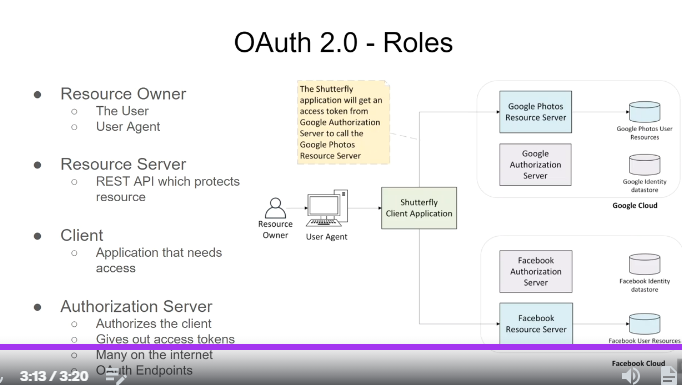
OAuth definition

1. The **Authorization**in OAuth (stands for Open Authorization) refers to the "**permission**" given by the user to the Third Party application to act on his or her behalf. In the **Shutterfly**demonstration, I clicked on the **Allow** button to give permission to the Shutterfly application to use my Google Photos.

Do not confuse this "Authorization" with traditional definition of Authorization for Enterprise applications. The traditional definition of Authorization refers to the ability of an Application to have different groups of users perform different actions within the application itself.

1. Throughout the course, I assume that the Shutterfly interaction with the Google Authentication Server is initiated from the backend server and not from the browser. In actual practice, this may or may not be true but this assumption helps me to explain **OAuth 2.0**
2. In future Lectures and Sections, I will describe exactly how everything works technically.

## OAuth Roles



1. We have 2 resource servers- 1st one is google photos resource sever, only authenticated and authorized users are allowed, auth servers will provide access tokens to access these servers

eX:- shutterfly will talk to google Authorization servers to get the access token which is needed to hit the Google photos resource server

Client id

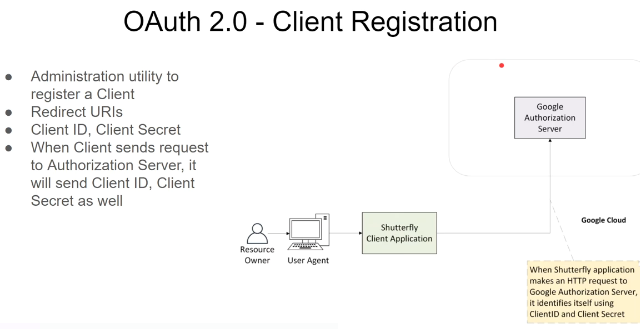
Google Auth server

Access token

Shutterfly client

Google Photos Resource server

## OAuth client Registration



When shutterfly will send the request to authorization server, we should pass the client id and client secret code , **like we are going to office with id card**, else they will think some rouge is coming throw him, always when we are accessing a secured environment we should show who we are

Opaque token

1. Hit Google photos Resource server with access token

**Google photos Resource server**

1. Verify access token with resource server
2. Query for access token

Client app like shutterfly

1. Got the access token

Google Auth server

1. **Token verification** Since access token is issued by the auth server, google photos Resource server will interact with auth server and check if the token is valid for how many minutes and
2. **Roles checking for the token** it will check the roles and like whether that token has access to delete anything , like if token is issued to see the photos he cant modify / he cant add the photos / he cant delete the album

Its called opaque token because token doesn’t depict the time and all. It contains only the string only auth sever can decode hence it is called opaque token

Authorization : Bearer \*\*\*\*\*\*\*\*\*\*token value\*\*\*\*\*\*

We should pass the token in the header

JWT an Structured token

JWT- json web token which will be given by auth servers instead of opaque token

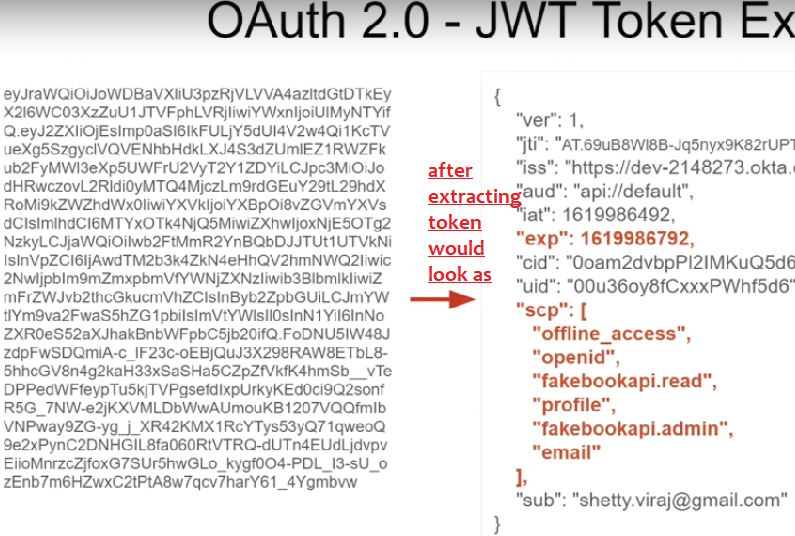
Why it is called json – because after extracting the token it will look like json

What is the issue with opaque token:- when auth severs generated opaque tokens, and when shutterfly app sends token to access resource servers, then auth servers needs to verify the access token whether this token is generated by valid auth server or not

To avoid separate call to authorization server for re-verification of those tokens , we will use signed tokens which need not be verified

Structured tokens are signed so now g.photos resource servers doesn’t call auth servers to verify the authentication

1. Resource sever will verify the signature of signed token and extract the scopes without sending token to auth servers



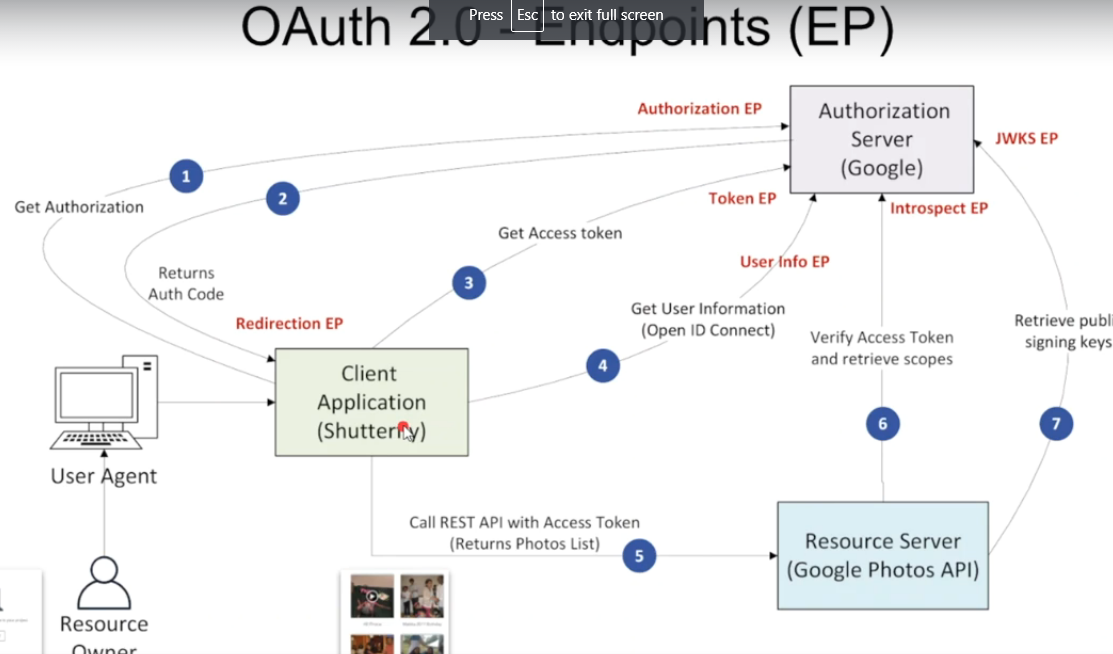
Scopes

Resource server will reject the request if appropriate scope is not found.

In the above see the scope “scp” field it has the scope facebook.read.api , it have only read access not write access

If the token is issued for gmail , u cant use that token for access youtube , if u try to hit the resource sever with that token , Resource server will see the scope and reject it

## Flow



1. If Client app wants to talk to Google photos resource server shutterfly client needs authorization so for that client shutter fly will ask for auth code and once it gets it , it will ask auth server for access token (2 types –opaque token , JWT)
2. If it is opaque token, it can’t be understood by resource server so again client will send that token to auth server because we need to extract token and get the scope of that token (states if it have delete access, or only read access…) and if it is JWT signed token it will just hit auth server and get the public key and since it is JWT – json web token It has all info refer above image for complete example
3. There is an additional Endpoint which is not shown in the previous lecture - the **Token Revocation** endpoint. Using the Token Revocation endpoint, a client can invalidate an access token and other related tokens. It's used for cleanup after a user logs out of the client application and in cases where the token is suspected of being compromised.
4. Req rejected by gmail server –invalid scope , because token issued to access google photos resource server not gmail server,
5. Hit auth server and get the token which has scope of photos resources

Google Gmail resource server

Google Auth server

You Tube

Google photos resource server

Google Gmail resource server

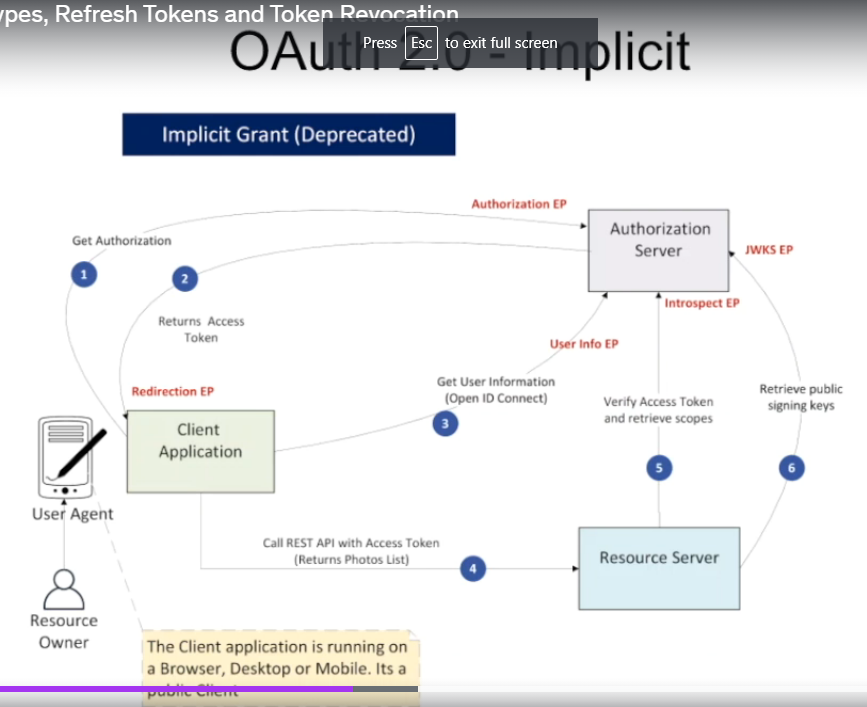
**Notes on Access tokens and Scopes**

An **accèss token** opens the door to the **Resource Server.** The Resource Server will first verify the token (i.e signature is good, expiry date validation passes, issuer is the right Authorization server and other checks). If verification passes, then Resource server will do authorization checks based on **scopes**associated with the access token. A Resource server can look at multiple scopes to figure out if the API call should be Accepted or Rejected. Think about a scope as an access privilege granted to the user to perform a certain functionality within the **Resource Server.** *Here I refer to the traditional definition of Authorization as defined for Enterprise Applications*.

For example, in the previous lecture we talked about the Google Photos API. The Google Photos API checks for the following scopes.

## Grant Types

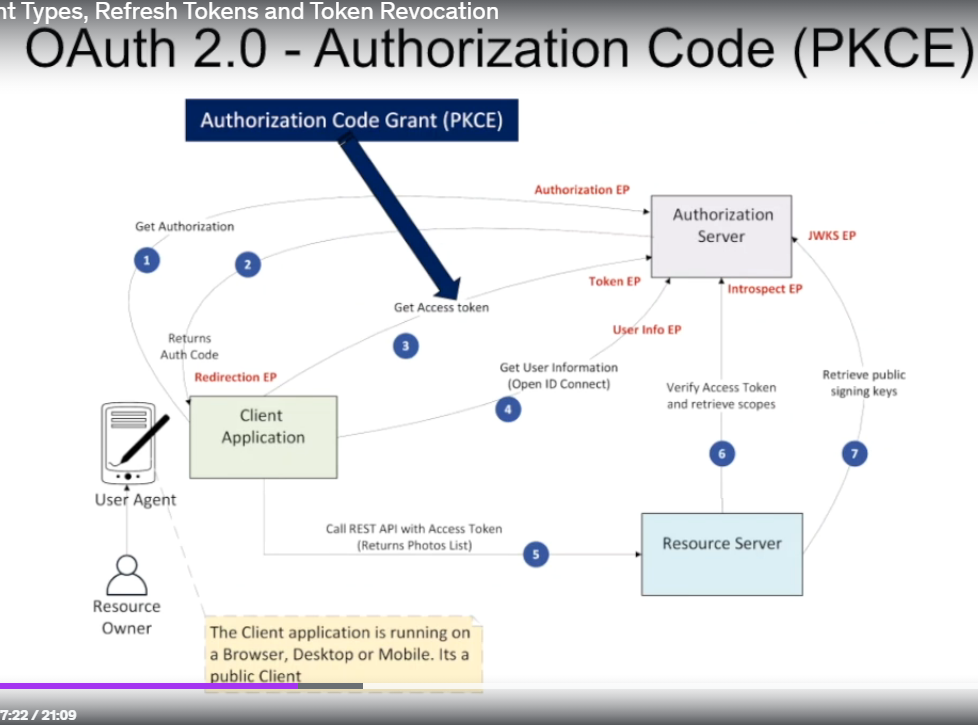
1. Implicit Grant –Its deprecated



It’s a single step process in first step itself we get the access token , which is not secure I also don’t know why is it so ? so its deprecated

In this implicit mode client will pass information like Client key, secret code and .. and it will immediately get the access token

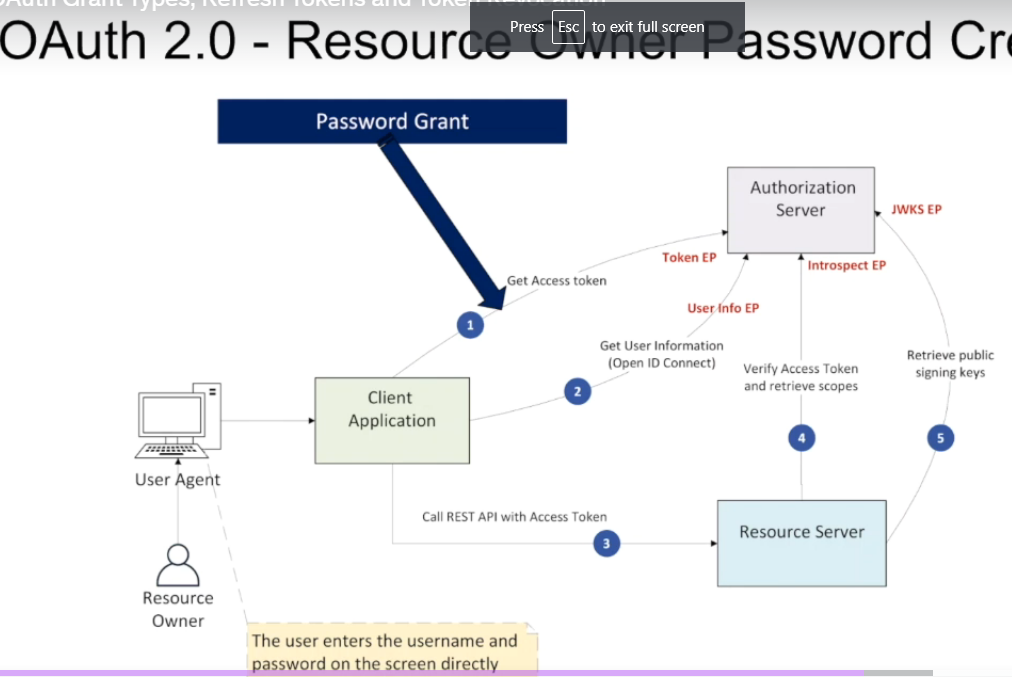
1. Authorization code grant –Always best



Authorization EP- means Authorization Endpoint

This is most recommended one

1. The client –shutterfly will send clients identifiable info like client id, secret code and it will **get the auth code** (Not access token)
2. Now client will send auth code and **get the access token**
3. Password grant



Lets say here if client is shutter fly and auth server is google auth server

Here we will give user our google name and password to shutterfly and it will happily store it and send it to google server and gets the access token,

This is security breach and worst approach when client and resource server are different,

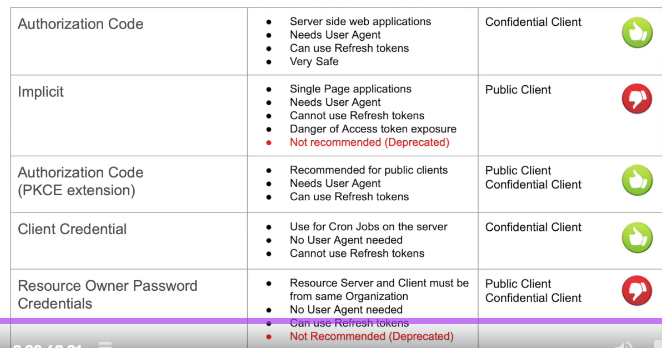
Use this approach when both client and server belongs to same app’s like gmail and google photos- client and resource servers are belong to same organization

When to use this?

Use it inside an organization because there all apps belong to same organization

Token Revocation

When the user logouts, if the auth code and if authorization tokens are not invalidated someone can come and tamper those codes and happily use those tokens on our behalf, so make sure to invalidate the token



As you can observe here, most recommended one is authorization code Grant with pkce extension

# References

For the Deep Dive sessions, I rely on a tool called **Postman**which makes it easy to send HTTP messages to the **Google**and **Okta**Authorization servers. It's an incredibly useful tool which is useful for testing REST APIs and to learn about HTTP messages.

Note : In my lectures, I copy the bearer token to the Authentication field to be explicit about where to include the token; Alternatively you can create a global variable (say **JWT-TOKEN**) and use the variable in all requests **{{JWT-TOKEN}}**. Then just set the variable **JWT-TOKEN** with the correct bearer token whenever needed.

The documentation for Postman is at the following location.

[Introduction | Postman Learning Center](https://learning.postman.com/docs/getting-started/introduction/)

You can install Postman using the following link.

[Download Postman | Try Postman for Free](https://www.postman.com/downloads/)

Note that the Postman UI has changed slightly from what is shown in the videos; but most of it remains the same. The following changes are noteworthy

* Main Screen Layout has changed. Some of the Tabs like **Collections, History, APIs**now appear vertically to the far left
* Clicking on **+** (New Collection) or **Adding a New Request**does not pop up a dialog box but reuses the panel to the right. An example image is shown below
* **https://www.googleapis.com/auth/photoslibrary**

View and manage your Google Photos library

* **https://www.googleapis.com/auth/photoslibrary.appendonly**

Add to your Google Photos library

* **https://www.googleapis.com/auth/photoslibrary.edit.appcreateddata**

Edit the info in your photos, videos, and albums created within this app, including titles, descriptions, and covers

* **https://www.googleapis.com/auth/photoslibrary.readonly**

View your Google Photos library

* **https://www.googleapis.com/auth/photoslibrary.readonly.appcreateddata**

Manage photos added by this app

* **https://www.googleapis.com/auth/photoslibrary.sharing**

Manage and add to shared albums on your behalf

**OAuth 2.0 Notes and Links**

An important idea in SAML and OAuth 2.0 is the idea of **Delegated Authentication**. In one of the last sections, I have a video which explains the Delegated Authentication and how it's fundamental to the idea of Cross Domain Authentication. Take some time to go through this lecture.

[The Idea Of Delegated Authentication](https://www.udemy.com/course/enterprise-oauth-for-developers/learn/lecture/34681508)

**OAuth 2.0 RFCs (Request For Comments)**

Some of the RFCs related to OAuth 2.0 are listed below for your reference.  As you can clearly see, OAuth 2 is a bunch of RFCs and not a single RFC. It evolved over a period of time. These RFCs go into a lot of details related to the fields and we will cover the vast majority of them in the coming sections.

* [The OAuth 2.0 Authorization Framework](https://tools.ietf.org/html/rfc6749)
* [Proof Key for Code Exchange by OAuth Public Clients](https://tools.ietf.org/html/rfc7636)
* [OAuth 2.0 Token Introspection](https://tools.ietf.org/html/rfc7662)
* [OAuth 2.0 Authorization Server Metadata](https://tools.ietf.org/html/rfc8414)
* [OAuth 2.0 Token Revocation](https://tools.ietf.org/html/rfc7009)
* [JSON Web Key (JWK)](https://tools.ietf.org/html/rfc7517)

**Cryptography Basics**

I would highly recommend that you take the 30 minutes tutorial of **Cryptography** basics provided in the Bonus section [**Cryptography Basics - Hashing, Encryption, Signatures**](https://www.udemy.com/course/enterprise-oauth-for-developers/learn/lecture/34923366) which gives a very good overview of Base64 encoding, URL encoding, SHA256 Hashing, RSA Encryption and Digital Signatures.

So, if access token contains only the scope "**https://www.googleapis.com/auth/photoslibrary.readonly**" and client calls the Google Photos API endpoint to upload a Photo -  that request will be rejected with an HTTP 403 error (Authorization error).

In case of Google APIs, the scopes look like URLs but they don't have to be. Scopes can be any string and when we create our own Resource Server we will see how to create our own scopes.