

OIDC Tokens

Generate OIDC Tokens and set them up with Identity Providers.



OIDC Token is a tool that exposes, allocates, accesses, and refreshes tokens from an OIDC-compatible Identity Provider. This is useful for setting up various cloud services to work with Unreal Editor, such as a [Cloud DDC](#). This page provides information on configuring the OIDC token tool and setting up identity providers.

Configuration

The configuration file can be placed under `Engine\Programs\OidcToken\oidc-configuration.json` or `<Game>\Programs\OidcToken\oidc-configuration.json`

The following is an example of what the configuration file could look like:

oidc-configuration.json

```
1
2 {
3
4   "OidcToken": {
5
6     "Providers": {
7
8       "MyOwnProvider": {
9
10        "ServerUri": "https://<url-to-your-provider>",
11
12        "ClientId": "<unique-id-from-provider-usually-guid>",
13
14        "DisplayName": "MyOwnProvider",
15
16        "RedirectUri": "http://localhost:6556/callback", // This needs to match what is configured as the redirect URI for your IdP.
17
18        "PossibleRedirectUri": [
19
20          "http://localhost:6556/callback",
21
22          "http://localhost:6557/callback",
23
24        ], // Set of redirect URIs that can be used. Ports can be in use so it is a good idea to configure a few alternatives. These need to match configuration in IdP.
25
```

```
26 "Scopes": "openid profile offline_access" // These scopes are the basic ones you will need. Some systems may require more and they may
    be named differently.
27
28 }
29
30 }
31
32 }
33
34 }
35
```

 Copy full snippet

Set Up an Identity Provider

This section explains how to set up some common **Identity Providers (IdP)**.

Okta

You need to set up the following items to use Okta as an IdP:

- A **Client** (Application) for **interactive login** (by users).
- A **Custom Auth Server** for control over how Okta maps claims and scopes and some groups that you can assign users to manage access control.
- If you want to run this in a build farm where interactive login is not possible, you need another **Client** (Application) set up to allow for logins.

 Okta does not support OIDC logins without custom auth servers. To set one up, see [Okta's documentation on auth servers](#).

To set up Okta as an identity provider, follow these steps:

1. Open the **Okta admin dashboard**.
2. Click **Applications > Applications**.
3. Create the **Client** for interactive login and enable the following grant types:
 - Refresh Tokens
 - Authorization Code
4. We recommended specifying **several sign-in URLs** under **localhost**. For example:
 - <http://localhost:8749/oidc-token>
 - <http://localhost:8750/oidc-token>
 - <http://localhost:8751/oidc-token>
 - <http://localhost:8752/oidc-token>
 - <http://localhost:8753/oidc-token>
 - <http://localhost:8754/oidc-token>

This makes it possible for the app to run on multiple local ports during the login process, which avoids issues with the port being busy.

 The ports listed above are an arbitrary example. Choose ports that suit your own needs.

1. Set up the **unattended login client**, which is similar to the Client you established in the previous steps, but with the following changes:
 - Use the **Client Credentials** grant type instead.
 - You do not need to specify any sign-in URLs.

This requires you to use **profile objects** with your client. For examples on how to update the created profile, see [Okta's documentation on updating profile attributes](#).

1. Once you create the client credential you want to use for unattended logins, submit a payload for it, similar to the following example:

```
1
2 "profile": {
3
4 "clientCredentialsGroups": [
5
6 "app-ue-storage-project-your-project-name"
7
8 ]
9
10 }
11
```

 Copy full snippet

2. Update your **groups claim** in the custom auth server with the following guidelines:

- You can map groups of users however you want, but you need to create at least one group that accurately includes all users you want to have access. At Epic Games, we usually have one per project.
- You also need to create an **admin group** for users that have some elevated access.
- Use a naming convention that makes it easy to identify the groups you want to use. This helps make sure you only send the groups that apply to Unreal Engine things as part of the tokens and not all groups a user belongs to.
- Make sure to assign at least one user to each group when creating them.

3. In the Okta admin page, configure the custom auth server under **Security > API**.




Custom auth servers are an add-on to Okta that may not be available for you, but it is required for Okta to handle OIDC logins.

To create the auth server, click the **Create Authorization Server** button. This auth server is not specific to Cloud DDC, but rather something you can use for any Unreal Engine services you want to use.

1. Edit the auth server and set up **Access Policies**. Create one policy for each client, and set it to allow that client to log in.
2. Open **Claims** for the Access token type, then set up a **Groups** claim and set it to filter out the groups you want to use, as well as include the `clientCredentialsGroups`. Use this custom expression:

```
1
2 (appuser != null) ? Arrays.flatten( Groups.startsWith("OKTA", "app-ue-", 100) == null ? {} : Groups.startsWith("OKTA", "app-ue-",
3   100) ) : app.profile.clientCredentialsGroups
```

 Copy full snippet

This filters out groups starting with `app-ue`.

1. Set up a **cache_access** scope which you can use for build farm logins.

Once complete the above steps, Okta will be ready to serve as your identity provider.

How to Test Okta as Your Identity Provider

To test Okta, follow these steps:

1. In the auth server, go to the **Token Preview**.
2. Choose the **interactive login** client with **Authorization Code** grant type.
3. Select the user which is assigned to the correct group.

When you preview this token, the JSON it shows should include a **groups** array which contains the names of the groups you have assigned.

Microsoft Entra (AzureAD)

To set up Microsoft Entra as an identity provider, follow these steps:

1. Go to the Microsoft [Azure Portal](#).
2. Click the **Microsoft Entra** service.
3. Go to **App registration** and create a new app registration for the desktop logins. It should have the following settings:
 - Single tenant
 - Contains a set of localhost redirect URIs using the public client/native option. The following are some examples of what your redirect URIs may look like:
 - <http://localhost:8749/oidc-token>
 - <http://localhost:8750/oidc-token>
 - <http://localhost:8751/oidc-token>
 - <http://localhost:8752/oidc-token>
 - <http://localhost:8753/oidc-token>
 - <http://localhost:8754/oidc-token>

Note down the `client id` of this app.

1. Go to **Token Configuration** and add a **groups** claim setting to use **Groups assigned to the application**.
2. Create a new **security group** for the **project user** role. Assign that security group to the role and then add all users you want to the security group.
3. Create the following **app roles**:
 - **Project User** (usually one per project)
 - **Admin**

The Project User role needs to be assignable to users, groups, and applications, while the Admin role is only for users.

1. Go back to **App registration** and create an app for the backend service (for example, "Unreal Cloud DDC").
2. Add an API scope to your new app and name it `user.access`. Assign the `client id` of the desktop app access to this API.
3. Create a new, separate app registration for your cooking apps and add a client secret to them so they can login unattended (or use managed identity or similar if you prefer). This app should also be assigned the project user role.
4. When you create the `oidc-configuration.json` file, you can find the **server uri** to use under the **Endpoints** button for your app registration. It is usually `https://login.microsoftonline.com/<directory-tenant-id>/v2.0`.
5. For **client ID**, use the `client id` of the desktop app you created. The scope needs to contain the API scope you created in the backend service, so it usually ends up being: `offline_access profile openid api://<api scope guid>/user.access`.

Once you have completed the above steps, Microsoft Azure should be ready to serve as your identity provider.