OpenID Connect extends OAuth 2.0. The OAuth 2.0 protocol provides API security via scoped access tokens, and OpenID Connect provides user authentication and single sign-on (SSO) functionality.

|  |  |
| --- | --- |
| [/token](https://developer.okta.com/docs/reference/api/oidc/#token) | Obtain an access and/or ID token by presenting an authorization grant or refresh token. |

|  |  |
| --- | --- |
| [/keys](https://developer.okta.com/docs/reference/api/oidc/#keys) | Return public keys used to sign responses. |

|  |  |
| --- | --- |
| [/.well-known/openid-configuration](https://developer.okta.com/docs/reference/api/oidc/#well-known-openid-configuration) | Return OpenID Connect metadata related to the specified authorization server. |

* ). In OAuth 2.0 terminology, Okta is both the authorization server and the resource server.  
  https://${yourOktaDomain}/oauth2/v1/authorize  
  In the **Sign-in redirect URIs** box, specify the callback location where Okta returns a browser (along with the token) after the user authenticates.   
  When a request is sent to the org authorization server's /authorize endpoint, it validates all requested scopes in the request against the app's grants collection. The scope is granted if it exists in the app's grants collection.  
  **Callback URL**: Define the callback location where Okta returns the token after the user finishes authenticating. This URL must match one of the redirect URIs that you configured in the [Create an OAuth 2.0 app in Okta](https://developer.okta.com/docs/guides/implement-oauth-for-okta/main/#create-an-oauth-2-0-app-in-okta) section.

**Scope**: Use okta.users.read for this example. Include the scopes that allow you to perform the actions on the endpoint that you want to access. Scopes requested for the access token must exist in the app's grants collection, and the user must have permission to perform those actions.  
The available scopes exist in a hierarchy, so that the manage scopes can do everything that the read scopes do, but more. Also, the self scopes only allow for access to the user who authorized the token. For example, a GET request to the /users endpoint with the okta.users.read scope returns all the users that the admin has access to. If the same request is sent with the okta.users.read.self scope, only the current user's account returns.  
Every action on an endpoint that supports OAuth 2.0 requires a specific scope. Okta scopes have the following format: okta.<resource name>.<operation>. For example, you can have resources that are users, clients, or apps with read or manage operations. The read scope is used to read information about a resource. The manage scope is used to create a resource, manage a resource, or delete a resource.

Use the okta.<resource>.read scopes to perform GET API operations. Use the okta.<resource>.manage scopes to perform any GET operations and POST, PUT, and DELETE API operations. The self scopes (okta.<resource>.<operation>.self) only allow access to the user who authorized the token. These scopes are used to perform end user API operations.  
The ID token is a security token granted by the OpenID provider that contains information about an end user. This information tells your client app that the user is authenticated, and can also give you information like their username or location.

You can pass an ID token to different components of your client. These components can use the ID token to confirm that the user is authenticated and also to retrieve information about them.

Access tokens, on the other hand, aren't intended to carry information about the user. They allow access to certain defined server resources. See [Validate access tokens](https://developer.okta.com/docs/guides/validate-access-tokens/).

**OAuth is an open-standard authorization framework that enables third-party applications to gain limited access to user’s data**.  
**Delegation** is a process in which an owner authorizes a service provider to perform certain tasks on the owner’s behalf. Here the task is to provide limited access to another party.

The scopes define the specific actions that apps can perform on behalf of the user. They are the bundles of permissions asked for by the client when requesting a token.

For example, we can share our LinkedIn posts on Twitter via LinkedIn itself. Given that it has write-only access, it cannot access other pieces of information, such as our conversations.

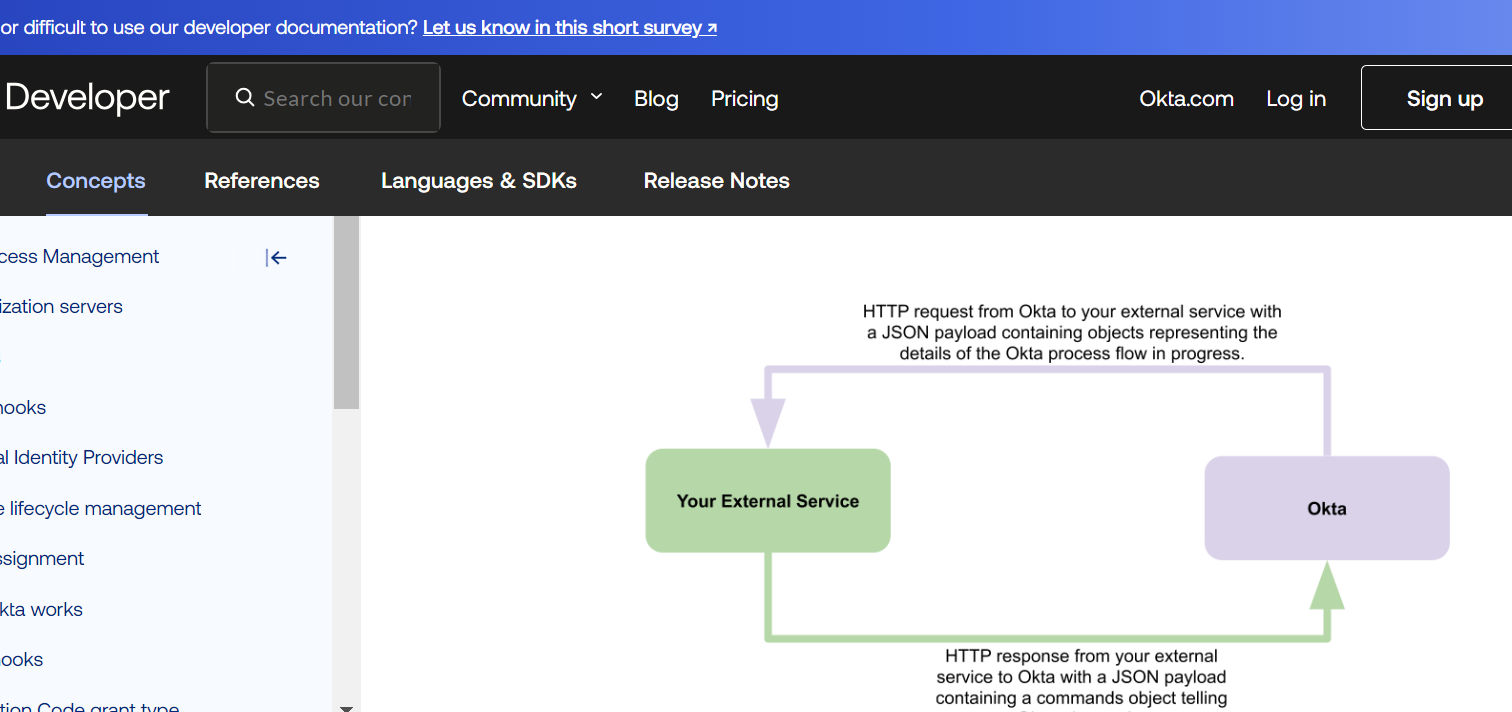
On the Consent screen, a user learns who is attempting to access their data and what kind of data they want to access, and the user must express their consent to allow third-party access to the requested data. You grant access to your IDE, such as CodingSandbox, when you link your GitHub account to it or import an existing repository. The Github account you are using will send you an email confirming this.

**A token is a piece of data containing just enough information to be able to verify a user’s identity or authorize them to perform a certain action.  
A Refresh token** i**s a string issued to the client by the authorization server and is used to obtain a new access token when the current access token becomes invalid.**

They do not refresh an existing access token, they simply request a new one. The expiration time for refresh tokens tends to be much longer than for access tokens. I

<https://medium.com/codenx/oauth-2-0-4cddd6c7471f>  
The Implicit flow was a simplified OAuth flow previously recommended for native apps and JavaScript apps where the access token was returned immediately without an extra authorization code exchange step.

It is not recommended to use the implicit flow (and some servers prohibit this flow entirely) due to the inherent risks of returning access tokens in an HTTP redirect without any confirmation that it has been received by the client.

Keys pass from okta inkline hopok in header to okta  
  
https://blog.postman.com/what-is-pkce/  
PKCE, which stands for “Proof of Key Code Exchange” and is pronounced “pixy,” is an extension of the [OAuth 2.0](https://blog.postman.com/what-is-oauth-2-0/) protocol that helps prevent code interception attacks  
  
When an authorization request is made, the authorization code grant type requires the authorization server to generate an authorization code, which is returned to the client application via a redirect URL. This code can then be exchanged for an access token, which can be used to access the user’s data.  
Before an authorization request is made, the client creates and stores a secret called the “code verifier.” The code verifier is a cryptographically random string that the client uses to identify itself when exchanging an authorization code for an access token.

* The client also generates a code challenge, which is a transformation of the code verifier. The code challenge is sent with the initial authorization request, along with a code challenge method. The code challenge method is the transformation mode used to generate the code challenge. There are two code challenge methods that PKCE supports: plain and S256.  
    
  **Plain:**In the plain mode, the code challenge is equal to the code verifier; nothing changes.
* **S256:**In S256 mode, the SHA-256 hash of the code verifier is encoded using the BASE64URL encoding. The S256 method is recommended by the specification and should be considered before the plain method.  
  Next, the code challenge is securely stored by the authorization server, and an authorization code is returned with the redirect URL as usual. When the client wants to exchange this authorization code for the access token, it sends a request that includes the initial code verifier. The server then hashes the code verifier using SHA-256 (if it has a code challenge method of S256) and encodes the hashed value as a BASE64URL. The corresponding value is then compared to the code challenge. If they match, an access token is issued. Otherwise, an error message is returned.
* This flow ensures that a malicious third-party application cannot exchange an authorization code for an access token, since the malicious application does not have the code verifier. Intercepting the code challenge is also useless because SHA256 is a one-way hashing algorithm and cannot be decrypted.
* When an authorization request is made, the authorization code grant type requires the authorization server to generate an authorization code, which is returned to the client application via a redirect URL. This code can then be exchanged for an access token, which can be used to access the user’s data.
* An attacker can intercept the authorization code that is sent back to the client and exchange it for an access token, which can cause serious data leaks or breaches. One popular method malicious actors use to intercept authorization codes is by registering a malicious application on the user’s device. This malicious application will register and use the same custom URL scheme as the client application, allowing it to intercept redirect URLs on that URL scheme and extract the authorization code:

client send authorize req

https://dev-6951651.okta.com/oauth2/ausf9sf5lf3CO8PaJ5d7/v1/authorize?

response\_type=code&

client\_id=0oaf9snpczNGTIfGv5d7&

state=T3NVZGF4MWt5QjROTUpDU3BrYzVHWXd6RU55akV5anZEQXFqeG90aXB3d1hy%3B%252Fpub%252Fget-quote%253Ftheme%253DDIRC%2526isIscResume%253Dtrue%2526pageno%253D1&

redirect\_uri=http%3A%2F%2Fguvotapcigdev01%3A8080%2Frome%2Flife%2Fpub%2Fget-quote&

scope=openid%20offline\_access&

code\_challenge=r4bxnWO4CxPS\_QM17Wlcmvwj\_Fj-HKsDMtg70kfEM4g&

code\_challenge\_method=S256&nonce=T3NVZGF4MWt5QjROTUpDU3BrYzVHWXd6RU55akV5anZEQXFqeG90aXB3d1hy

response

http://guvotapcigdev01:8080/rome/life/pub/get-quote?

code=6l54pJVjXebUzftjE9HuujYBYygKe7gmaAtBYiWY2dw&

state=T3NVZGF4MWt5QjROTUpDU3BrYzVHWXd6RU55akV5anZEQXFqeG90aXB3d1hy;%252Fpub%252Fget-quote%253Ftheme%253DDIRC%2526isIscResume%253Dtrue%2526pageno%253D1

payload of token

grant\_type: authorization\_code

code: 6l54pJVjXebUzftjE9HuujYBYygKe7gmaAtBYiWY2dw

redirect\_uri: http://guvotapcigdev01:8080/rome/life/pub/get-quote

code\_verifier: OTR1bjFVRmwzN0Zud2VyNktucHU2TkxCflFNbXVJLTBBUmcyLlBHZ2dVLVQu

client\_id: 0oaf9snpczNGTIfGv5d7