

OAuth 2.1 Playground

[View OAuth 2.1 Guides](#) 

APEX Cloud secures APIs through **OAuth 2.1** by Corppass. Dive into each step of the Authorization flow and learn how to handle each component.

1

Creating an application and registering it as an OAuth Client

Create an application and register it as an OAuth Client

2

Subscribe your application to an OAuth 2.1 protected API

3

Create a secret code verifier and code challenge

4

Build the authorization URL and redirect the user to the authorization server

5

Prepare the client assertion JWT before exchanging code for token

Prepare the client assertion JWT before exchanging the code for a token

6

Exchange the authorization code and code verifier for an access token

Checklist

⏪ Start from the beginning

- Registered an OAuth client from an application to get a Client ID
- Specified at least 1 redirect URL for my OAuth client
- Specified scopes for my OAuth client
- Provided my JWKS
- Subscribed my application to the API I want to transact with
- Generated an API key to allow my application to be recognized by the API Gateway
- Generated a code verifier
- Generated a code challenge from the code verifier
- Constructed the Authorization URL with my OAuth client ID, scopes, redirect URL, code challenge

Checklist verbs should be in present tense since they are “to-do”

Change ‘my’ to ‘your’

Construct the Authorization URL with your OAuth client ID, scopes, redirect URL, and code challenge

1. Create an application and register an OAuth Client

Before you can begin the flow, you'll need to register an oauth client via an application. Registration will give you a **client ID** that the authorization server will recognise (with the client's corresponding scopes and redirect urls).

For the purposes of this demo, we don't require that you create an actual application. Instead, you can use the demo application below that will be stored in your browser's local storage.

Application Name

My OAuth Playground App

Auto-generated Client ID

fb43153e-2c6c-4e11-91c0-5dc42aaa0b37

✓ Register Demo Application

Before you can begin the flow, you'll need to register an **OAuth** client via an application. After registering, you will get a client ID that the authorization server will recognise, with the client's corresponding scopes and redirect **URLs**.

For the purposes of this demo, we don't require that you create an actual application. Instead, you can use the demo application below that will be stored in your browser's local storage.

The Redirect URL is the endpoint that the Authorization Server will callback to (if exactly matched) with the Auth code after a successful user login. You may submit multiple redirect URLs if you require them, but we will use just one for this demo.

If the redirect URL is not one of the registered redirect URLs, then the server will show an error indicating such, and not redirect the user.

Redirect URL

<https://services-dev.api.developer.tech.gov.sg/resources/playground/oauth>

✓ Use Default Demo Redirect URL

The Redirect URL is where the Authorization Server will call back to if it matches the Auth code after a successful user login. While you can submit multiple redirect URLs if needed, for the purpose of this demo, we'll be using just one.

If the redirect URL isn't a registered redirect URL, the server will show an error and won't redirect the user.

Scopes are used to control access levels granted to an access token. Access tokens can have specific permissions, such as `READ_A` or `WRITE_B`. If a client with a `READ_A` scope tries to access an API requiring `READ_B` access, the request will be denied.

For this example, please select all of the scopes below to register the demo application with (that will be used in the later sections)

Scopes


`READ_USER`

`READ_ASSET`

Scopes are used to control the access levels granted to an access token. Access tokens can have specific permissions, such as `READ_A` or `WRITE_B`. If a client with a `READ_A` scope tries to access an API requiring `READ_B` access, the request will be denied.

For this example, select all of the scopes below to register the demo application. These will be used in the later sections.

The JSON Web Key Set (JWKS) is a set of keys containing the public keys used to verify any JSON Web Token (JWT) issued by the Authorization Server. For this demo, you may generate a random JWKS below, and we will store the corresponding private key (that will be used in the later steps) in your browser's local storage.

In the actual client registration, instead of specifying the JWKS in JSON form, you are expected to host the JWKS in an endpoint of your choice. You may refer to our guides on [creating the JWKS Endpoint](#) 

JWKS

```
1  {
2    "keys": [
3      {
4        "kty": "EC",
5        "kid": "85-bTmcBpynGgtXQz0X1iLMxlfdl2zmjx31K-w3NkeY",
6        "use": "sig",
7        "alg": "ES256",
8        "crv": "P-256",
9        "x": "wvJL-TRTpgDFLVP5_qkET1XAHnsfAh0Z6xeJNoN2HwI",
10       "y": "HX0vyX875xUauw32ip0ywMwFJU8ma06Ek2Kv-L2b-ko"
11     }
12   ]
13 }
```

The JSON Web Key Set (JWKS) is a set of keys containing the public keys used to verify any JSON Web Token (JWT) issued by the Authorization Server. For this demo, you may generate a random JWKS below, and we will store the corresponding private key in your browser's local storage, to be used in later steps.

In the actual client registration, instead of specifying the JWKS in JSON form, you are expected to host the JWKS in an endpoint of your choice. You may refer to [our guide on creating the JWKS Endpoint](#).

2. Subscribe your application to an API and generate API keys for your application

Now that you have an active OAuth Client registered from a base application, you may continue the OAuth development process by subscribing the application to any API (ideally one that is protected by OAuth 2.1).

This will be the API in the APEX API gateway that will require the OAuth token generated at the end of the OAuth flow.

Application Name

My OAuth Playground App

As only approved, **subscribed** applications are allowed to transact with an API behind the gateway, we use the API key (that will be tied to the application after generation) to determine if the incoming request has the correct access rights to the requested API.

Now that you have an active OAuth Client registered from a base application, you may continue the OAuth development process by subscribing the application to any API, ideally one that is protected by OAuth 2.1.

This API in the APEX API Gateway will require the OAuth token generated at the end of the OAuth flow.

Since only **approved** and **subscribed** applications are allowed to transact with an API behind the Gateway, we use the API key which will be associated with the application after generation to determine whether the incoming request has the correct access rights to the requested API.

3. Create a Code Verifier and Challenge

Before redirecting the user to the authorization server, the client first generates a secret code verifier and challenge.

The code verifier is a cryptographically random string using the characters A-Z, a-z, 0-9, and the punctuation characters `-_~` (hyphen, period, underscore, and tilde), between 43 and 128 characters long.

`code_verifier`

🔗 Generate Code Verifier

Before redirecting the user to the authorization server, the client needs to first generate a secret code verifier and code challenge.

The code verifier is a cryptographically random string using the characters A-Z, a-z, 0-9, and the punctuation characters `-_~` (hyphen, period, underscore, and tilde), between 43 and 128 characters long.

Once the client has generated the code verifier, it uses that to create the code challenge. For devices that can perform a SHA256 hash, the code challenge is a BASE64-URL-encoded string of the SHA256 hash of the code verifier. Otherwise, the same verifier string is used as the challenge.

```
base64url(sha256(code_verifier))
```

↻ Generate Code Challenge from Verifier

Separate into two paragraphs:

Once the client has generated the code verifier, it uses that to create the code challenge.

For devices that can perform a SHA256 hash, the code challenge is a BASE64-URL-encoded string of the SHA256 hash of the code verifier. Otherwise, the same verifier string is used as the challenge.

4. Build the Authorization URL

With the code challenge generated, you may construct Authorization URL with your application's client ID, scope(s) and redirect URL of choice.

This is what we have gathered so far (you may navigate backwards to verify the information below):

With the code challenge generated, you can now create the Authorization URL with your application's client ID, scope(s), and redirect URL of choice.

These are the parameters we have gathered so far. You may navigate backwards to verify the information below.

Sample OAuth Server

My OAuth Playground is requesting permission for the purpose of:

Read user profile

Read asset data

Reject

Allow

My OAuth Playground is requesting permission to:

5. Prepare the `client_assertion` JWT before exchanging code for token

The User has logged in and given consent to the scopes required. The Authorization Server has redirected back to the specified redirect uri (this page) with a `short-lived authentication code`.

Before you can submit a `POST` request to exchange the authentication code for a valid access token, you will first need to prepare the `client_assertion` as part of the `POST` request in the next step

5. Prepare the `client_assertion` JWT before exchanging the authentication code for an access token

The user has logged in and given consent to the scopes required. The Authorization Server has redirected back to the specified redirect URI (this page) with a short-lived authentication code.

Before you can make a `POST` request to exchange the authentication code for a valid access token, you first need to prepare the `client_assertion` parameter as part of the `POST` request in the next step.

With the values above, we can construct the base signing options for the eventual JWT that will be the `client_assertion`. Once ready, you may click on the sign button to create the `client_assertion` with the options below.

Client Assertion Options

```
1 {
2   "issuer": "fb43153e-2c6c-4e11-91c0-5dc42aaa0b37",
3   "subject": "fb43153e-2c6c-4e11-91c0-5dc42aaa0b37",
4   "audience": "https://services-dev.api.developer.tech.gov.sg/sample/oaui",
5   "jwtid": "session-1701701147867",
6   "keyid": "85-bTMcBpynGgtXQz0X1iLMxlfdl2zmjx31K-w3NkeY",
7   "expiresIn": "5m"
8 }
```

Client Assertion Token

-

✓ Sign Client Assertion with Private Key

Using the values above, we can set up the base signing options for the JWT (`client_assertion`).

Click the button below to create the `client_assertion` with the options provided.

6. Exchange the Authorization Code for the access token

The User has logged in and given consent to the scopes required. You have also prepared the `client_assertion` token.

You are now prepared to swap the authorization code for an access token. The client application will construct a **POST** request to the `token` endpoint with the parameters below:

From the parameters above, we can construct the **POST** request (curl for this example):

Token Endpoint

```
1 curl --location 'https://services-dev.api.developer.tech.gov.sg/sample/oa
2 --header 'Content-Type: application/x-www-form-urlencoded' \
3 --data-urlencode 'code=811010a1-f185-4d73-96cb-f0ebb5fd07f7' \
4 --data-urlencode 'client_assertion=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIs
5 --data-urlencode 'client_id=fb43153e-2c6c-4e11-91c0-5dc42aaa0b37' \
6 --data-urlencode 'client_assertion_type=urn:ietf:params:oauth:client-as
7 --data-urlencode 'grant_type=authorization_code' \
8 --data-urlencode 'redirect_uri=https://services-dev.api.developer.tech.
9 --data-urlencode 'code_verifier=MmP4qzZC42G0Fwq50XfIQ-NFeIK3AhWMrSt-koj
```

6. Exchange the authorization code for an access token

The user has logged in and given consent to the scopes required and the `client_assertion` token has been generated.

You can now exchange the authorization code for an access token. The client application will construct a **POST** request to the token endpoint with the parameters below:

Using the parameters above, we can construct the **POST** request. The example below uses curl.



Congratulations!

You have successfully completed the entire Authorization flow! With your new short-lived access token, you can try calling the demo API that you have subscribed earlier in the tutorial to conclude this tutorial.

▶ Try out with Subscribed API

Separate into two paragraphs:

You have successfully completed the entire Authorization flow!

With your new short-lived access token, you can now try calling the demo API that you have subscribed to earlier in the tutorial.