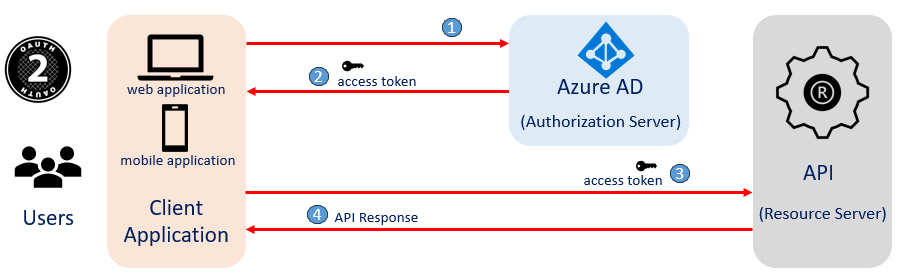
**Part 2B: OAuth 2.0 Authorization Code Grant with Azure AD**



**Register an Application with Azure AD**

**Prerequisites**

* An Azure account that has an active subscription. [Create Your Azure Free Account Today | Microsoft Azure](https://azure.microsoft.com/en-gb/free/?WT.mc_id=A261C142F)
* The Azure account must have permission to manage applications in Azure Active Directory (Azure AD).
* Set up a Tenant. [Quickstart: Create an Azure Active Directory tenant — Microsoft Entra | Microsoft Learn](https://learn.microsoft.com/en-us/azure/active-directory/develop/quickstart-create-new-tenant" \t "_blank)

*Create a Test User*

It is recommended to register a test user with Azure AD. Which we will use during the testing to acquire an OAuth token in various OAuth flows. I created a test user TestUser01.

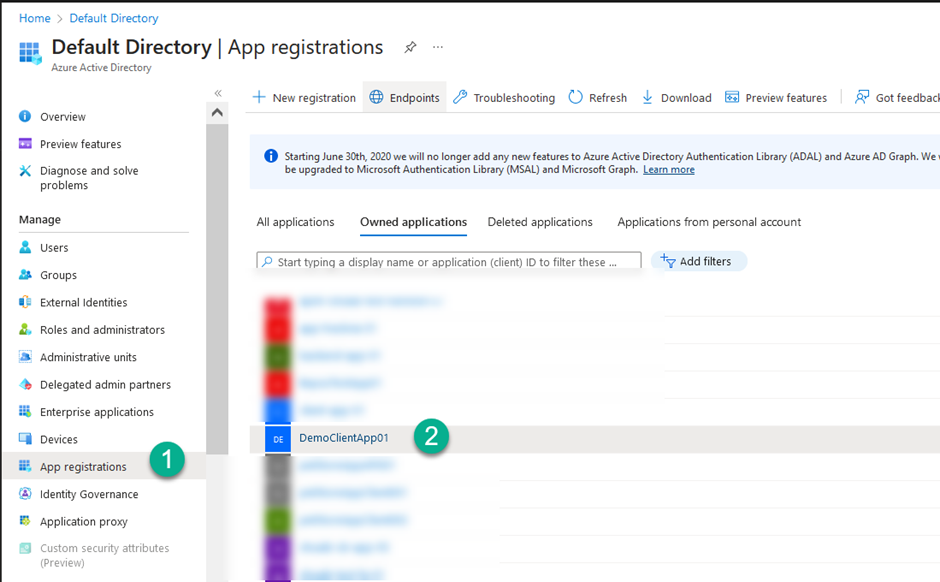
[Add or delete users — Microsoft Entra | Microsoft Learn](https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/add-users-azure-active-directory#add-a-new-user)

**Register a client application to get client\_id and client\_secret**

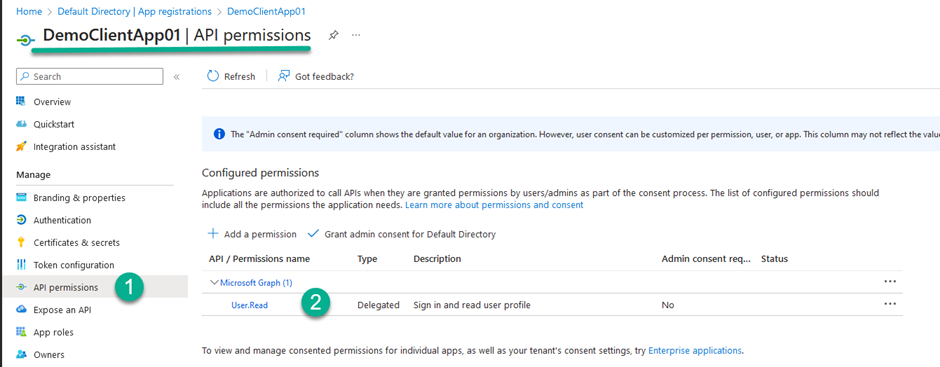
Get started with the by registering an application in Azure AD from the Azure portal. Application must be registered in a tenant that you manage.

Follow these steps to create the app registration:

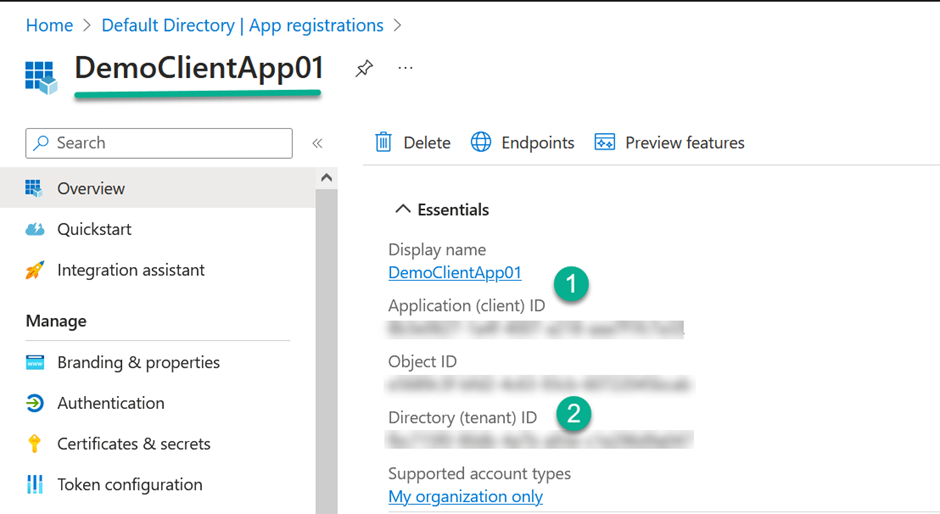
1. Sign into the Azure portal.
2. Select Azure Active Directory.
3. Under **Manage**, select **App registrations > New registration**.
4. Enter a display **Name** for your application e.g., DemoClientApp01.
5. Under Supported account types choose the first option. Leave Redirect URI blank. We are going to add the URI in the later step.
6. Register the Application by pressing the Register button.
7. Application DemoClientApp01 is registered successfully in Azure AD.
8. If Application doesn’t appear in the application list. Please refresh the page.



Open the application and select API Permissions. Note that Microsoft Graph → User.Read delegated permission is already assigned for sign in user profile.



Before moving forward save client ID and Tenant ID from the overview of DemoClientApp01.

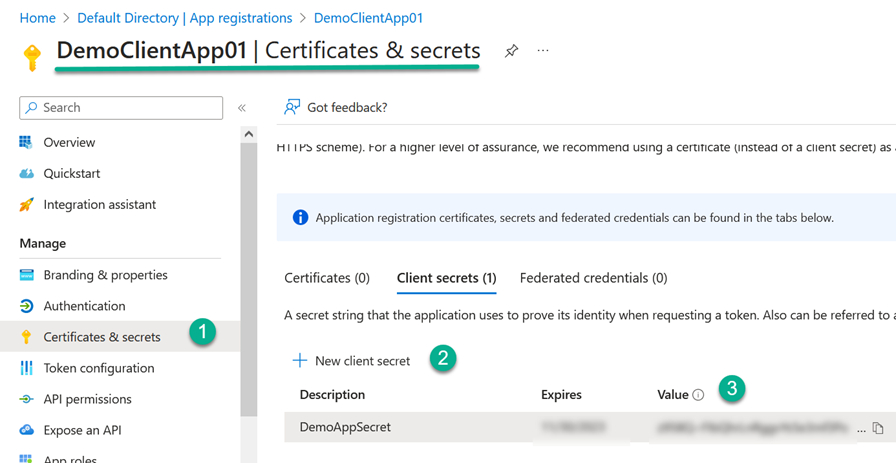


*Create a client secret.*

A client secret is a string that the application uses to prove its identity when requesting a token. You can also add client certificate instead of client secret or both certificates and secrets. A client certificate is considered more secure than client secrets. For this demo we are going to add client secret only.

1. From Azure Active Directory, select your DemoClientApp01.
2. Select **Certificates & secrets** > **Client secrets** > **New client secret.**
3. In a **Add a client secret** window, add a description for your client secret.
4. Select an expiration for the secret and click **Add.**

*Important: Record the secret’s value for use to get token. This secret value is never displayed again after you leave this page.*

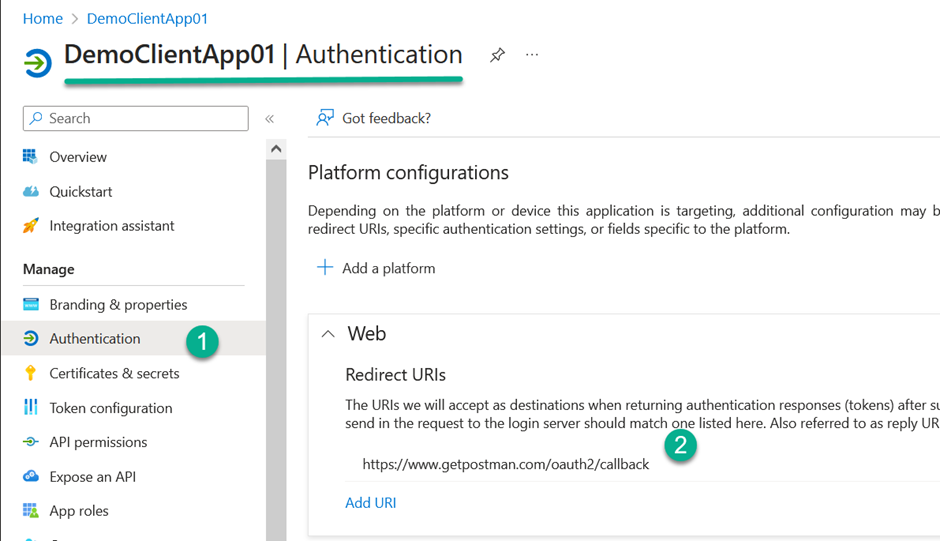


*Add Redirect URI for Postman.*

A *redirect URI* is the location where the Azure Active Director redirects a user’s client and sends security tokens after authentication. This redirect URI is going to be send in the request to the login server and must match one of the listed URI. Postman provides the following URI which we can be used as a redirect URI <https://www.getpostman.com/oauth2/callback.>

Redirect URI is going to be configured in **Platform configurations** of the DemoClientApp01. To configure redirect URI, follow these steps:

1. From Azure Active Directory, select your DemoClientApp01.
2. Select **Authentication** > **Platform configurations > Add a platform**.
3. Under **Configure platforms**, select the **Web Application** > **Web** tile for application type (platform) to configure its settings.
4. Add the redirect URI for postman.
5. Select **Configure** to complete the platform configuration.



**Authorization and Token Endpoint URLs**

The following are the authorization and Token endpoint URLs provided by Microsoft.

Authorization end point URL format to get auth code.

* https://login.microsoftonline.com/<<your tenant ID>>/oauth2/v2.0/authorize

Token end point URL format to get access token. This is going to be used to exchange the authorization code for an access token.

* https://login.microsoftonline.com/<<your tenant ID>>/oauth2/v2.0/token

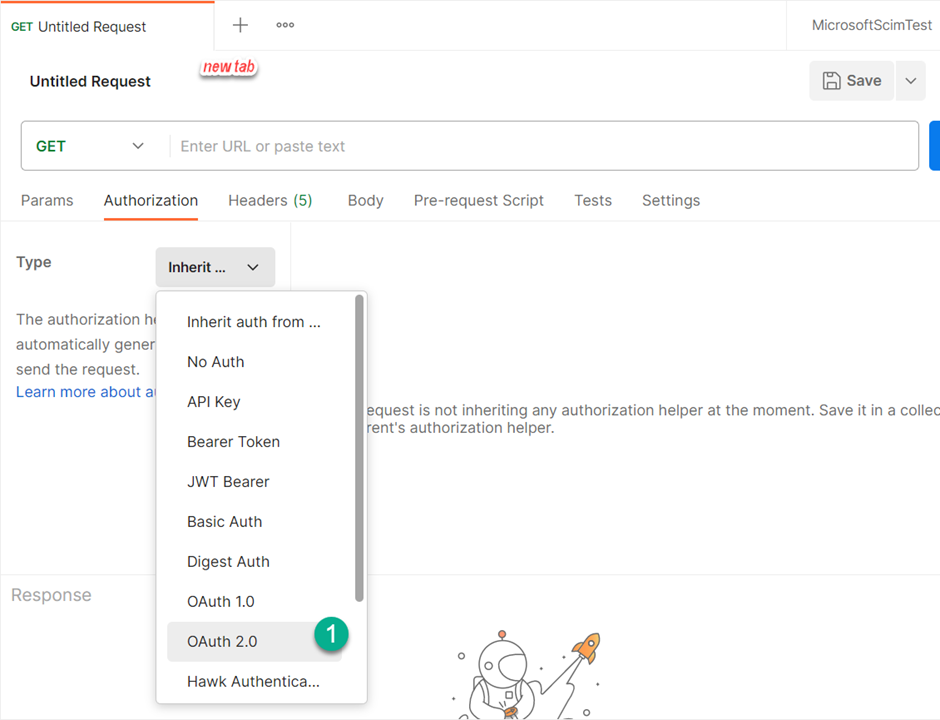
Note: <<your tenant ID>> is an ID you saved when register DemoClientApp01.

**Get access token using Postman.**

Download and install postman from the following URL. You can start it for free.

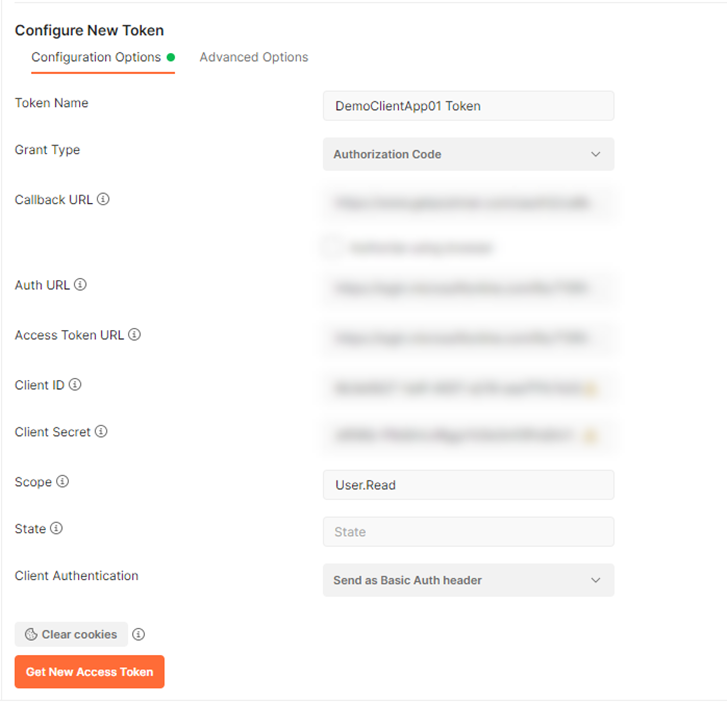
[Download Postman | Get Started for Free](https://www.postman.com/downloads/)

Once Postman is installed successfully then open a new tab and select Authorization and Auth 2.0.



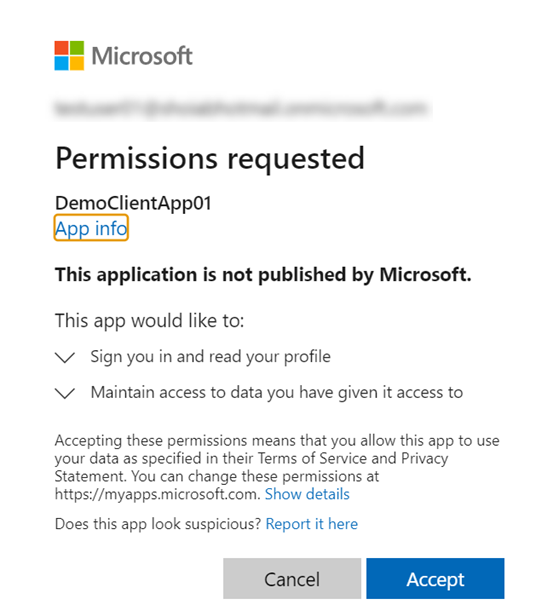
Under Configure New Token select the Grant Type as Authorization Code. Fill the following information:

* Token Name: Any Suitable Name for a Token
* Grant Type: Authorization Code
* CallBack URL: It is the Redirect URI provided above. For Postman it could be <https://www.getpostman.com/oauth2/callback>
* Auth URL: https://login.microsoftonline.com/<<your tenant ID>>/oauth2/v2.0/authorize
* Access Token URL: https://login.microsoftonline.com/<<your tenant ID>>/oauth2/v2.0/token
* Client ID: 8\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Client Secret: z\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Scope: User.Read

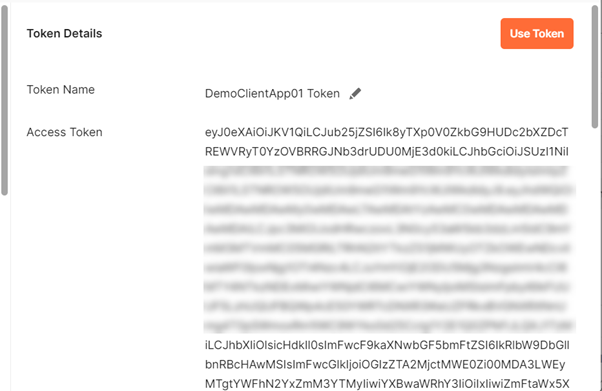


*Note: In Enterprise App > DemoClientApp01 > properties > Assignment required is set to Yes then users (and other apps or services) must first be assigned to this application before being able to access it. If this option is set to no, then all users will be able to sign in, and other apps and services will be able to obtain an access token to this service.*

Now Click the New Access Token Button. A login screen will pop up asking to enter username and password. Once the user is authenticated successfully, it will ask for a *permission*to allows you to sign into the app with your account and let the app read your profile.



Once you Accept it a new Access Token is issued by Azure AD.



You can the examine the details of the Logs by selecting Console in Postman which will give you the complete trace of the Request/Response calls.



**Access Token by Azure AD**

An Azure AD issues an access token as JSON Web Tokens (JWTs) that contain claims. Access token can also be an opaque token that conform to the [OAuth 2.0 framework](https://datatracker.ietf.org/doc/html/rfc6749#section-1.4). JWT is an open standard ([RFC 7519](https://tools.ietf.org/html/rfc7519)) that defines a way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because JWT is digitally signed. JWTs consist of three parts separated by dots (.), which are:

* Header
* Payload
* Signature

Therefore, a JWT typically looks like the xxxxx.yyyyy.zzzzz. I am not going to explain what Header, Payload and Signature means. I am going to write a separate post to explain them and how to verify JWT.

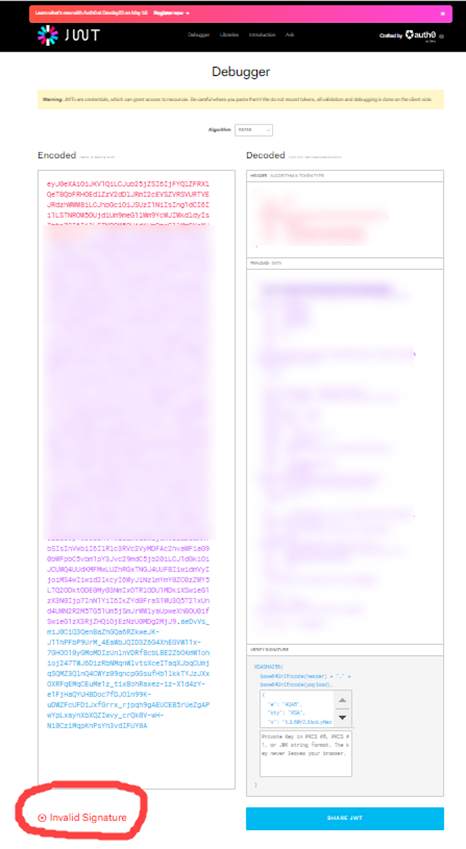
There are some online tools available which you can use to verify the JWT token generated in above step.

Some of the online options are:

* [JSON Web Tokens — jwt.io](https://jwt.io/)
* [jwt.ms: Welcome!](https://jwt.ms/)
* [JSON Web Token Verifier (davetonge.co.uk)](https://jwt.davetonge.co.uk/)

**Decode JWT Access Token**

I am going to use the first option to decode the JWT token. Let’s open the website jwt.io. Then copy and paste the token acquire in the previous step.



*Note: You can also use the value of scope as*[*https://graph.microsoft.com/User.Read.*](https://graph.microsoft.com/User.Read.)*Then you will get aud claim as “aud”: “https://graph.microsoft.com" instead of “aud”: “00000003–0000–0000-c000–000000000000”.*

The signature of this token is invalid because this token is issued for Microsoft Graph. These tokens use special signing mechanism, and it is not possible to validate signature using public validators. And when you are using Microsoft Graph API you don’t need to validate the token instead graph API server side will validate it.

Get Shoaib Alam’s stories in your inbox

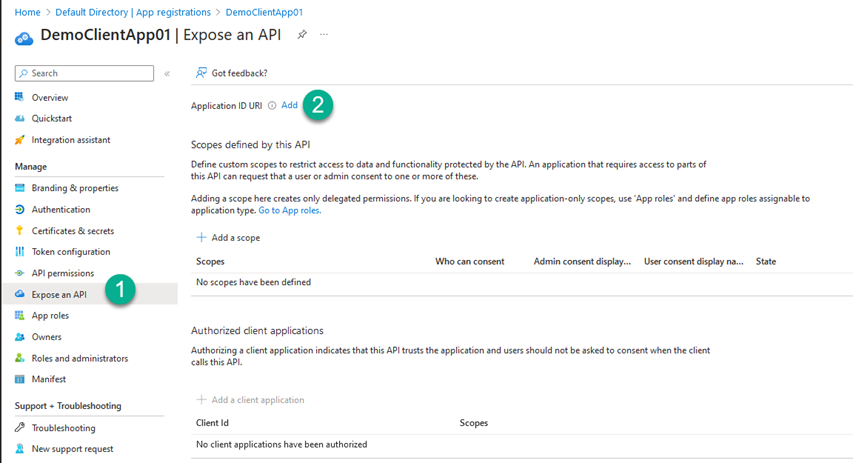
Join Medium for free to get updates from this writer.

Subscribe

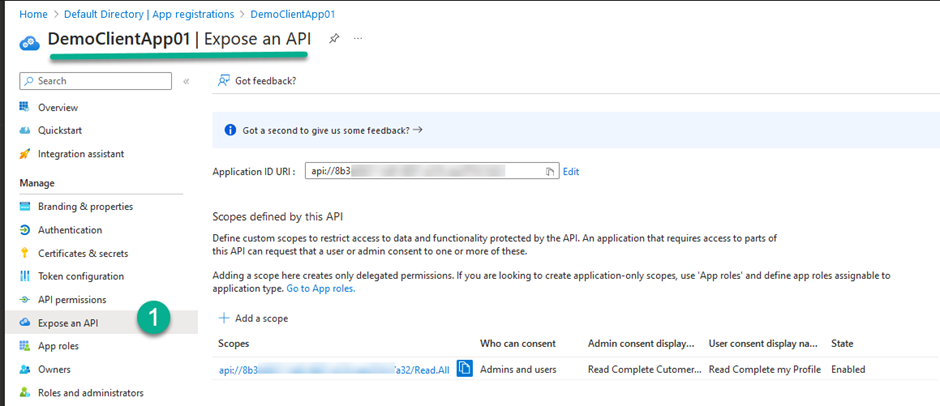
You could validate the access token if you are acquiring the access tokens to access your own resource server and not the Graph API. To do this we need to create a custom scope for DemoClientApp01 application.

**Custom scope**

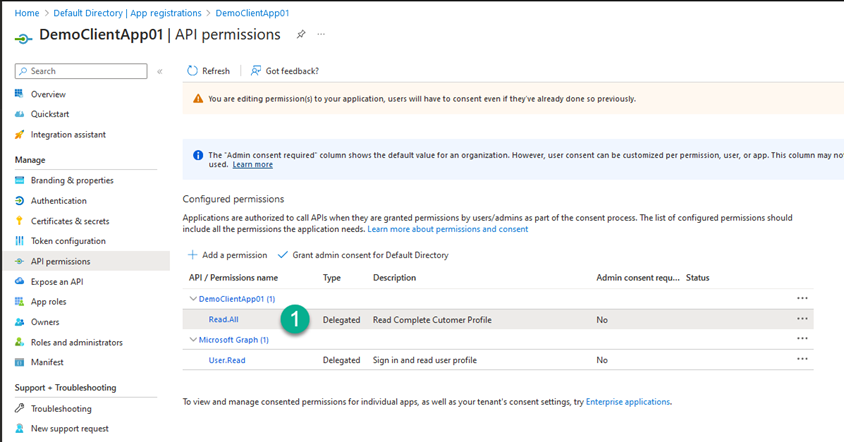
To create a custom scope for DemoClientApp01 the first step is to add an Application ID URI for DemoClientApp01. The Application ID URI is a globally unique URI used to identify the web API. You can either use the default value in the form of api://<application-client-id> or specify a more readable URI. I am going to use default value. This URI is the prefix for custom scopes like api://<application-client-id>/scope-name



Create a custom scope by going to **Expose an API > Add scope** and follow the prompts. Let’s add Read.All scope like api://{application-id}/Read.All



Add newly created scope in the application API permissions (API permissions > Add a permission > My APIs > DemoClientApp01 > Delegated Permissions > Check Read.All Scope > Add permission)



In postman change the Scope to api://{application-id}/Read.All and get the new access token.



As you can see now signature is verified and aud claim becomes api://{application-id}/ and a scp claim Read.All added in the access token.

**Add an application to assign Permissions and Scopes**

So far, we have created a DemoClientApp01 client app access to your own API. In this step we are going to create a new Azure AD Application and expose it to DemoClientApp01 app (or multiple client apps) by adding scopes. By registering and exposing an App through scopes, we can provide permissions-based access to its resources to authorized users and client apps that access your API.

*Register an application and add Scopes.*

Register a new Application with Azure AD called DemoWebApp. Skip adding a redirect URI and credentials steps. We are going to add scopes to the DemoWebApp so it can provide granular permission to consumers.

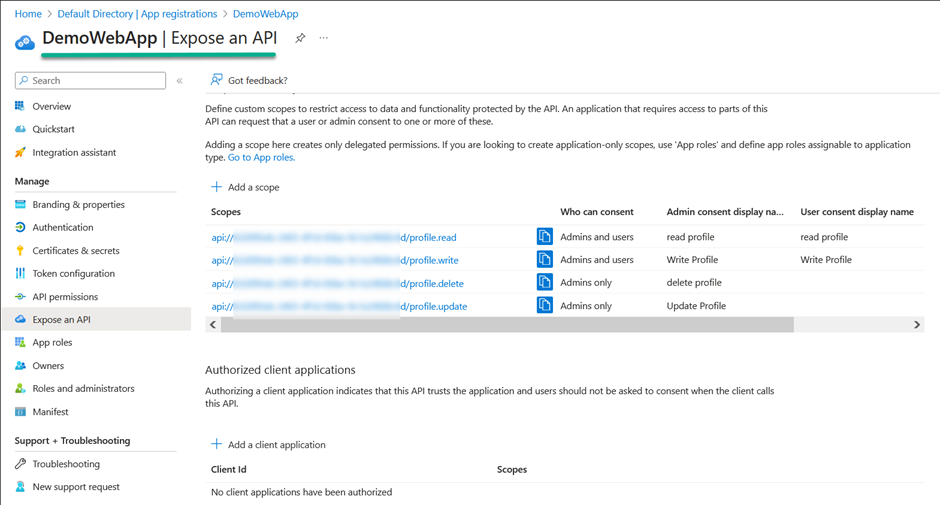
To add scopes to DemoWebApp, first add an Application ID URI to it. We are going to use the default value in the form of api://<application-client-id> as explain above. Create the following four custom scopes from **Expose an API > Add scope** and follow the prompts.

* api://{application-id}/profile.read
* api://{application-id}/profile.write
* api://{application-id}/profile.delete
* api://{application-id}/profile.update

profile.read and profile.write scope assigned an Admins and users consent and profile.delete and profile.update assigned an Admins only consent.

For Admins and users consent when a user attempts to sign into an application using its credentials., these credentials are checked to determine whether consent has already been granted for the user or not. If no previous record of user or admin consent for the required permissions exists, then they’ll be prompted to consent those permissions to the application.

Some applications might require an administrator to be the one who grants consent. If Admins only consent scope is assigned, then permissions can only be consented by an administrator. For example, permission to do highly privileged operations or sensitive data.



**Delegated Permission**

When a user signs into an app and uses it to access some resource, like their profile, the app will first need to ask for permission to access this resource on behalf of the user. This common scenario is called delegated access. Delegated access requires delegated permission.

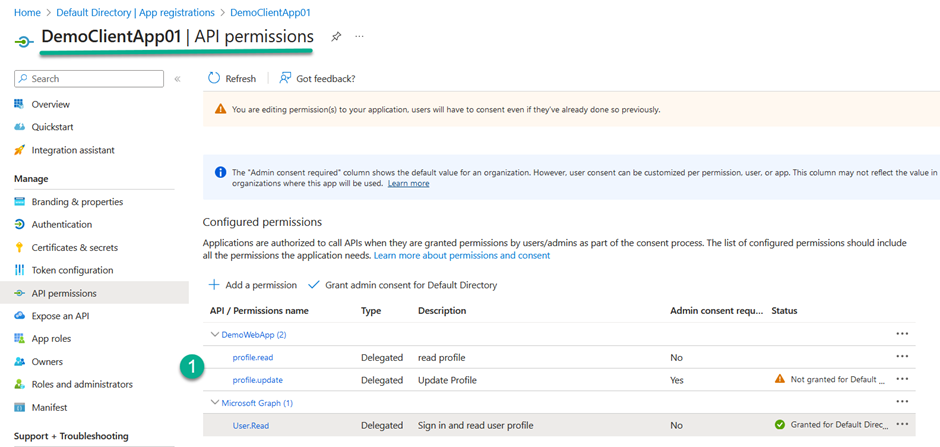
Delegated permissions can also be referred to as scopes. Scopes are permissions for a given resource that represent what a client application can access on behalf of the user.

When using delegated permissions, it is also required to grant a consent by users to allow the application to access their resources. Consent can also be granted by an admin, who can consent the application to access the resources for any user who uses the application.

I will discuss Application Permission in Client Credential Flow.

**Add Delegated permission to client application.**

From Azure Active directory navigate to DemoClientApp01 and add permission from **API Permissions > Add a permission > My APIs**and select the DemoWebApp created in the above step. Select **Delegated Permission** from type of permissions and choose profile.read and profile.update delegated permissions and click Add Permissions.

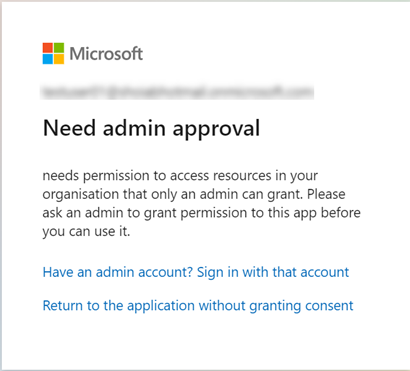


Now we are ready to test using Postman. Add the following two scopes for the request:

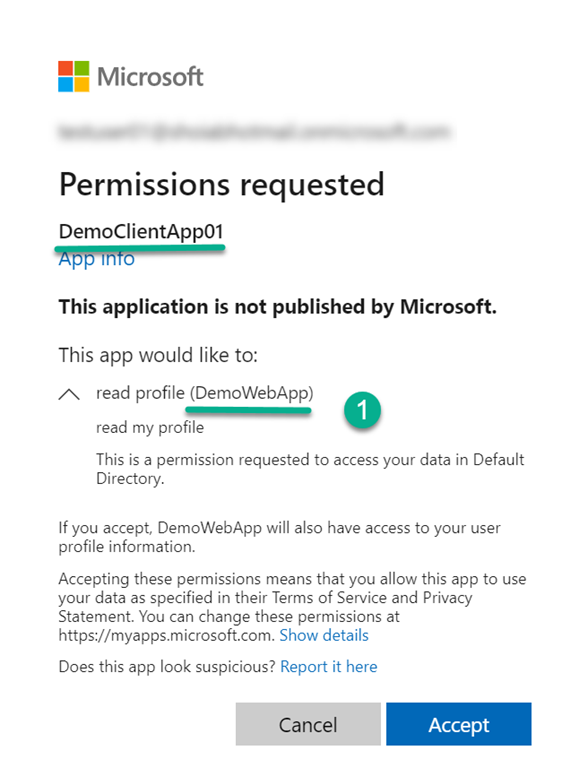
* api://{application-id}/profile.read
* api://{application-id}/profile.update

Remember profile.read requires Admins and Users consent but profile.update requires Admins only consent.

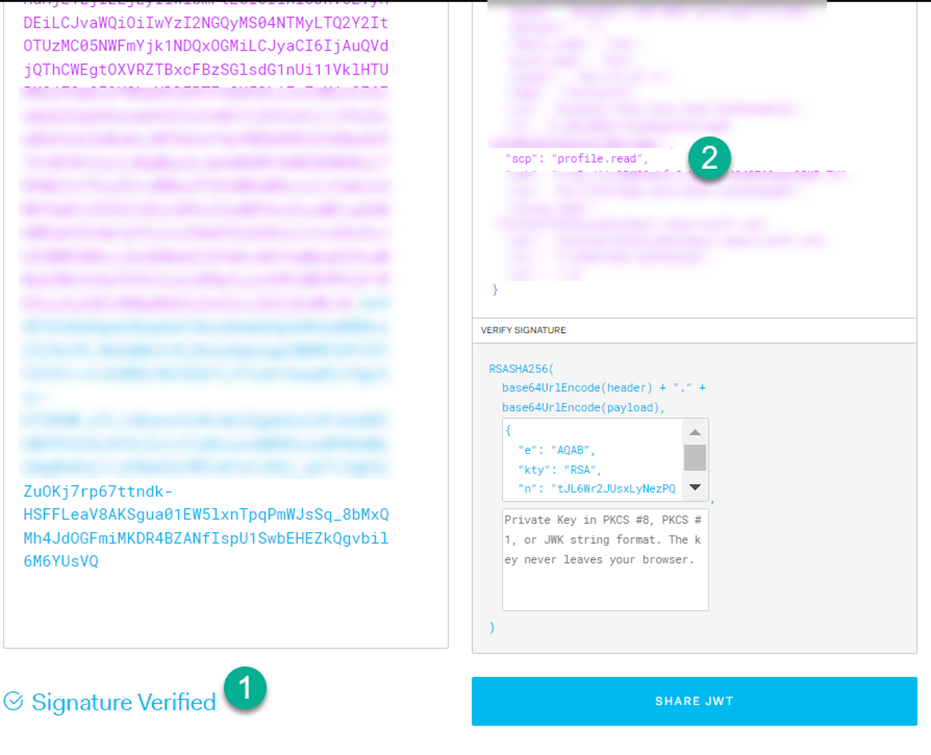
Click the Get New Access Token button. It will prompt that it Need admin approval and deny the access. This is due to the fact that we are sending profile.update scope as a part of token request and it requires Admins only consent.



Remove the profile.update scope and leave profile.read only and try again. Now it is going to request for a permission (if the consent has not already been granted) to a user as profile.read scope has Admins and users consent.

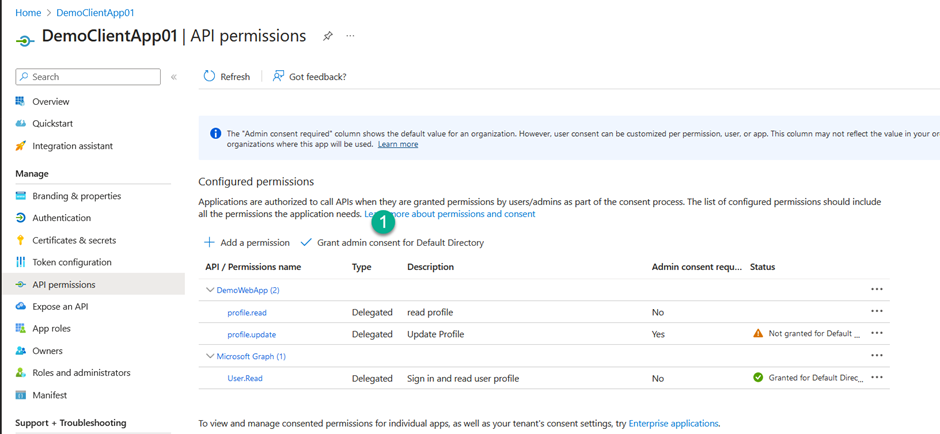


Once the user clicks the Accept button it will generate an access token. When you decode the access token a scope claim as scp is added to the token and signature is also verified.



You can grant tenant-wide admin consent to your applications you developed or registered directly in your tenant. Granting admin consent is a sensitive operation, potentially allowing the application’s publisher access to sensitive data or the permission to do highly privileged operations.

Carefully review the permissions that the application is requesting before you grant consent.

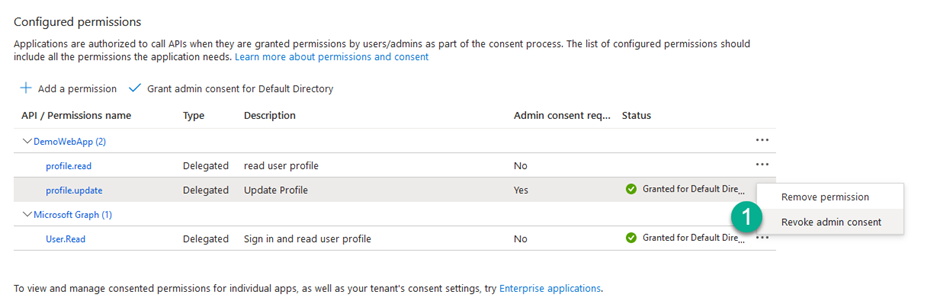


Note: There is “Yes” under the “Admin consent required column” and the “Not granted” warning in the “Status” column for profile.update permissions name. This indicates that the permission will not take effect until an admin consent to it.

You can grant admin consent from **API Permissions > Grant admin Consent for Default Directory**and click the Yes button when prompted for confirmation.

The Configured Permissions list displays again with the warning replaced by a “Granted” message in the “Status” column. The account now has permission to profile.update profile.

Use Postman to get new access token. It is not going to prompt for an admin approval to update the profile. Also, in the access token the scope i.e., scp claim has both profile.read and profile.update permissions. You can revoke admin consent by right clicking the context menu and select Revoke admin consent option and follow the prompt.



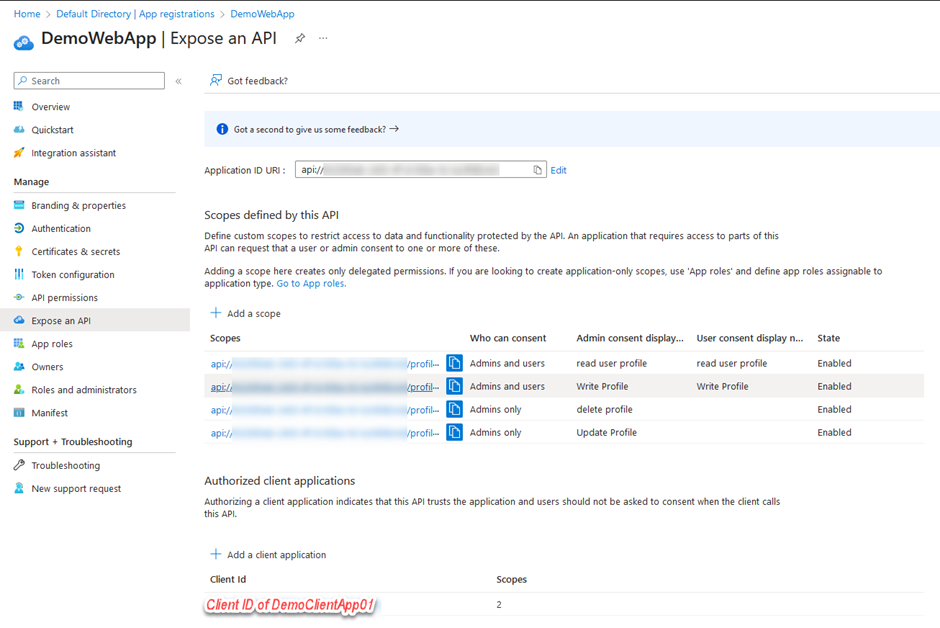
Use Postman to get new access token and it will prompt for an admin approval and deny the access.

**Preauthorization**

Preauthorization allows a resource application owner to grant permissions without requiring users to see a consent prompt for the same set of permissions that have been preauthorized. This way, an application that has been preauthorized won’t ask users to consent to permissions.

If you add your client app ID in Authorized client applications list, when your client calls the API, it will not need to consent. Authorizing a client application indicates that this API trusts the application and users should not be asked to consent when the client calls this API.

You can authorize the client application i.e., DemoClientApp01 from **DemoWebApp > Expose an API > Add a client application Grant**and follow the prompts.



**Get access token Http GET and POST Request**

*Authorization code using HTTP GET method and authorization end point.*

An alternate method to get an OAuth token is to use HTTP GET and POST methods. To start the OAuth process, send an HTTP GET to the authorization server (Step 1). The resource owner authenticates to authorization server and authorizes the client (Step 2a & 2b).

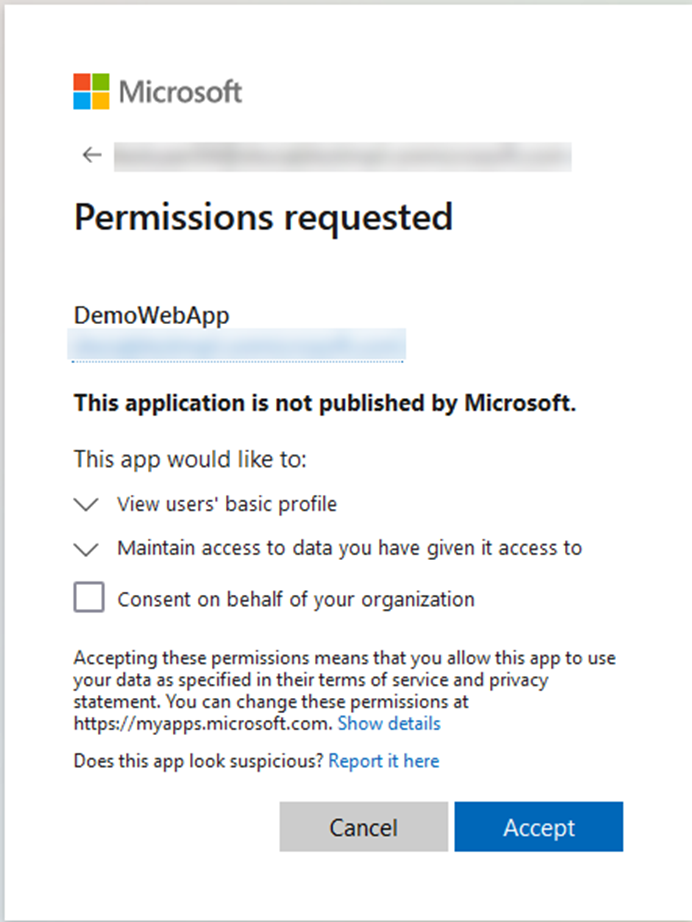
The authorization server redirects the user to a redirect\_uri with authorization code (step 3).

The HTTP GET methods required the followings query parameters:

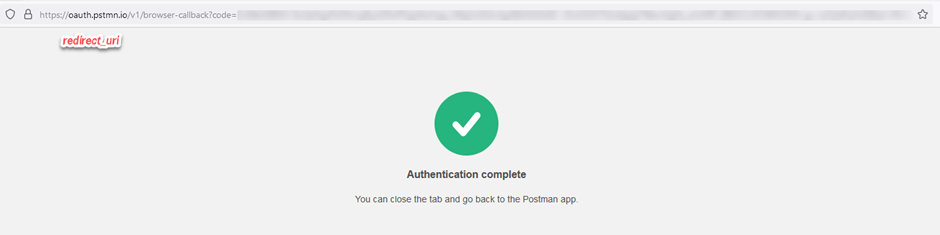
* response\_type
* client\_id
* redirect\_uri
* scope



Copy and paste the above URL in the browser. A login screen will pop up asking to enter username and password. Once the user is authenticated successfully, it will ask for a permission to allows you to sign into the app with your account and let the app read your profile.



Once you Accept the required permission it will redirects to redirect\_url and generate a code as a query parameter (step 3).

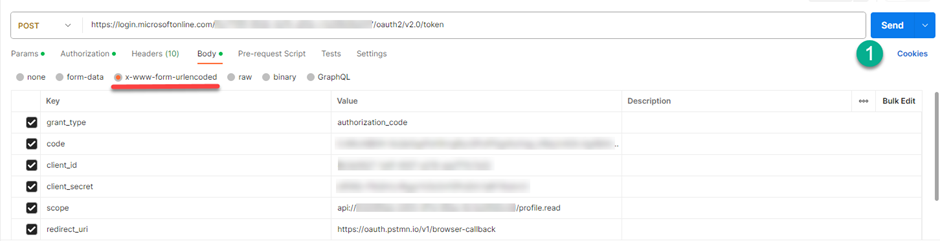


Extract the code from the query parameter.

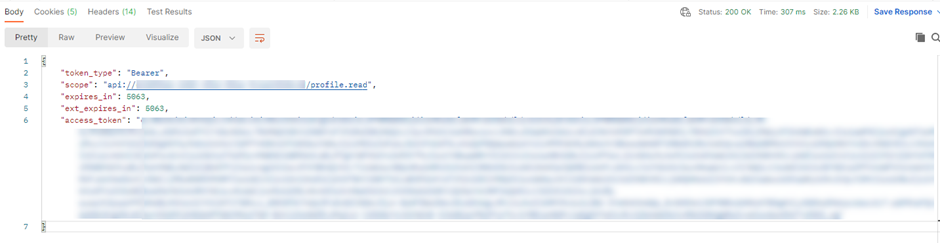
*Access token using HTTP POST method and token end point.*

Open Postman and create the HTTP POST request which requires the followings for Body. It is recommended to create the POST request upfront as authorization code is short lived and might expire by the time you finish creating the POST request (step 4).

* grant\_type
* code — From previous HTTP GET request.
* client\_id
* client\_secret
* Redirect\_uri



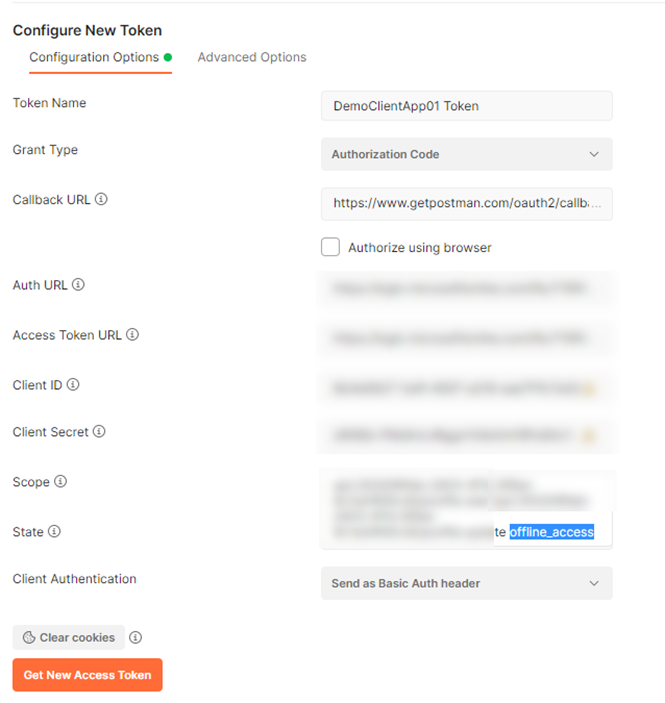
Send the POST request to the token end point (step 5) and it will generate an access token.



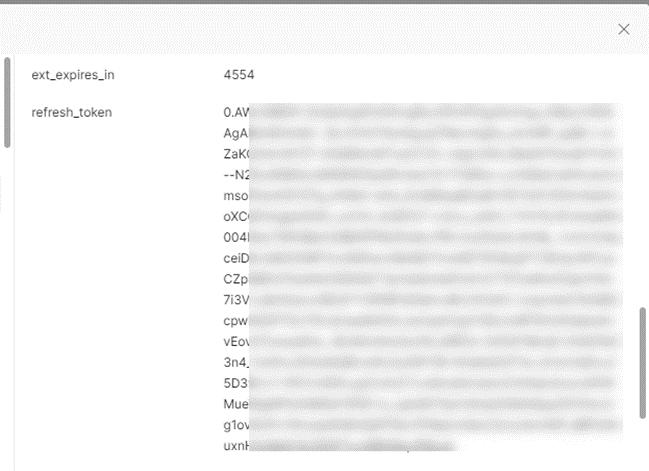
**Refresh Token**

Access tokens are short lived and once they expire, the client application can use a refresh token to “refresh” the access token. Refresh token allows a client application to get new access tokens without having to ask the user to log in again. This happens behind the scenes without user interaction, facilitating an improved user experience without compromising security. Refresh tokens do not give the user any additional access beyond what was originally allowed.

To get a Refresh Token, include the offline\_access scope when initiating a token request.



In the response an additional token is added called refresh\_token as below:



You can refresh or get a new access token by submitting another POST request to the /token endpoint, this time providing the refresh\_token. Refresh tokens are valid for all permissions that your client has already received consent. The body of the request must include the following parameters and the Content-Type header must be set to *application/x-www-form-urlencoded*.

* grant\_type
* client\_id
* client\_secret
* refresh\_token

Set the refresh token parameter to the value of the refresh token retrieved in the previous step, and the grant type set to *refresh\_token*

