StoreFront Cloud end user API Preview

The StoreFront Cloud end user API is an HTTP based API providing access to IT resources such as Windows apps and desktops. It is secured by an OIDC logon flow and OAuth2 access/refresh tokens obtained from the Citrix Identity Platform. Once authenticated, the client (browser or native) can use access tokens to make requests to protected resources on the user's behalf. The API is not intended for worker services and does not include functionality for administrators to manage store configuration.

Using the StoreFront Cloud end user API, you can integrate the functionality of StoreFront Cloud to your own native and web client applications.

Client applications can be written in any language provided it supports HTTP requests and can display HTML in a web view UI control.

Components and Terminology

- **Citrix Authorization Server**: integrates with a variety of IdPs to offer a consistent API experience for authentication and authorization.
- **Discovery**: A Citrix API for finding available services, API endpoints, and relevant metadata.
- **Store**: A set of Citrix APIs for interacting with published resources.

Getting Started

API Pre-requisites

The following items need to be completed prior to using the API:

1. Ensure that you have enabled StoreFront Cloud and created a store. You can access the API using the store's Cloud URL or custom URL.
2. An application ID is required to make API requests to the API. In order to obtain one you have to create an OAuth client.
3. It may take up to 5 minutes after creating an OAuth client before the application ID is accepted by the API.

API usage pattern

At a high level, the client accesses the API by performing calls in the following order:

1. Fetch the discovery configuration, to determine behavior and discover URLs for subsequent operations.
2. Perform authentication, to allow access to resources.
3. List sessions, to find the sessions to re-connect at logon.
4. Perform session reconnect to launch any resources that the user might want to re-connect to.
5. Resource enumeration, to fetch (per-resource) URLs for launch, favorite, icon and operations.
6. Launch and favorite resources in response to user actions in the UI.
7. Client logs off or disconnects from desktop/application sessions and terminates the authenticated session.

> **Notes:**

> - An application ID is required to make API requests to the API. In order to obtain one you have to create an OAuth client.

- Requests **must** contain either the `Citrix-ApplicationId` header, or the `ApplicationId` query parameter with the value set to the above mentioned application ID. The Open API spec guides you on which is required for what endpoints, see the API Exploration page.

Hypermedia Control and Parameterized URLs

This API uses the concept of hypermedia controls, where links to resources and actions are obtained from the server by calling the discovery API. This means that at no time should clients construct URLs (apart from the initial discovery URL). In this document various parameterized URLs are discussed, such as `/resources/{id}`.

It is important to note that these are indicative only, and again, the client code should not construct URLs directly. Instead, all parameterized URLs should be obtained from the API itself. For example, the URL for a particular resource action such as launch, should be extracted from the JSON response returned from the request to `/resources`. In that case, the `icaFileUrl` property would contain the value to use.

In some cases (indicated in the relevant sections), additional parameters can be supplied to a service that is invoked using a HTTP GET request. These parameters are supplied to the service by adding query string parameters to the base endpoint. As stated before, clients should not construct the base endpoint URL themselves. Query string parameters may already be present in the base endpoint - clients should check whether they need to provide the entire query string section, or only add parameters to the existing section.

HTTP POST Parameters

Throughout the API, any request parameters in the POST body must follow standard form URL-encoding rules, with Content-Type set to: `application/x-www-form-urlencoded`.

Fetch the Discovery Configuration

The first step is to call the Discovery API. This returns a JSON document describing various client configuration settings, including a number of URL paths (relative to the customer) that the client must use for subsequent API calls. To find out more information about the returned properties read the Open API Spec response schemas, described in the API Exploration page.

For example (replace '{CUSTOMER_DOMAIN}' with either your `cloud.com` domain or configured custom domain). Only relevant details are shown:

curl https://{CUSTOMER_DOMAIN}/citrixapi/discovery/configurations --header "Citrix-ApplicationId: {APPLICATION_ID}"

Response:


```
{
    "services": [
        {
            "service": "store",
            "endpoints": [
                {
                    "id": "ListAvailableSessions",
                    "url": "https://{CUSTOMER_DOMAIN}/citrixapi/store/sessions",
                    "capability": [
                        "ListAvailableSessions"
                    ]
                },
                {
                    "id": "ListResources",
                    "url": "https://{CUSTOMER_DOMAIN}/citrixapi/store/resources",
                    "capability": [
                        "ListResources"
                    ]
```

```
                },
                {
                    "id": "GetDetectionTicket",
                    "url": "https://{CUSTOMER_DOMAIN}/citrixapi/store/
client/detectionTicket",
                    "capability": [
                        "GetDetectionTicket"
                    ]
                },
                {
                    "id": "GetDetectionStatus",
                    "url": "https://{CUSTOMER_DOMAIN}/citrixapi/store/
client/detectionStatus",
                    "capability": [
                        "GetDetectionStatus"
                    ]
                }
            ],
            ...
        }
    ],
    "authScopes": [
        "wsp"
```

```
    ],
    "clientSettings": {
        "customerDomain": "{CUSTOMER_DOMAIN}",
        "customerId": "{CUSTOMER_ID}",
        "authDomain": "{AUTH_DOMAIN}",
        "lockedDown": true,
        "webUiCapabilities": "MobileReceiverReady",
        "products": [...],
        "storeGuid": "{GUID}",
        "oidcConfiguration": {
            "oidc_discovery_endpoint": "https://accounts.cloud.com/
core/.well-known/openid-configuration",
            "oidc_clients": {...}
        },
        "serviceRecordId": "{SERVICE_RECORD_ID}",
        "inactivityTimeoutInMinutes": "1440",
        "inactivityTimeoutInMinutesMobile": null,
        "acr_values": "{ACR_VALUES}",
        "promptLoginEnabled": true,
        "disallowICADownload": false
    }
```

```
}
```

Throughout this documentation, endpoint URLs are representative. You should always use the URLs defined by the Configuration JSON.

Authentication

Some endpoints are protected and require an OAuth access token to allow access. If the required token is not present, or you include a token but it has expired, the API returns a response with HTTP status code 400 and a response body.

For example if the first call is to resource enumeration and you don't yet have a token:

```
curl https://{CUSTOMER_DOMAIN}/citrixapi/store/resources --header "Citrix-ApplicationId: {APPLICATION_ID}"
```

Response:

```
{

    "error": "invalid_grant",

    "error_description": "Access token is invalid."

}
```

To obtain an access token and find out more about the API authentication, read our authentication guide.

List Sessions

To get a list of the user's sessions, use the returned URL provided in the discovery configuration document with ID `ListAvailableSessions`.

The list includes all of the user's sessions: sessions that are disconnected, and sessions that are connected on any device, including the current device. See Determining whether to re-connect a session for how to decide which sessions to re-connect.

The `clientName` and `deviceId` query parameters are optional. When provided, they are included in the session re-connect URLs returned in the response. Read more about client names here.

For example (assumes you have stored the access token in a variable called 'token'):

```
curl https://{CUSTOMER_DOMAIN}/citrixapi/store/sessions?
clientName={CLIENT_HOSTNAME}&deviceId={CLIENT_HOSTNAME} --header "Citrix-
ApplicationId: {APPLICATION_ID}" --header "Authorization: Bearer $token"
```

Response (simplified for brevity):

```
[

    {

        "sessionId": "{SESSION_ID}",

        "icaFileFetchTicketUrl": "https://{CUSTOMER_DOMAIN}/citrixapi/
store/sessions/{SESSION_ID}/icaFileFetchTicket?
clientName={CLIENT_HOSTNAME}&deviceId={CLIENT_HOSTNAME}",

        "icaFileUrl": "https://{CUSTOMER_DOMAIN}/citrixapi/store/sessions/
{SESSION_ID}/icaFile?clientName={CLIENT_HOSTNAME}
&deviceId={CLIENT_HOSTNAME}",

        "userSessionType": "Application",

        "connectionState": "disconnected",

        "sessionStartTime": "{SESSION_START_TIME}",

        "clientName": "{OTHER_CLIENT_HOSTNAME}",

        "deviceId": "{OTHER_DEVICE_ID}",

        "applications": [

            {

                "id": "{APP_ID}",
```

```
                "name": "Calculator Cloud",

                "imageUrl": "https://{CUSTOMER_DOMAIN}/citrixapi/store/
resources/images/{IMAGE_HASH}"

            }

        ],

        ...

    }

]
```

The endpoints returned in the response can be used to perform session re-connection.

It is possible that no sessions are available, and an empty response gets returned:

```
[]
```

Determining whether to re-connect a session

Use the `connectionState` and `deviceId` properties of each session to decide whether to re-connect to it:

- **`disconnected`**: the session is still running, but no device is connected to it. The session can be re-connected.
- **Any other value, for example `connected`**: the session is connected to a device. Compare its `deviceId` with the device ID of the current client (or its client name, if the client does not send a device ID):
  - If they match, the session is already connected on the current device and does not need to be re-connected.
  - If they differ, the session is connected on another device. Re-connecting to it moves the session to the current device.

A session can end between being listed and being re-connected. In that case the re-connect request returns `404 Not Found`, and the client should list the sessions again.

Session re-connect

To perform a session re-connect on one of the above sessions, make a GET request to the `icaFileUrl`.

For example:

```
curl -X GET 'https://{CUSTOMER_DOMAIN}/citrixapi/store/sessions/
{SESSION_ID}/icaFile?clientName={CLIENT_HOSTNAME}
&deviceId={CLIENT_HOSTNAME}&clientAddress={CLIENT_IP_ADDRESS}' --header
"Citrix-ApplicationId: {APPLICATION_ID}" --header "Authorization: Bearer
$token"
```

As outlined here we have set the client name and device id as the client computers hostname. This is used later during session disconnect.

If the session is ready to launch then the API request returns an ICA file with content type `application/x-ica`, for example:

```
[Encoding]

InputEncoding=UTF8



[WFClient]

ProxyFavorIEConnectionSetting=Yes

ProxyTimeout=30000

ProxyType=Auto

ProxyUseFQDN=Off

RemoveICAFile=yes

TransportReconnectEnabled=On

<Rest of ICA file content, omitted for brevity>
```

Clients typically load the `icaFileUrl` URL into an iframe to obtain an ICA file and trigger a launch via file type association. The HDX client uses the information in the ICA file to create a session to the resource.

Resource Enumeration

To get a list of resources available to the user, use the returned URL provided in the discovery configuration document with ID `ListResources`.

The response body is a JSON object that includes an array of resources and links to perform operations on those resources.

For example:

```
curl https://{CUSTOMER_DOMAIN}/citrixapi/store/resources --header "Citrix-
ApplicationId: {APPLICATION_ID}" --header "Authorization: Bearer $token"
```

Response (simplified for brevity):

```
{

    "resources": [

        {

            "resourceId": "{RESOURCES_ID}",

            "id": "{ID}",

            "name": "Microsoft Word",

            "links": {

                "resourceDetailsUrl": "https://{CUSTOMER_DOMAIN}/
citrixapi/store/resources/{RESOURCES_ID}",

                "imageUrl": "https://cdn.azureedge.net/
{RESOURCES_ID}.png",

                "icaFileUrl": "https://{CUSTOMER_DOMAIN}/citrixapi/store/
resources/{RESOURCES_ID}/icaFile",
```

```
                "icaFileFetchTicketUrl": "https://{CUSTOMER_DOMAIN}/
citrixapi/store/resources/{RESOURCES_ID}/icaFileFetchTicket",

                "favoriteUrl": "https://{CUSTOMER_DOMAIN}/citrixapi/store/
resources/{RESOURCES_ID}/$favorite",

                "unfavoriteUrl": "https://{CUSTOMER_DOMAIN}/citrixapi/
store/resources/{RESOURCES_ID}/$unfavorite"

            },

            "path": "\\Office Apps\\",

            "disabled": false,

            "keywords": [],

            "clientTypes": [

                "ica30",

                "rdp"

            ],

            "resourceType": "Citrix.MPS.Application",

            "properties": [],

            "playsFileTypes": [],

            "mandatory": false,

            "recentsPosition": 4,

            "isFavorited": false

        },

        ...

    ]
```

```
}
```

For each resource, this includes the operation endpoints that can be used to perform actions such as launch the resource.

Launch Resource

To launch the resource, make a GET request to the `icaFileUrl` endpoint returned by the response above.

For example:

```
curl https://{CUSTOMER_DOMAIN}/citrixapi/store/resources/{RESOURCES_ID}/
icaFile?clientName={CLIENT_HOSTNAME}&deviceId={CLIENT_HOSTNAME}
&clientAddress={CLIENT_IP_ADDRESS} --header "Citrix-ApplicationId:
{APPLICATION_ID}" --header "Authorization: Bearer $token"
```

As outlined here we have set the client name and device id as the client computers hostname. This is used later during session disconnect.

If the session is ready to launch then the API request returns an ICA file with content type `application/x-ica`, for example:

```
[Encoding]

InputEncoding=UTF8



[WFClient]

ProxyFavorIEConnectionSetting=Yes

ProxyTimeout=30000

ProxyType=Auto

ProxyUseFQDN=Off

RemoveICAFile=yes
```

```
TransportReconnectEnabled=On

<Rest of ICA file content, omitted for brevity>
```

The ICA file contents can then be saved locally to disk in a temporary file and opened to trigger a HDX launch. More information about the supported launch flows can be found here.

Disconnect and Logoff

The client should give the ability to disconnect and logoff from active sessions.

Disconnect

To disconnect an active session, use the Resources Disconnect API.

Once disconnected, the client can later re-connect by calling List Sessions.

For example:

```
curl -X POST 'https://{CUSTOMER_DOMAIN}/citrixapi/store/sessions/
$disconnect-by-device' --header "Authorization: Bearer $token" --header
"Citrix-ApplicationId: {APPLICATION_ID}" --header "Content-Type:
application/json" --data '{"clientName": "{CLIENT_HOSTNAME}", "deviceId":
"{CLIENT_HOSTNAME}"}'
```

The `CLIENT_HOSTNAME` must match what was sent during session re-connect and resource launch.

The response has a status code of `200 OK` if all sessions are fully disconnected or `206 partial success` which means sessions are being disconnected in the background.

The indication that this has worked is that all active launched sessions on the current device are disconnected.

Logoff

The session logoff endpoint can be used to close all active sessions associated with the given device.

For example:

```
curl -X POST 'https://{CUSTOMER_DOMAIN}/citrixapi/store/sessions/$logoff-by-device' --header "Citrix-ApplicationId: {APPLICATION_ID}" --header "Authorization: Bearer $token" --header "Content-Type: application/json" --data '{"clientName": "{CLIENT_HOSTNAME}", "deviceId": "{CLIENT_HOSTNAME}"}'
```

The `CLIENT_HOSTNAME` must match what was sent during session re-connect and resource launch.

The response has a status code of `200 OK` if all sessions are logged off successfully or `206 partial success` which means sessions are being logged off in the background.

Disconnect or Logoff Specific Sessions

The Disconnect and Logoff endpoints above act on all sessions associated with a device. To disconnect or log off individual sessions instead, use the URLs provided in the discovery configuration document with IDs `DisconnectUserSessions` and `LogOffUserSessions`, passing the `sessionId` values returned by List Sessions.

For example, to disconnect two sessions:

```
curl -X POST 'https://{CUSTOMER_DOMAIN}/citrixapi/store/sessions/$disconnect-by-id' --header "Authorization: Bearer $token" --header "Citrix-ApplicationId: {APPLICATION_ID}" --header "Content-Type: application/json" --data '{"sessionIds": ["{SESSION_ID_1}", "{SESSION_ID_2}"]}'
```

To log off sessions, send the same request body to the `LogOffUserSessions` URL, for example `https://{CUSTOMER_DOMAIN}/citrixapi/store/sessions/$logoff-by-id`.

The response contains a result for each session:

```
{

    "results": [

        {

            "sessionId": "{SESSION_ID_1}",

            "isSuccess": true,
```

```
            "failureReason": null
        },
        {
            "sessionId": "{SESSION_ID_2}",
            "isSuccess": false,
            "failureReason": "{FAILURE_REASON}"
        }
    ]
}
```

The request can succeed for some sessions and fail for others, so check `isSuccess` for each session. If `sessionIds` is missing or empty, contains the same session ID more than once, or contains an invalid session ID, the API returns `400 Bad Request`.

Authentication

Some end-points are protected and require authentication information to allow access. If the required authentication information is not present, then the API indicates this with a `400 Bad Request` response, with the reason stated as **invalid_grant**.

Authentication is performed using OpenID Connect which when complete enables the calling client to request OAuth2 access tokens to use the API. The client should be prepared to respond to a invalid_grant response at any point, and in particular both Enumeration and Launch requests may require the client to obtain fresh authentication information.

When building your own application, it is recommended to use a session-based Token Management Service, which would manage duration of and access to OAuth 2.0 access tokens. See our provided example here.

To find out more about OAuth, the flows and core concepts, visit the official RFC.

Citrix Authorization Server

The Citrix Authorization Server endpoint configuration can be found in your customer discovery configurations document as shown in the getting started guide, under `clientSettings` - `oidcConfiguration` - `oidc_discovery_endpoint`.

Performing a `GET` request on the `oidc_discovery_endpoint` provides you with the authorization server configuration, specifically this authentication guide we are interested in the `authorization_endpoint` and the `token_endpoint`.

For the purposes of this document are using the following values:

- `authorization_endpoint`: `<https://accounts.cloud.com/core/connect/authorize>`
- `token_endpoint`: `<https://accounts.cloud.com/core/connect/token>`

Obtaining an Access Token

Once you've created your client, you can then try to obtain the Access Token to use in the APIs to perform actions such as resource enumeration, launching and favoriting.

Your client application should perform an OAuth Authorization Code Flow with Proof Key for Code Exchange (PKCE) as described below. You can refer to our client examples to see how we've achieved that in .NET. There are many client libraries in your language of choice for performing the OAuth flow. We also support the standard OAuth Authorization Code Flow but only for Private Clients.

Citrix's access token is valid only for 30 minutes for security reasons. If you'd like to get a new one without user interaction, see Refresh Tokens.

Refresh tokens (offline access)

Conditions for getting a Refresh Token:

1. The client being used has Offline Access enabled, configured during client creation
2. During the Authorization Code Flow with PKCE the scope `offline_access` is sent

If the conditions are met, a Refresh Token is provided alongside an Access Token.

The Refresh Token can be used to get a new Access Token without user interaction:

1. The client application makes a POST request to the Citrix Authorization Server token endpoint `<https://accounts.cloud.com/core/connect/token>` with the following information in the request body, and a Content Type set to `application/x-www-form-urlencoded`:

FORM FIELD	FORM VALUE	DESCRIPTION
client_id	*Your client ID*	See Client Management
client_secret	*Your client secret*	Optional, only required for Private clients
grant_type	refresh_token	Indicates that a refresh request is in progress
refresh_token	*Your refresh token*	The refresh token that is generated during the Authorization Code Flow

Raw example:

```
POST https://accounts.cloud.com/core/connect/token

Content-Type: application/x-www-form-urlencoded

client_id={CLIENT_ID}&client_secret={CLIENT_SECRET}
&grant_type=refresh_token&refresh_token={REFRESH_TOKEN}
```

2. The Citrix Authorization Server responds with a new Access Token and Refresh Token.

The lifetime of Citrix's refresh token is 24 hours.

Example of OAuth Authorization Code Flow

To perform the standard Authorization Code Flow, refer to the example for Authorization Code Flow with PKCE and ignore the `code_verifier`, `code_challenge` and `code_challenge_method` specific steps and properties from API calls.

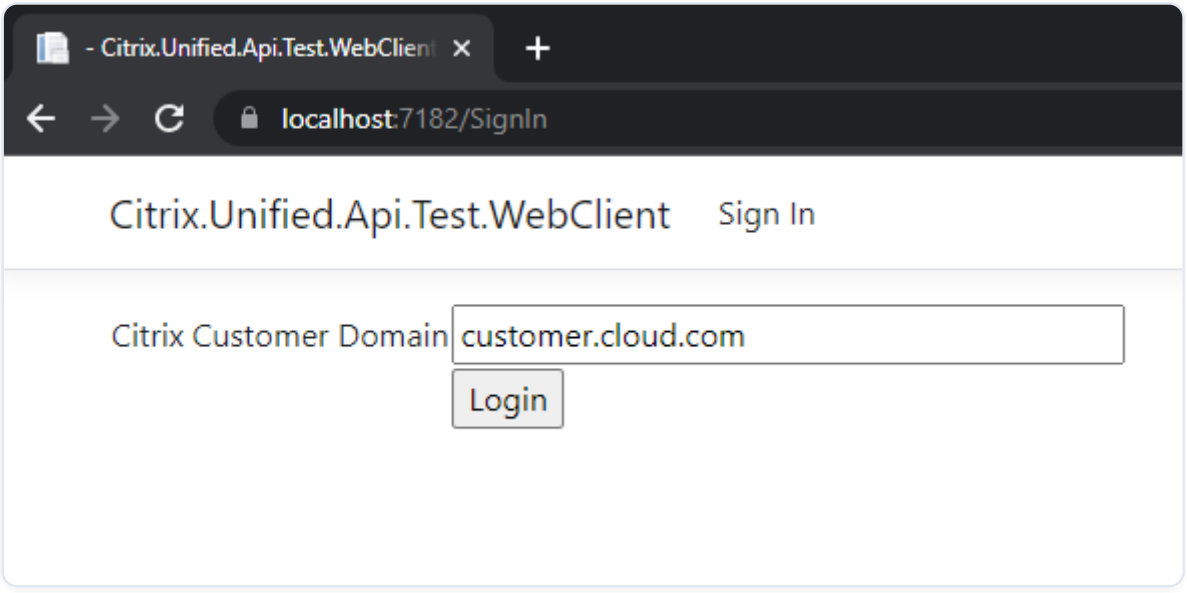
We recommend using the Authorization Code Flow with PKCE for the highest level of security.

Example of OAuth Authorization Code Flow with Proof Key for Code Exchange (PKCE)

For this example, we are going to use the sample WebClient application.

This is the recommended flow for the highest level of security.

1. The user enters their customer URL in the login box and clicks 'Login'



2. The client SDK creates a one-time cryptographically random 128-byte `code_verifier` and uses this to generate a `code_challenge` (Base64 URL encoded string containing the SHA-256 digested form of the `code_verifier`). The code verifier must have a minimum length of 43 characters and a maximum length of 128 characters. See the OAuth spec for more details: `<https://www.rfc-editor.org/rfc/rfc7636#section-4.1>`.
3. The client SDK redirects the user to Citrix Authorization Server via a GET request to `<https://accounts.cloud.com/core/connect/authorize>` and sends various bits of information in the QueryString:

QUERY FIELD	QUERY VALUE	DESCRIPTION
client_id	*Your client ID*	See Client Management
redirect_uri	`<https://localhost:7182/callback>`	This is the callback of the locally running WebClient example
response_type	code	Specifies that we are using the Authorization Code Flow
scope	openid wsp spa leases offline_access	The values 'openid wsp spa leases' are required, if you selected Offline Access during client creation also include 'offline_access'
code_challenge	*Your Code Challenge*	As mentioned above, encoded SHA-256 digested form of the `code_verifier`
code_challenge_method	S256	Must be `S256` to indicate we are using PKCE SHA-256 Signature
prompt	Login	Indicates to the Authorization Server we should prompt the end user for re-authentication.

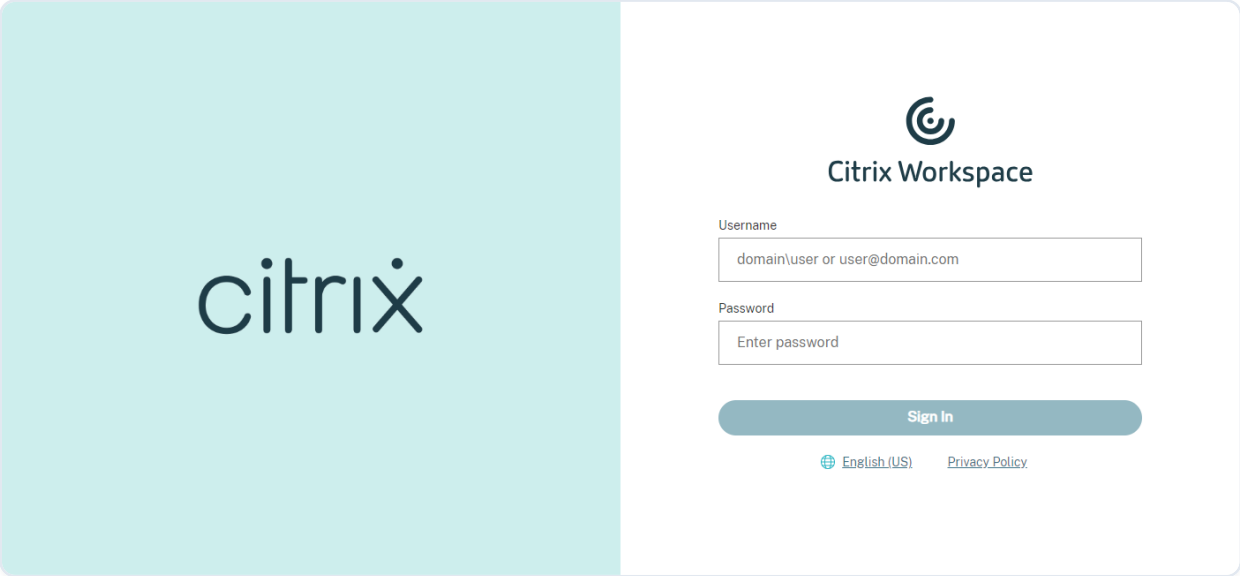
QUERY FIELD	QUERY VALUE	DESCRIPTION
response_mode	form_post	Indicates that the response parameters are encoded in the body using `application/x-www-form-urlencoded` format.
acr_values	*Customer ACR Values*	Citrix specific value for additional validation purposes, can be obtained via the customer discovery configuration
state	*Your State Value*	Randomly generated string, used to prevent CSRF attacks.

Note: The WebClient C# library also adds a cryptographic number-used-only-once (nonce) value, along with a state.

Raw example:

> GET https://accounts.cloud.com/core/connect/authorize?client_id={CLIENT_ID}
> &redirect_uri={REDIRECT_URI}
> &response_type=code&prompt=Login&scope=openid%20wsp%20spa%20leases%20offline_access&code_challenge={CODE_CHALLENGE}
> &code_challenge_method=S256&response_mode=form_post&acr_values={ACR_VALUES}&state={STATE}

4. Citrix Authorization Server redirects the user to the login screen, the previous request results in a `302 Found` response with the Location header set to the login screen:



5. After authentication, the end user might see a consent page depending on the OAuth Client configuration
6. Upon successful login, Citrix Authorization Server stores the `code_challenge` and redirects the user back to the application (WebClient example) with an `authorization code` valid for one-time use
7. The `state` value provided during the authorization call is returned in the `/callback?...` query string and body, which must be compared against the sent `state` to protect from CSRF attacks
8. The client SDK then calls the `token` endpoint (https://accounts.cloud.com/core/connect/token) and sends the `authorization code` and `code_verifier` created in step 2 along with client information with the Content Type `application/x-www-form-urlencoded`:

FORM FIELD	FORM VALUE	DESCRIPTION
client_id	*Your client ID*	See Client Management
client_secret	*Your client secret*	Optional, only required for Private clients
code	*Your authorization code*	The `authorization code` sent back to the client in step 6
grant_type	authorization_code	Tells the server we are performing the authorization code flow

FORM FIELD	FORM VALUE	DESCRIPTION
redirect_uri	`<https://localhost:7182/callback>`	This is the callback of the locally running WebClient example
code_verifier	*Your code verifier*	The `code_verifier` created in step 2

Raw example:

> POST https://accounts.cloud.com/core/connect/token
>
> Content-Type: application/x-www-form-urlencoded
>
> client_id={CLIENT_ID}&client_secret={CLIENT_SECRET}
> &code={AUTHORIZATION_CODE}
> &grant_type=authorization_code&redirect_uri={REDIRECT_URI}
> &code_verifier={CODE_VERIFIER}

9. Citrix Authorization Server validates the `code_challenge` and `code_verifier`
10. Citrix Authorization Server responds with the `access token` (and optionally a `refresh token` if your client has Offline Access and you sent `offline_access` in the scopes during step 3)
11. The WebClient application can then call APIs such as Resources

Authentication Errors

CORS Errors

If you are attempting to perform the OAuth authentication steps directly in a web application running in the browser, and you see CORS errors with APIs such as:

- `<https://accounts.cloud.com/core/.well-known/openid-configuration>`
- `<https://accounts.cloud.com/core/connect/authorize?client_id=....>`

This occurs when using the public client from a web browser. In this scenario it is necessary to use the private client.

400 Bad Request

There are some potential errors that could occur during the authentication flows and while making requests to the APIs. We closely follow the OAuth RFC error responses and a `400 Bad Request` error is returned and includes more details in the response body.

For example, a `400` response is returned when trying to call an API without a valid Access Token:

```
{

    "error": "invalid_grant",

    "error_description": "Access token is invalid."

}
```

Possible reason for the `400 Bad Request` response:

- Missing Access Token
- Invalid or expired Access/ Refresh token
- Client is invalid
- Scope is invalid

Unable to Process Your Request

The following error page could be returned by the Citrix Authorization Server during the authentication flows:



Likely reasons for this page being shown:

- Client is disabled
- The redirect URI being sent during the authorization code flow is not configured as an 'Allowed redirect URI' on the client
- The client is trying to send an `offline_access` scope during the authorization code flow and offline access was not configured on the client
- The ACR values sent are not correct for the customer, ensure you follow the discovery instructions to find your specific customer ACR values

If the error persists, please contact Citrix Support and provide the Transaction ID.

OAuth Client Management

Application ID

An application ID is required to make requests to the API, in order to obtain one you have to create an OAuth client as described on this page.

API requests **must** contain either the `Citrix-ApplicationId` header, or the `ApplicationId` query parameter with the value set to the above mentioned application ID. The Open API spec guides you on which is required for what endpoints, see the API Exploration page.

OAuth Client Registration

To perform the OIDC authentication steps, it is necessary to request a Client.

Depending on the type of application you want to build either a public or private (confidential) client needs to be created.

Public Client

For native applications that handle the authorization code flow with PKCE and store the access and refresh token in memory, you need to request a public client as the application won't be able to securely store the private client secret.

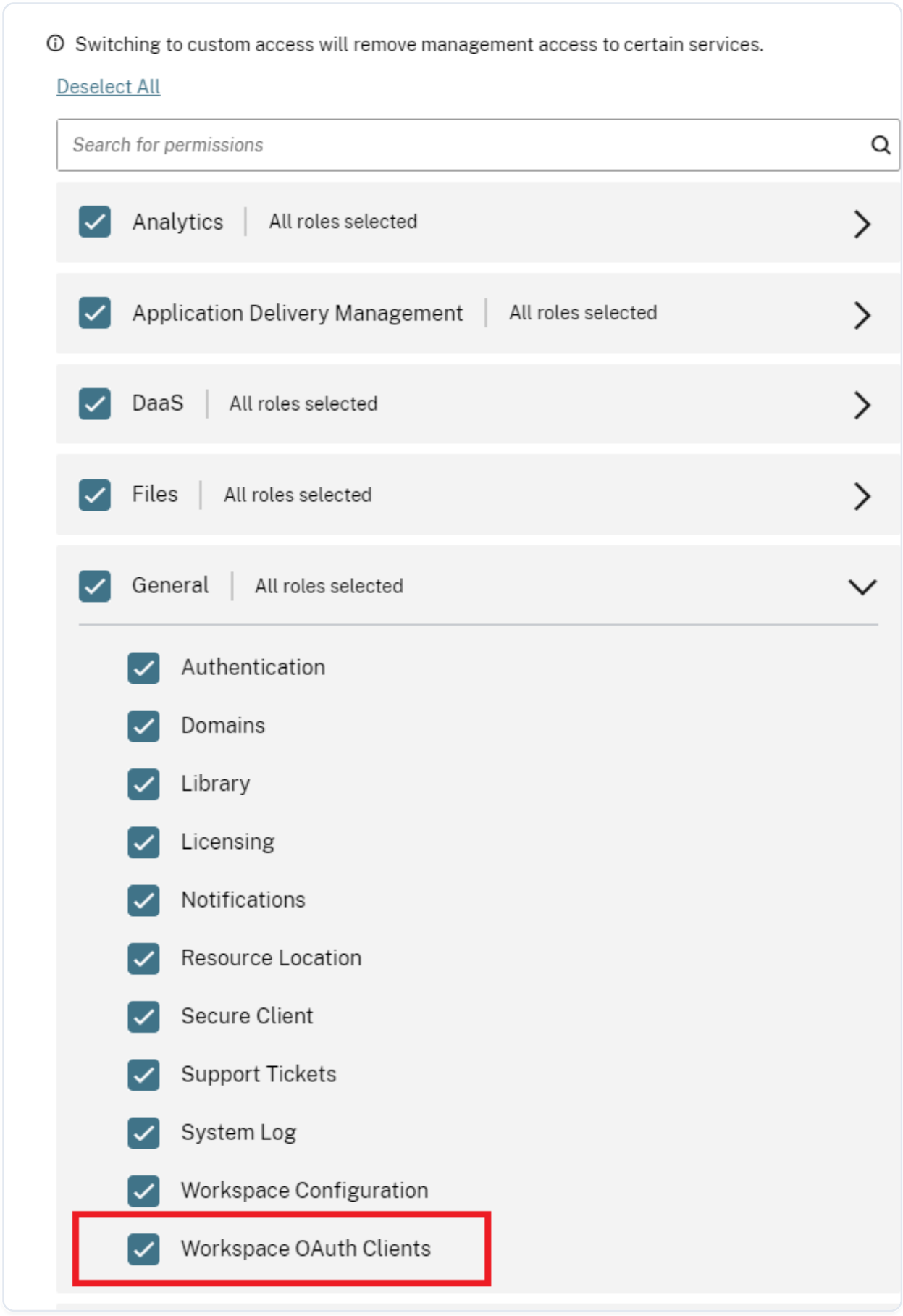
For web applications running in a browser, it is not possible to use a public client due to CORS. Instead you must use a private client and create a backend service using a pattern such as token mediating backend (TMB) or backend for frontend (BFF).

Private Client

For server side web clients - including the example Token Management Service - you should request a private client. The private client supports the standard authorization code flow or the authorization code flow with PKCE.

Creating a Client

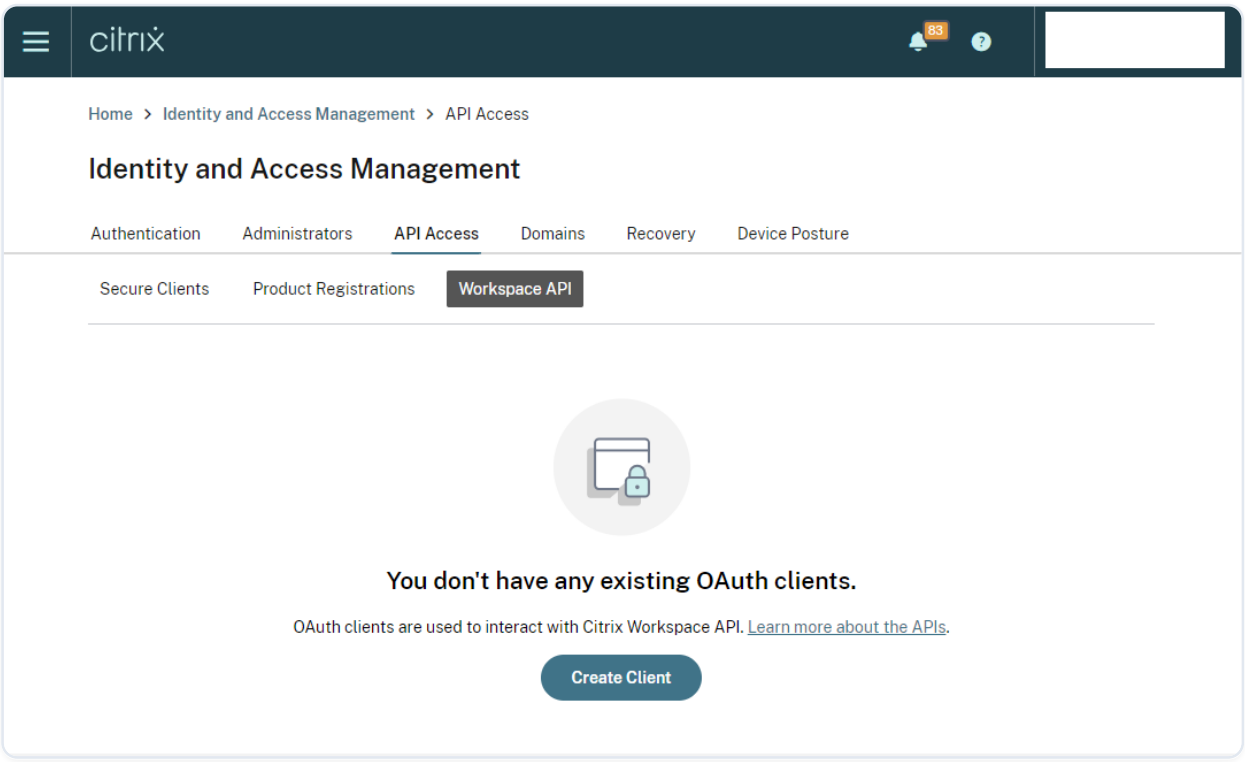
1. Login to the Citrix Cloud Admin Portal and select the customer you'd like to register a client for. The account you login with needs to be either be a 'Full' admin, or a custom admin with the 'Workspace OAuth Clients' permission selected.


Switching to custom access will remove management access to certain services.
Deselect All
Search for permissions
Analytics
All roles selected
Application Delivery Management
All roles selected
DaaS
All roles selected
Files
All roles selected
General
All roles selected
Authentication
Domains
Library
Licensing
Notifications
Resource Location
Secure Client
Support Tickets
System Log
Workspace Configuration
Workspace OAuth Clients

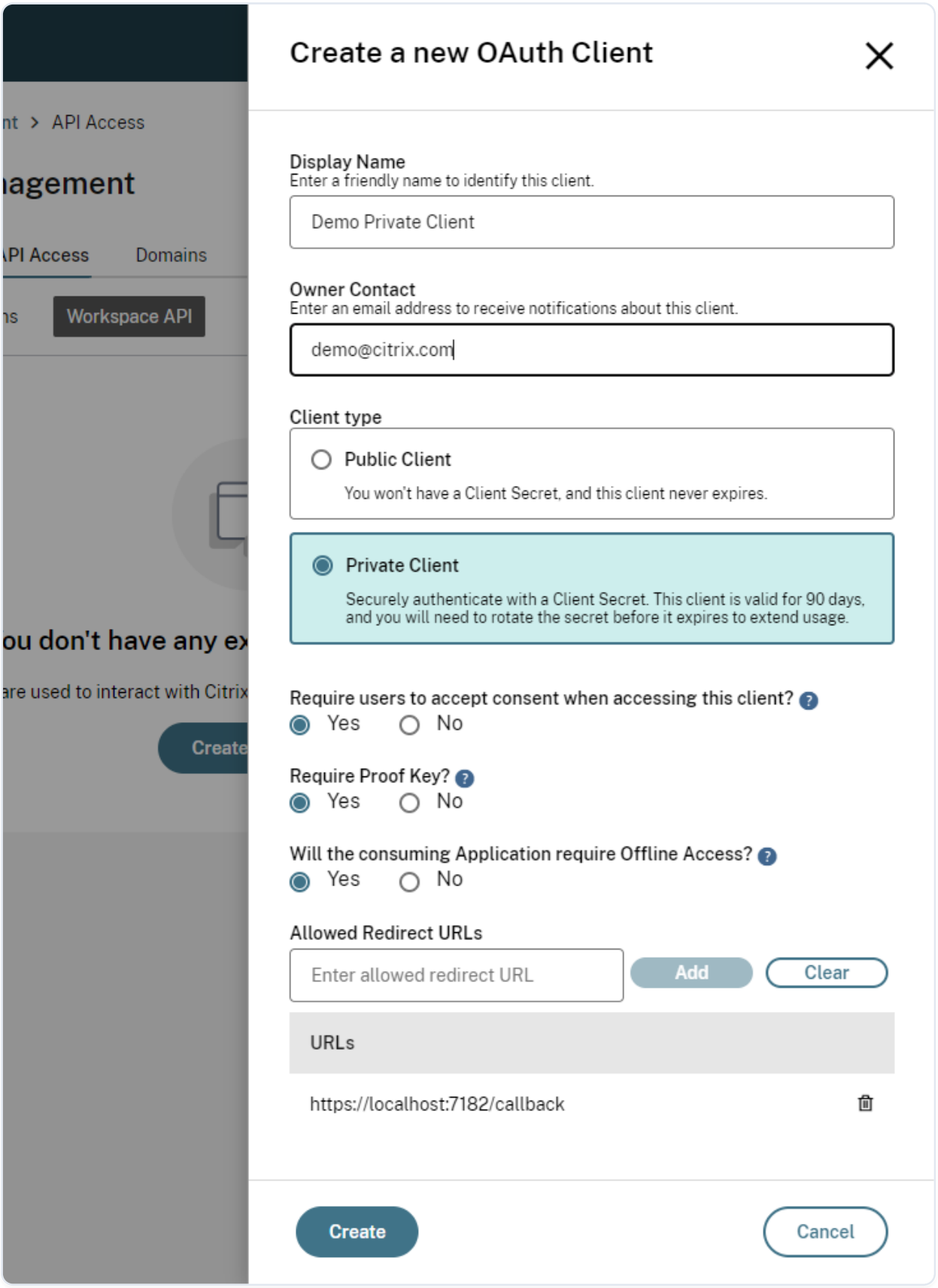
Warning:

> It is recommended to assign the **Workspace OAuth clients** custom role with caution. The access privileges associated with this role might enable administrators to access end user's resources (VDAs or applications) on the Workspace platform. It is also important to note that administrators with **Full access** automatically have access permissions equivalent to that of an administrator with the **Workspace OAuth clients** permission.

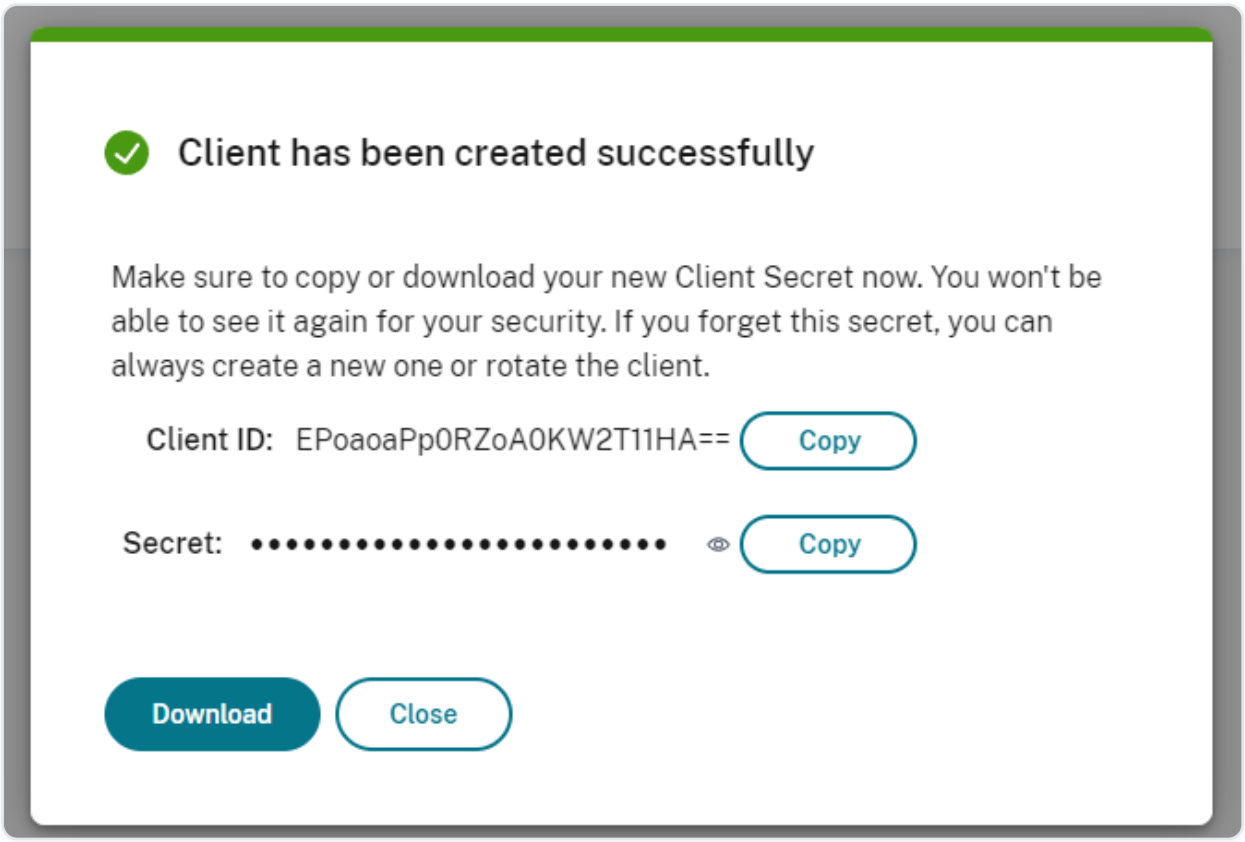
2. Navigate to **Identity and Access Management** > **API Access** > **StoreFront Cloud API**



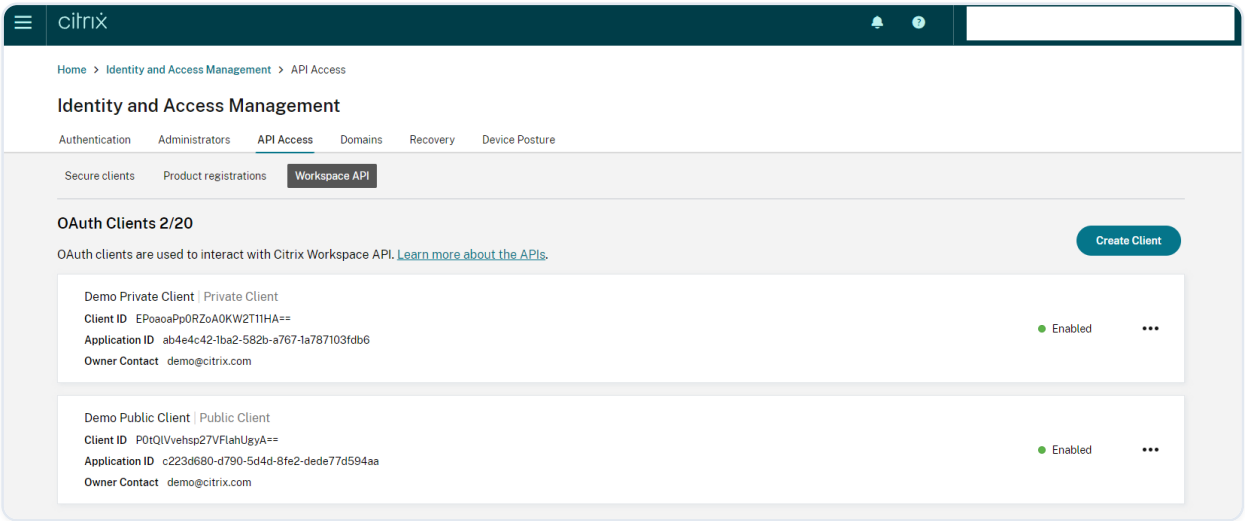
3. Select 'Create Client' and fill out the required details, here is an example that works with the sample applications:



4. For a Private Client, once you click 'Create' a modal appears with the Client ID and Secret for you to download and copy:



5. You are able to see all your created clients on the page:



Client Validity and Rotation Requirements

For Private Clients only.

Client secrets are valid for a period of 90 days and need rotating before this elapses. Please see instructions on rotating the client secret below.

Before undertaking development that relies on your client secret being valid, ensure that you are able to rotate, update applications using, and expire your client secret using the guide.

OAuth 2.0 Client Rotation

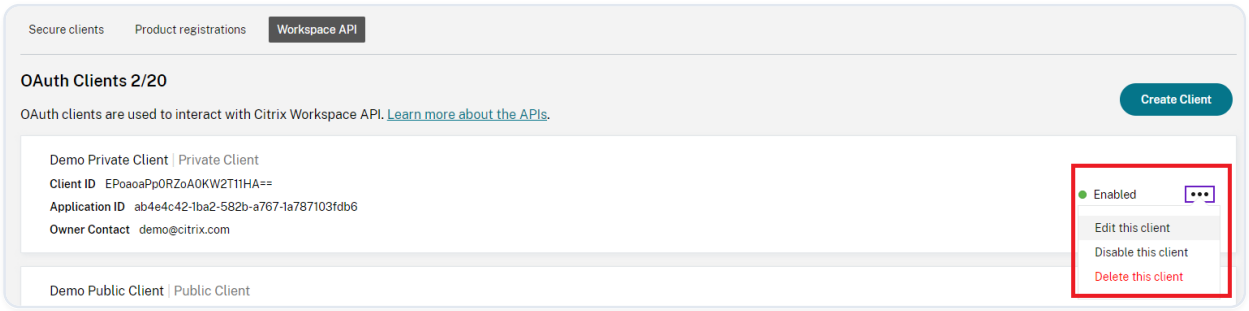
OAuth 2.0 private clients are valid for a period of 90 days. The validity period ensures that the owner of the client follows recommended security practices.

A reminder email is automatically sent to the 'Owner Contact' when the client gets close to expiry.

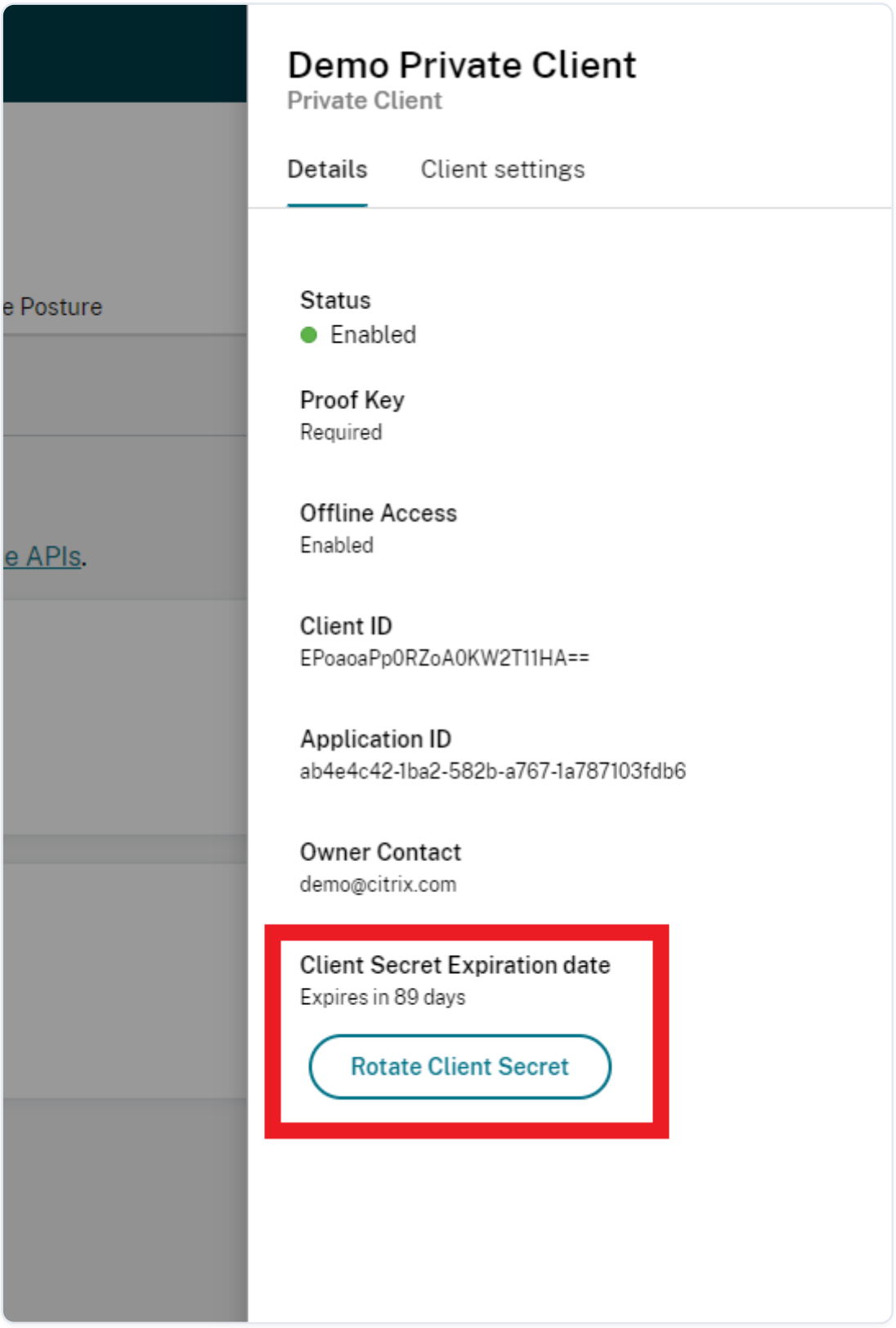
To rotate the client:

Manual Rotation

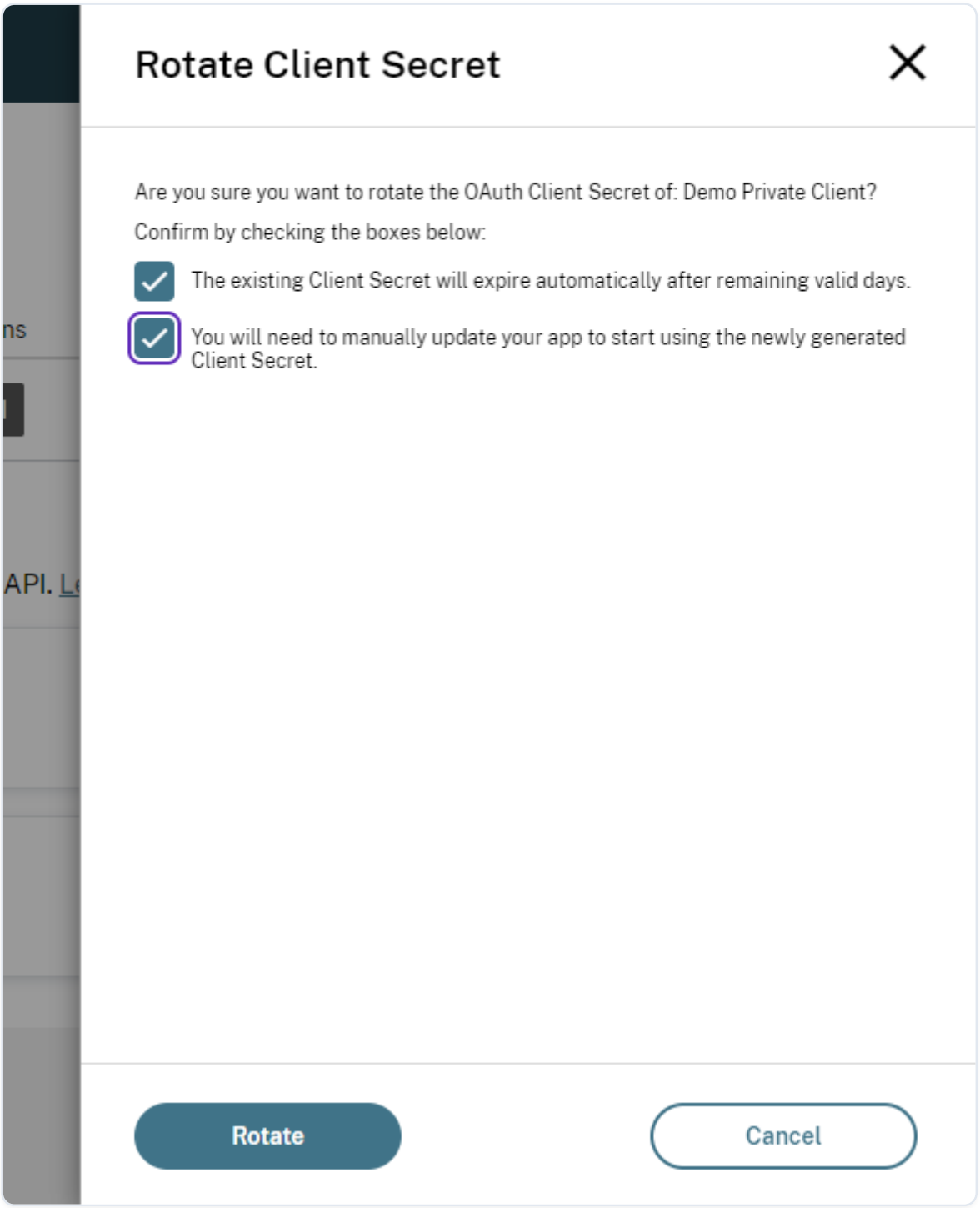
1. Login to Citrix Cloud Admin Portal and navigate to **Identity and Access Management** > **API Access** > **StoreFront Cloud API**.
2. Find the client you want to rotate in the client list (a warning is shown when a client is close to expiry), click on the three dots, and select 'Edit this client':



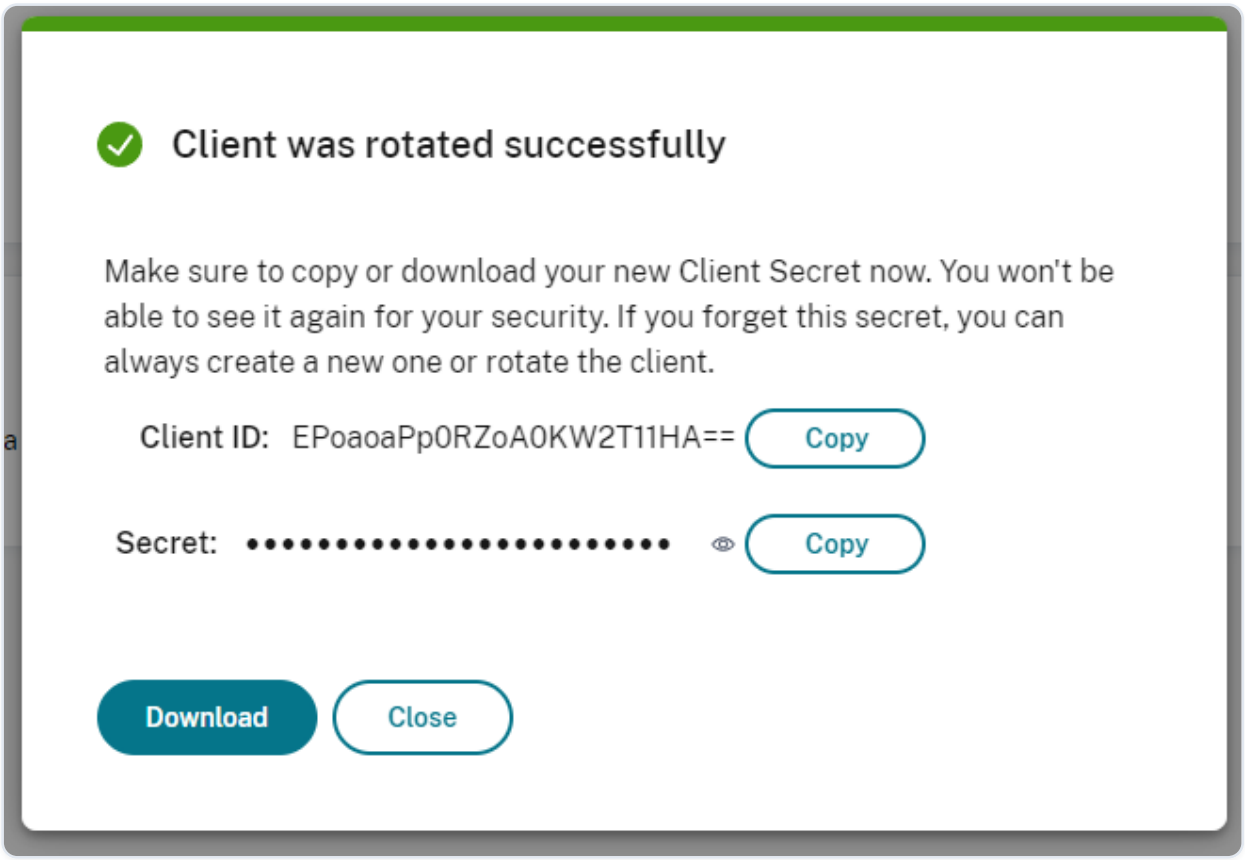
3. Here you are able to see the client details and perform a few operations, here we are interested in the 'Client Secret Expiration date' and the 'Rotate Client Secret' button:



4. Clicking the 'Rotate Client Secret' button shows you a page giving some more information on client rotation. Once agreeing to the terms, click 'Rotate':

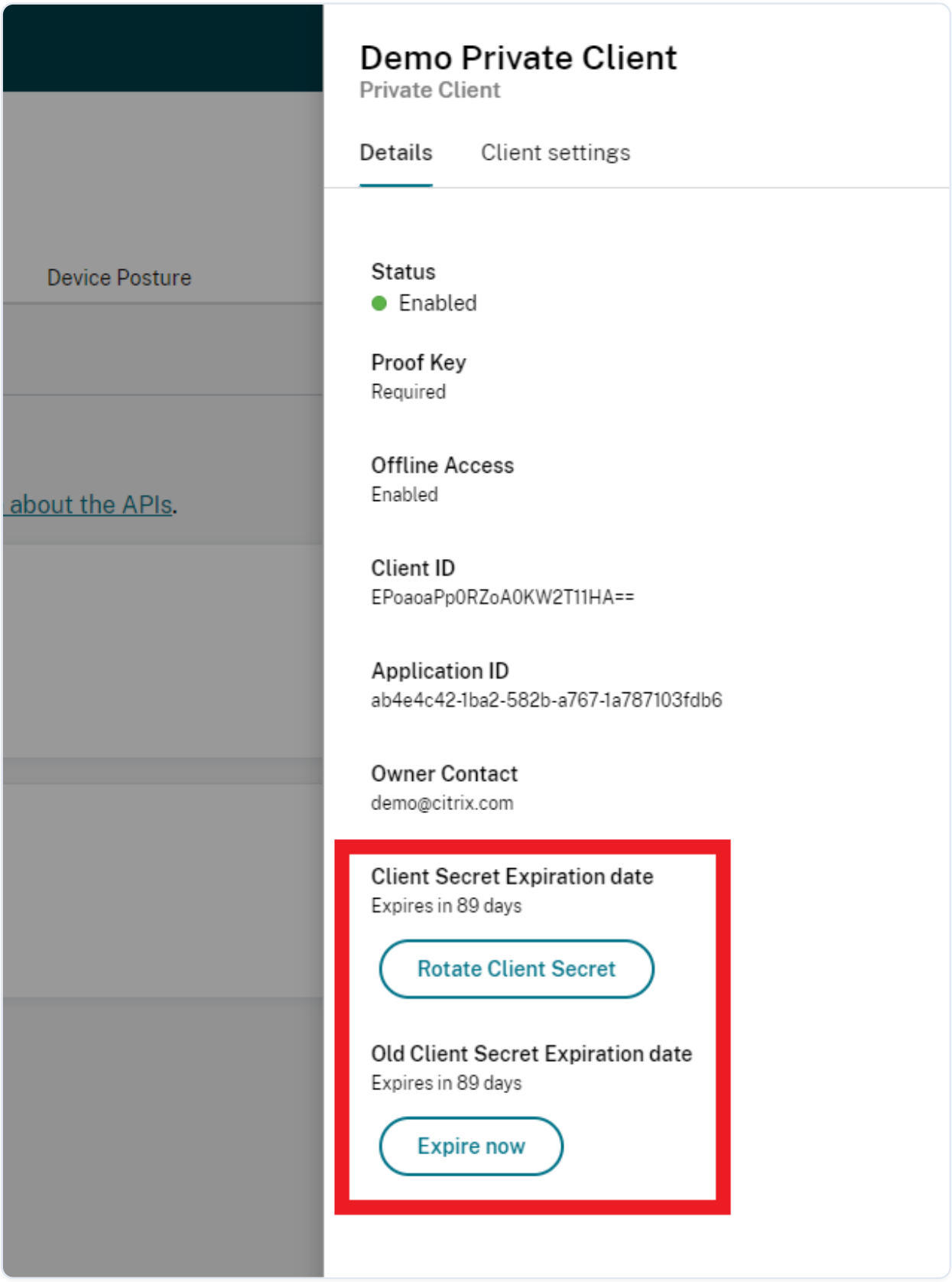


5. A modal shows you the new Client secret including the Client ID, giving you the ability to download and copy the information:

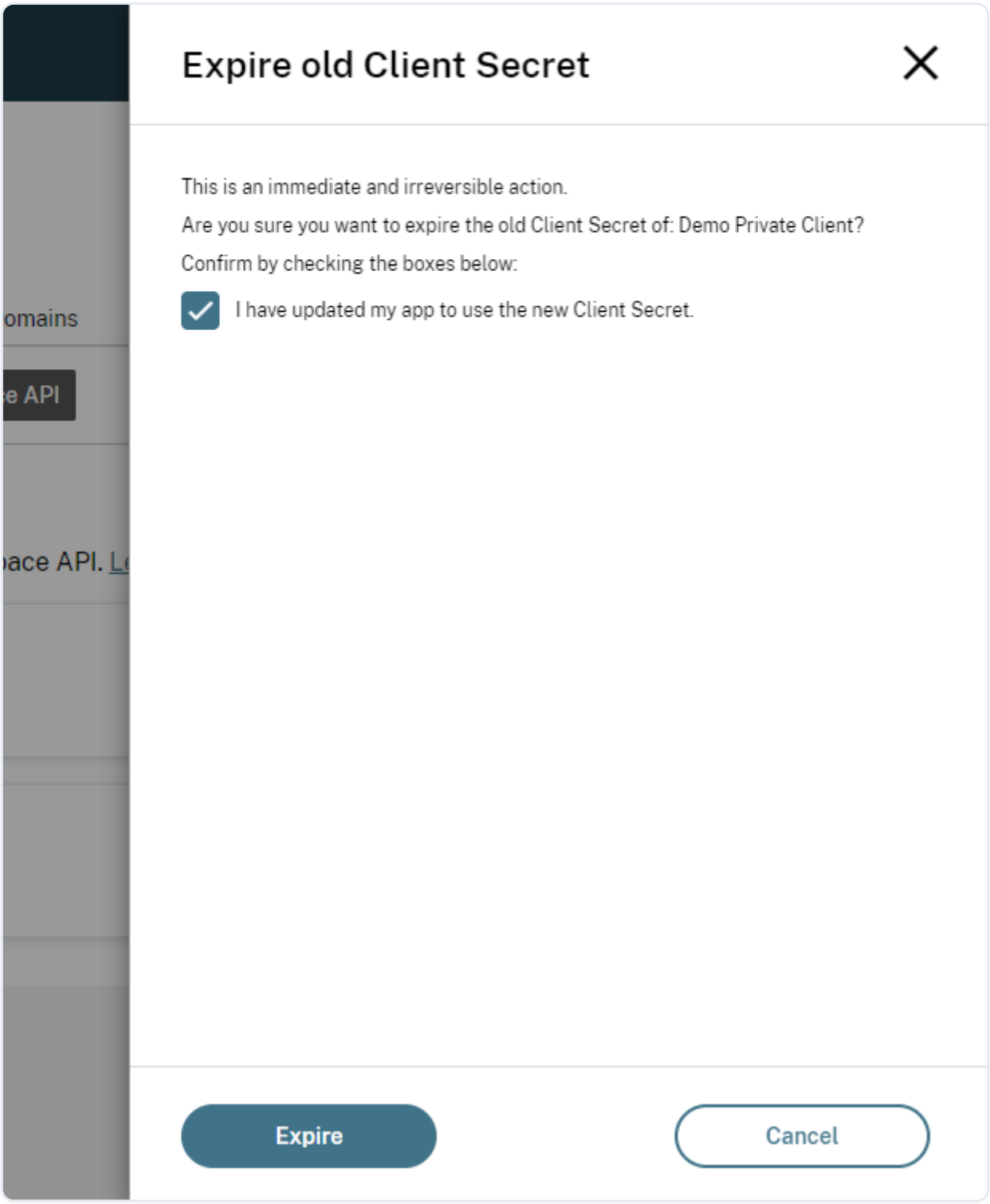


Once the client has been rotated, you can start using the new Client secret in your application, and the old secret continues to work until it expires, or until you trigger an expiry:

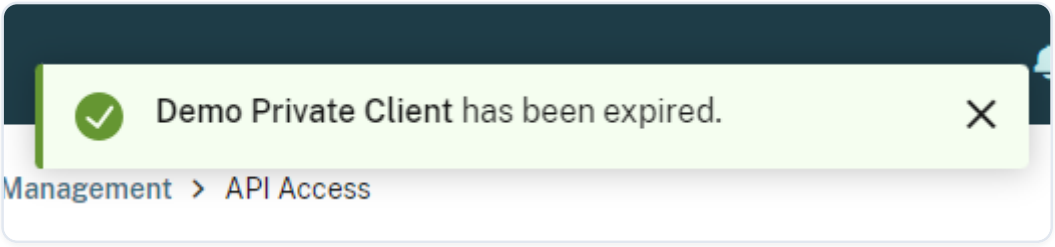
1. Navigate to the client details page by clicking 'Edit this client' and you'll be able to see the new secret Expiration date, alongside the old secret Expiration date:



2. Click on the 'Expire Now' secret to bring up the confirmation page:



3. A confirmation is displayed showing the old Client secret has been expired and no longer works:



Programmatic Rotation

It is possible to perform the client rotation and secret expiry using the Citrix Cloud API, this might be useful if you want to automate the client rotation.

Before you perform the rotation, consider the following:

- The existing Client Secret expires automatically after remaining valid days.
- You need to manually update your app to start using the newly generated Client Secret.

To perform the rotation:

1. Follow the Citrix Cloud API guide to create a service principal and obtain a CC Bearer Token.
2. Perform a web request to find all your customer clients to find the client `name` which is used to perform the rotation:

   Request:

   ```
   curl https://cws.citrixworkspacesapi.net/{CUSTOMER_ID}/rpclients --
   header "Authorization: CWSAuth bearer=$token"
   ```

   Note: The CUSTOMER_ID can be found on the Citrix Cloud service principals page (see step 1).

   Response:

   ```
   [
       {
           "name": "customer-private-305",
           "display_name": "Demo Client",
           "api_key": "api_key",
           "secret_update_time": 1686218219,
           "secret_expiration": 1693958400,
           "secret_update_time_old": null,
   ```

```
        "secret_expiration_old": null,
        "owner": "customer@customer.com",
        ...
    },
    ...
]
```

3. Using the above response, you can determine if you want to rotate the client based upon the `secret_expiration` value, which is a Unix timestamp.
4. Perform a web request to rotate the client. In the response, you'll see the `secret` time values update, and a new `secret` value which is valid for another 90 days.

   Request:

```
curl -X POST https://cws.citrixworkspacesapi.net/{CUSTOMER_ID}/
rpclients{CLIENT_NAME}/$rotateSecret --header "Authorization: CWSAuth
bearer=$token"
```

   Response:

```
{
    "client": {
        "name": "customer-private-305",
        "display_name": "Demo Client",
        "api_key": "api_key",
        "secret_update_time": 1691151918,
        "secret_expiration": 1698927918,
        "secret_update_time_old": 1691151918,
```

```
        "secret_expiration_old": 1693958400,
        "owner": "customer@customer.com",
        ...
    },
    "secret": "{NEW_SECRET}"
}
```

Once the client has been rotated, you can start using the new Client secret in your application, and the old secret continues to work until it expires, or until you trigger an expiry.

To perform an expiry of the old secret:

1. Ensure you have updated your app to use the new Client Secret, as it is not possible to authenticate using the expired client secret.
2. Perform a web request to expire the old secret:

   Request:

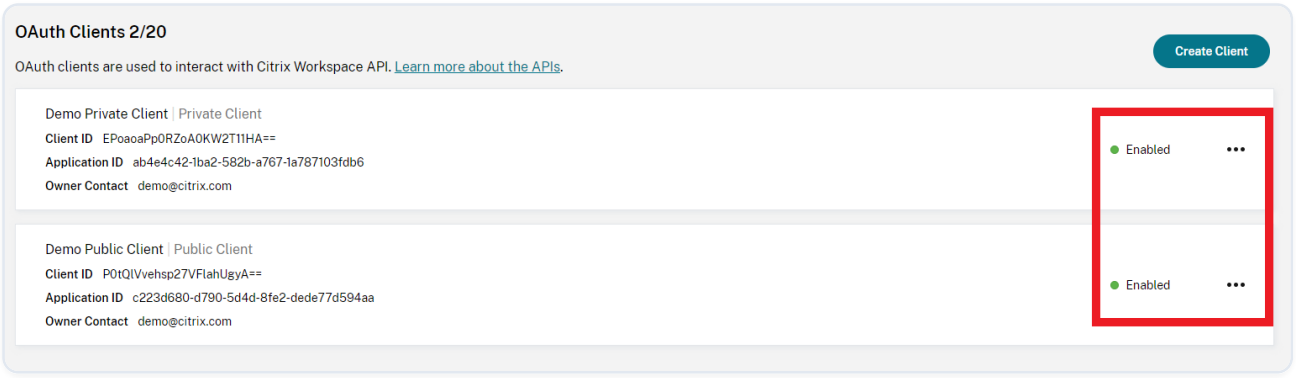
   ```
   curl -X POST https://cws.citrixworkspacesapi.net/{CUSTOMER_ID}/
   rpclients/{CLIENT_NAME}/$expireOldSecret --header "Authorization:
   CWSAuth bearer=$token"
   ```

   The response is a `200 OK` with an empty body, indicating the secret has been expired.

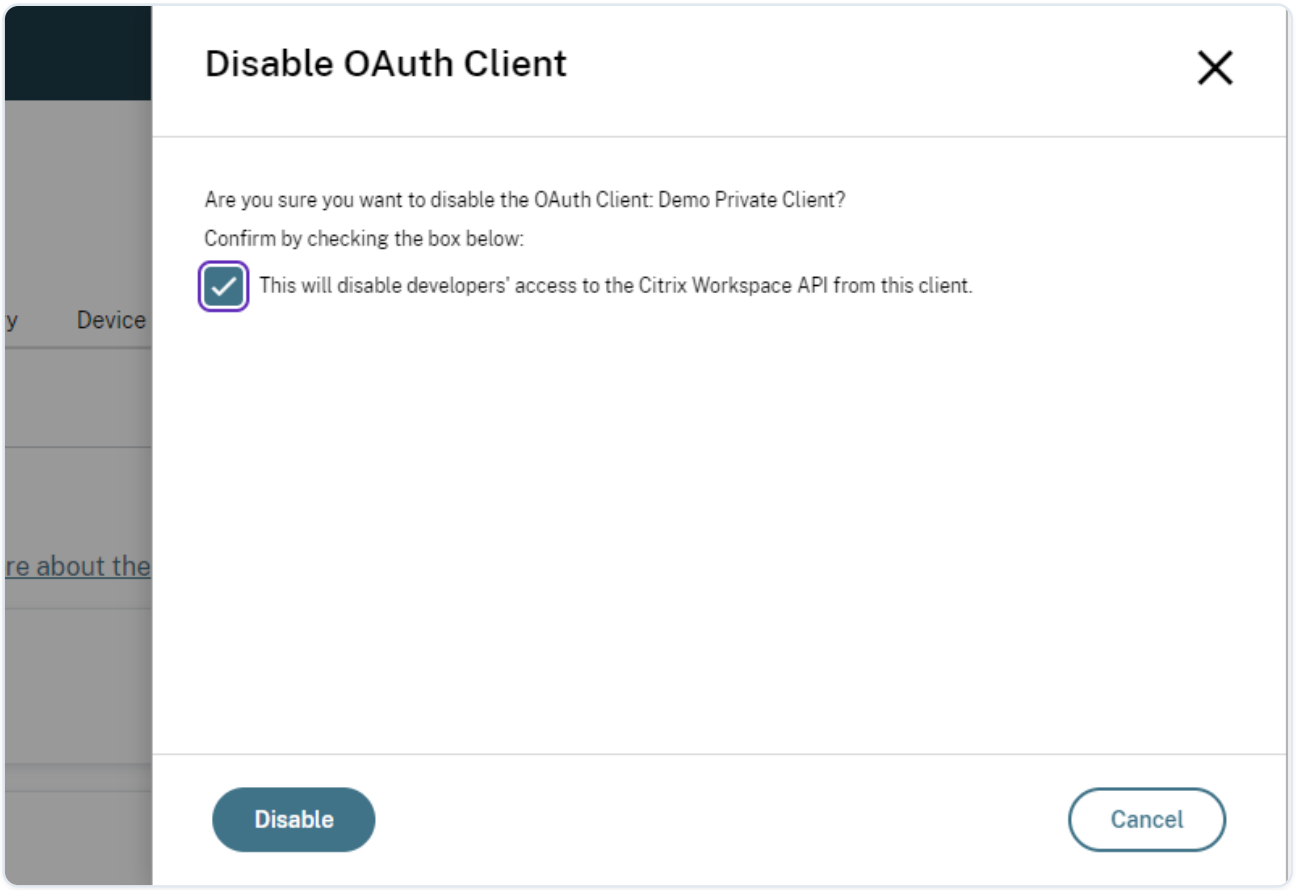
Enable and Disable Clients

Newly created clients are enabled by default. However, it is possible to disable them in the admin portal. This causes any application using them to stop working until the client is re-enabled.

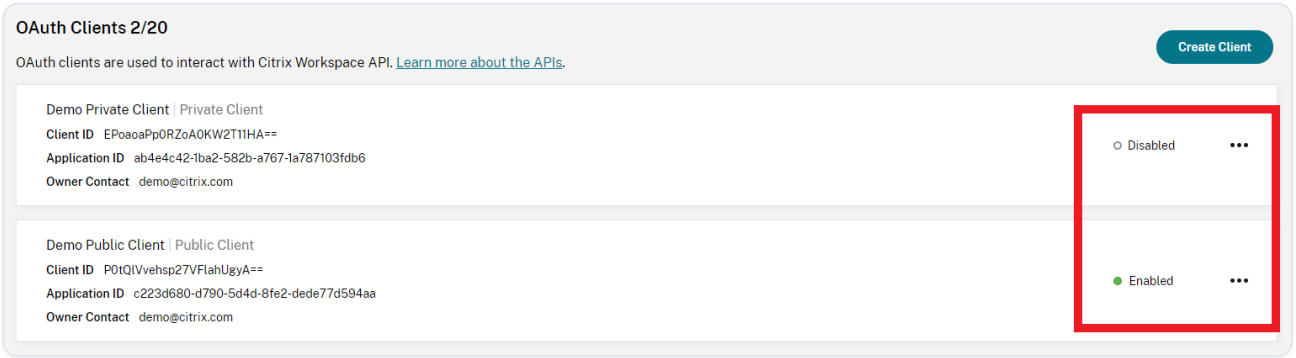
The status of a client can be seen on the client page:



Clicking on the three dots, you can select 'Disable this client' and tick the confirmation box:



Once disabled, the client shows that on the client page:



It can then be re-enabled following a similar process.

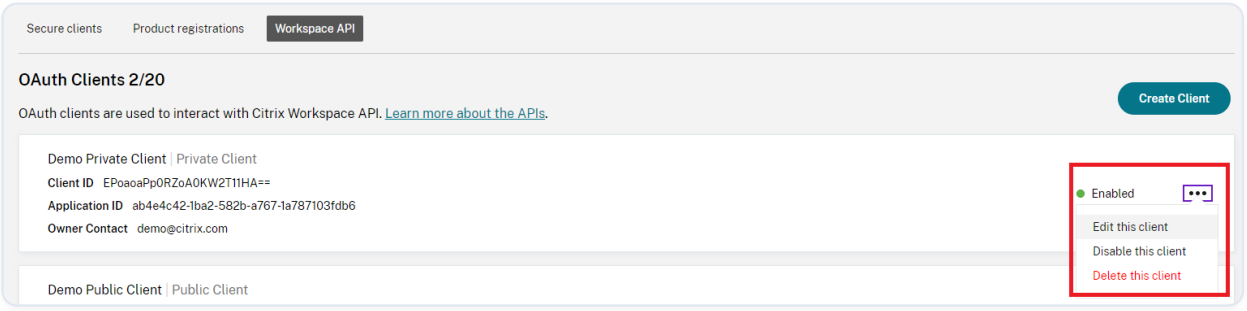
If the client is disabled and an application attempts to use it for the authorization code flows, it gets a `400 Bad Request` back from the Citrix Authorization Server as detailed here.

Editing a Client

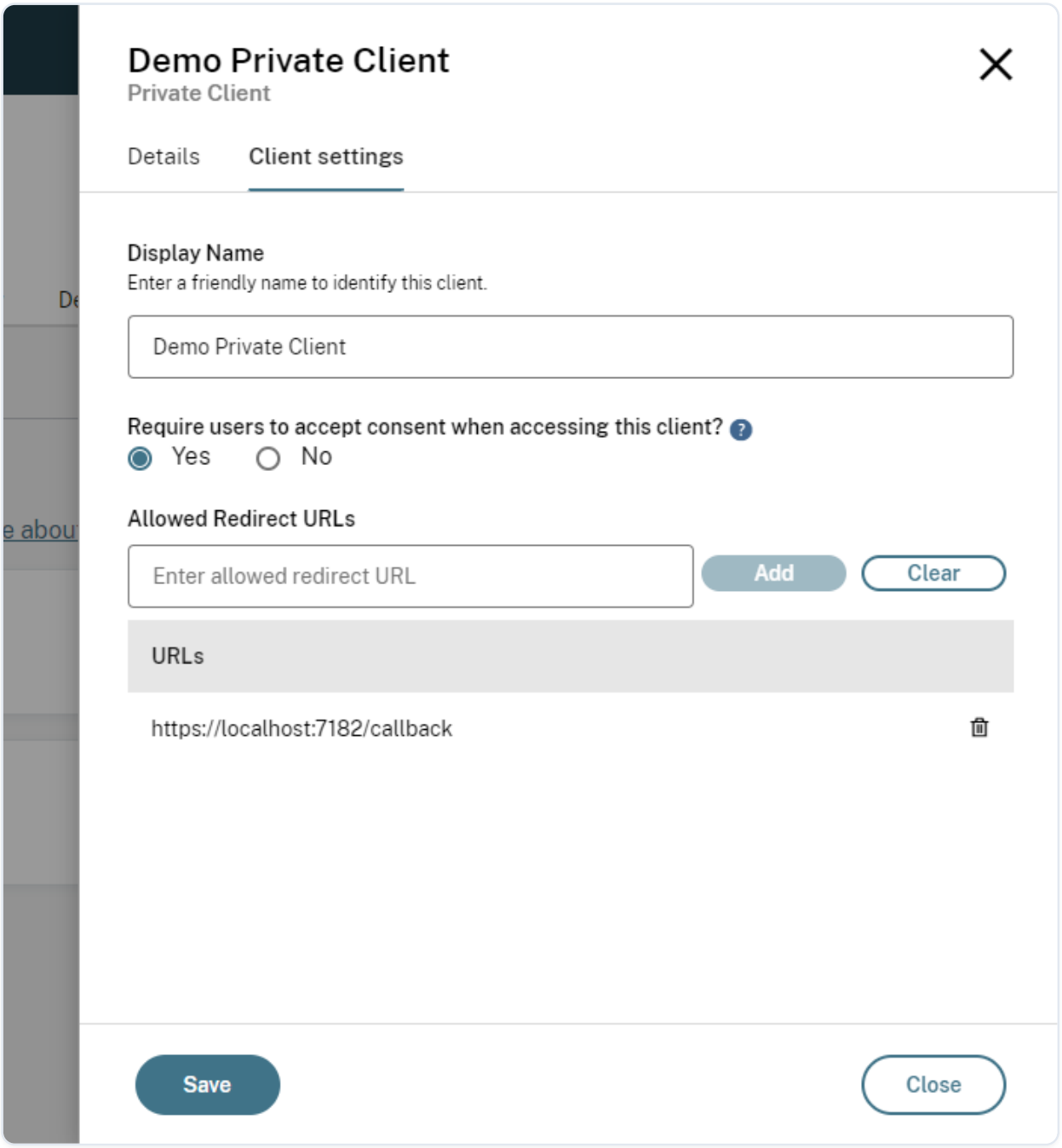
You can edit the following information for a client:

- Display Name
- Requiring User Consent
- Redirect URIs

1. Login to Citrix Cloud Admin Portal and navigate to **Identity and Access Management** > **API Access** > **StoreFront Cloud API**.
2. Find the client you want to edit in the client list, click on the three dots, and select 'Edit this client':



3. Click on the Client settings tab:



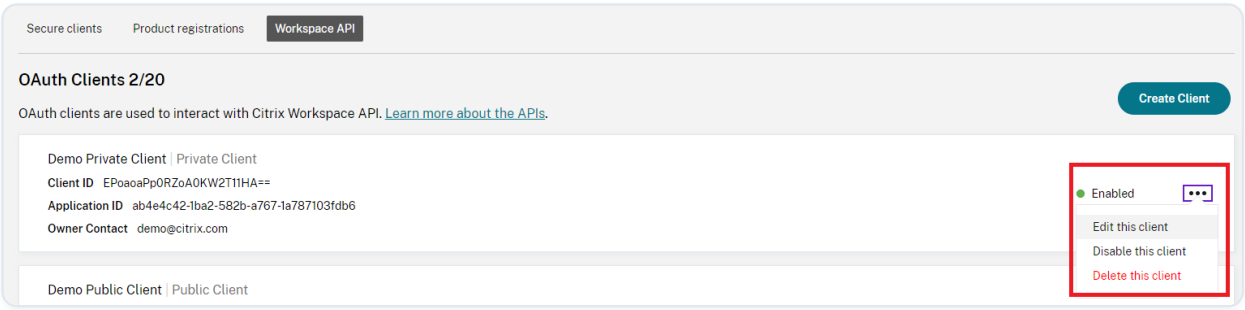
4. Make any changes and click 'Save'.

Deleting a Client

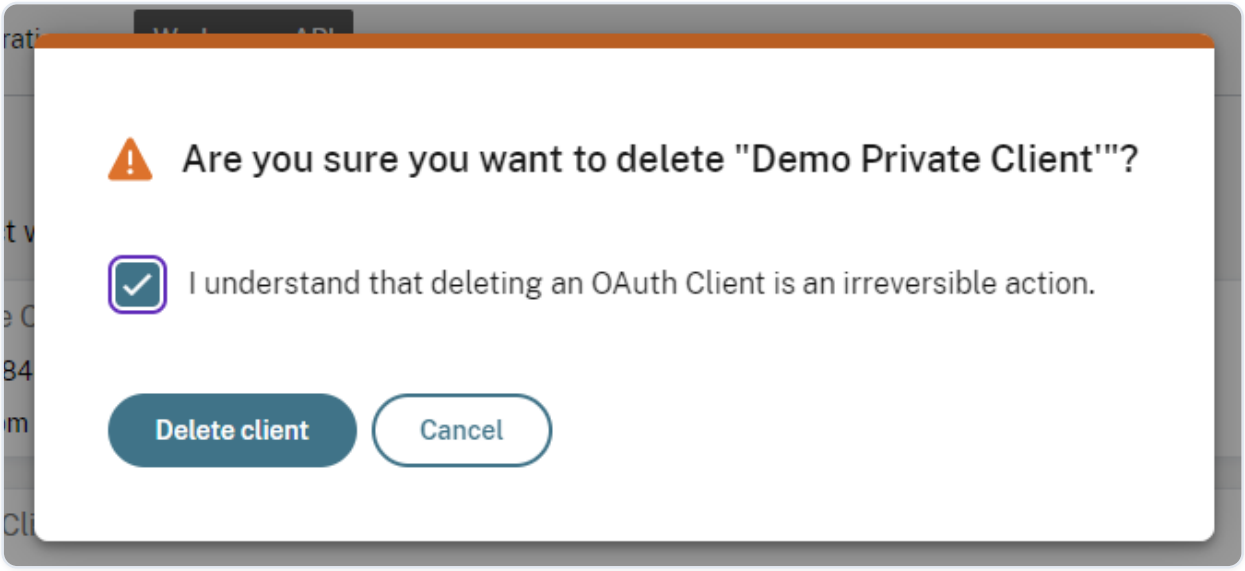
Deleting a client means that any application using the client stops working.

To delete a client via the dropdown menu:

1. Login to Citrix Cloud Admin Portal and navigate to **Identity and Access Management** > **API Access** > **StoreFront Cloud API**.
2. Find the client you want to delete in the client list, click on the three dots, and select 'Delete this client':

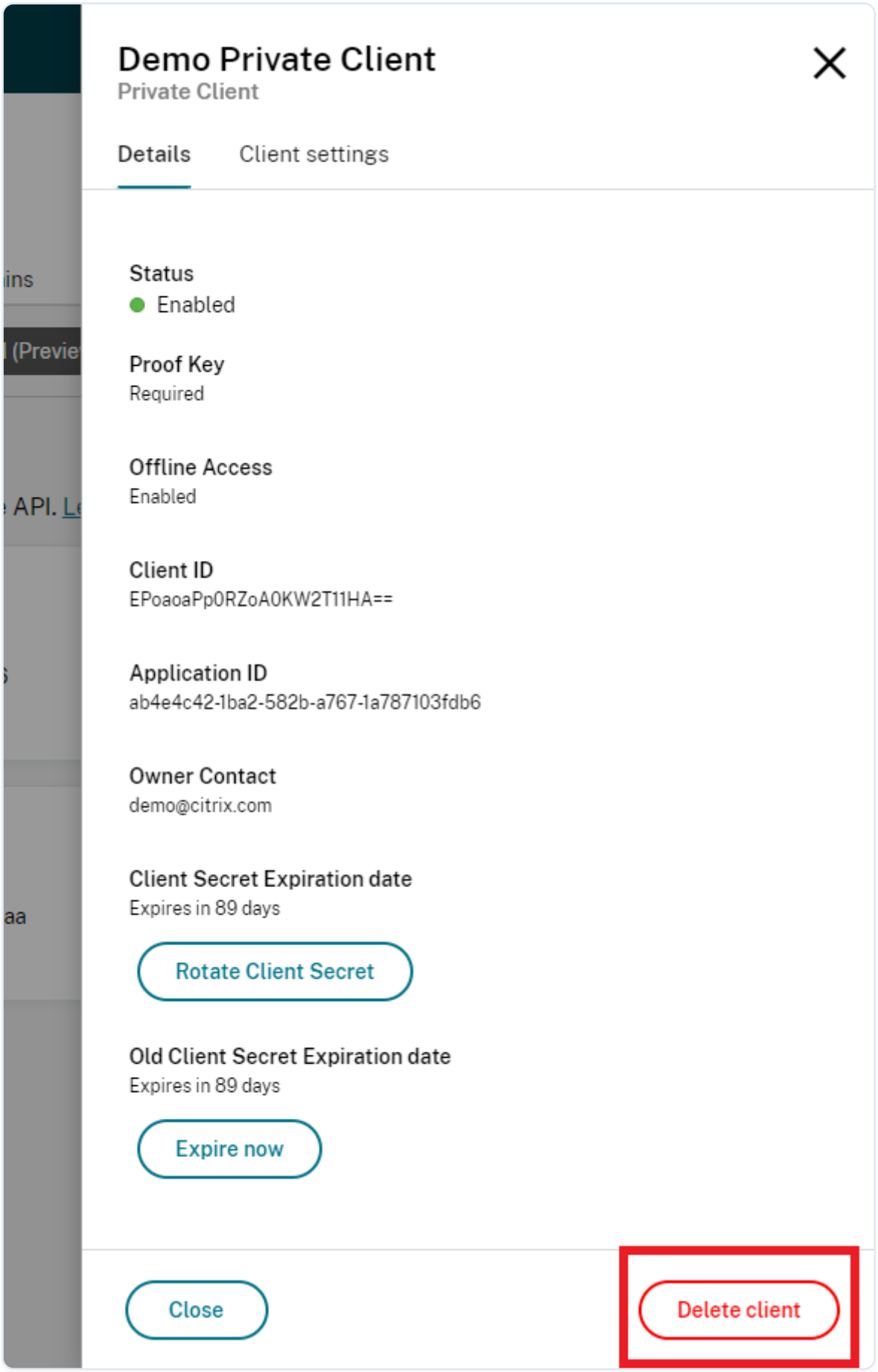


3. A confirmation appears. Select the confirmation and click 'Delete Client':



4. The client can now be deleted and afterwards, disappears from the client list.

You can also delete the client via the client details page:


Demo Private Client
Private Client
Details
Client settings
Status
Enabled
Proof Key
Required
Offline Access
Enabled
Client ID
EPoaoaPp0RZoA0KW2T11HA==
Application ID
ab4e4c42-1ba2-582b-a767-1a787103fdb6
Owner Contact
demo@citrix.com
Client Secret Expiration date
Expires in 89 days
Rotate Client Secret
Old Client Secret Expiration date
Expires in 89 days
Expire now
Close
Delete client

Store Discovery API Documentation

Discovery

The Discovery API helps you determine the service URLs that are available for the client, as well as the customer metadata that is necessary for the client operation.

Open API spec is available via your customer URL `<https://{CUSTOMER_DOMAIN}/citrixapi/swagger/index.html?urls.primaryName=discovery>`.

> **Notes:**
>
> - Replace '{CUSTOMER_DOMAIN}' with either your `cloud.com` domain or configured custom domain.
> - An application ID is required to make requests to the API. In order to obtain one you have to create an OAuth client.
> - Requests **must** contain either the `Citrix-ApplicationId` header, or the `ApplicationId` query parameter with the value set to the above mentioned application ID. The Open API spec guides you on which is required for what endpoints.
> - It may take up to 5 minutes after creating an OAuth client before the application ID is accepted by the API.
> - Does not require an authorization token.
> - Response may contain one or many service entries in the service list.
> - A service represents a grouped set of APIs with operations that relate to a similar domain.
> - A service entry can contain one or many API endpoints with an optional capability list.
> - The client can reference endpoints by "id" and services by "service" (name) as they are constant.

Store

The Open API spec is available via your customer URL `<https://{CUSTOMER_DOMAIN}/citrixapi/swagger/index.html?urls.primaryName=store>`.

Notes:

- Replace '{CUSTOMER_DOMAIN}' with either your `cloud.com` domain or configured custom domain.
- An application ID is required to make requests to the API. In order to obtain one you have to create an OAuth client.
- Requests **must** contain either the `Citrix-ApplicationId` header, or the `ApplicationId` query parameter with the value set to the above mentioned application ID. The Open API spec guides you on which is required for what endpoints.

Authentication

The Authentication API contains endpoints that are protected by OAuth. You need to provide an Authorization header containing the OAuth Access token, e.g. `"Authorization" = "Bearer <token>"`. If a valid Authorization header isn't provided, the request fails with a `400 Bad Request` response, with the reason stated as **invalid_grant**.

For example:

```
{

    "error": "invalid_grant",

    "error_description": "Access token is invalid."

}
```

To obtain an Access Token, follow the Authentication walkthrough.

Client Name and Device ID

Some API endpoints take parameters `clientName` and/or `deviceId` to identify the client. Where `deviceId` is omitted, the `clientName` is used as the `deviceId`.

- **Client Name**: Used to identify the client in broker policies. This should be the computer hostname. If the hostname is not available then you should use an alternative identifier for the device.
- **Device ID**: This is used to identify which device is connected to a session. You should use the same Device ID consistently for a specific device. Typically the computer hostname is used, but you can use an alternative value as long as it uniquely and consistently identifies a device.

It is recommended to set the same value for `clientName` and `deviceId` but it is not a requirement.

Supported Launch Flows

The following sequence diagrams give a more detailed view on how you can use the APIs to perform ICA launches.

The API currently only supports launching resources that have the `ica30` `clientType` that can be found during resource enumeration.

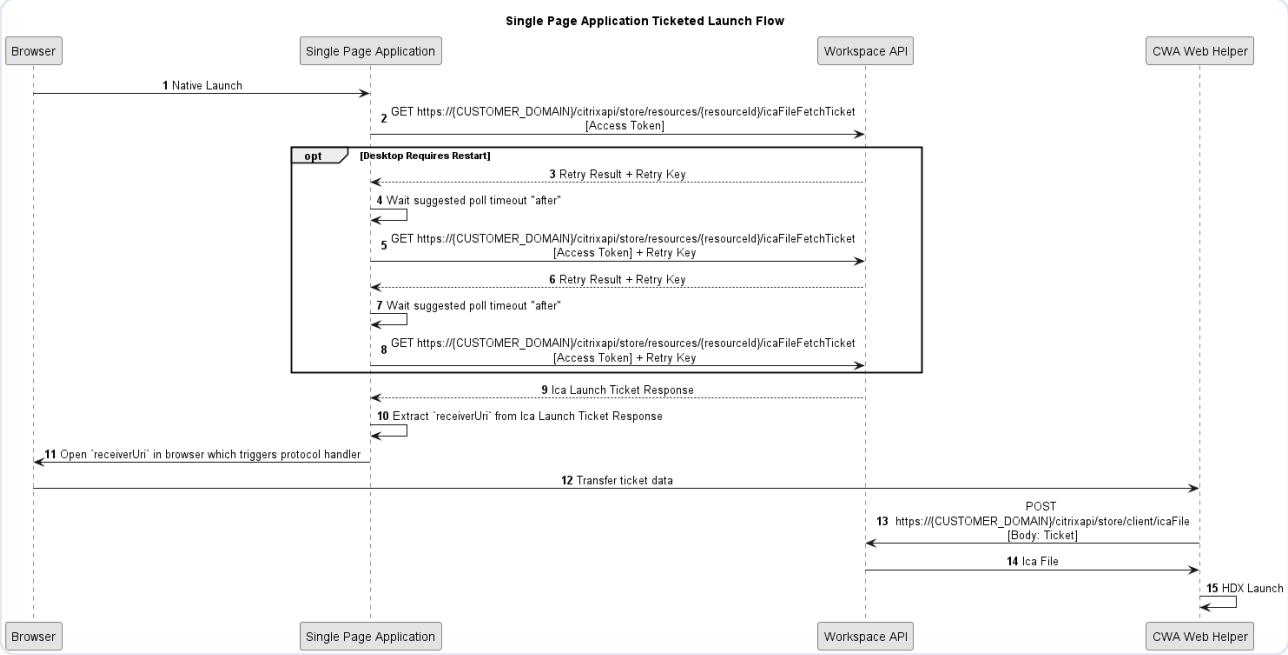
Launch using ICA file fetch ticket and receiver URI

This is the recommended approach to launch a virtual app or desktop. Rather than getting an ICA file, the application gets a one time ICA file fetch ticket. It passes this ticket to Citrix Workspace app which uses that ticket to retrieve the ICA file. This avoids needing to save the ICA file on disk which is a security risk. The following steps are how this launch flow is executed:

1. Perform an enumeration or session list. For each resource, the response includes a property `icaFileFetchTicketUrl` which contains the ICA file fetch ticket URL.
2. Call the endpoint discovered in the above step. If it returns retry required (`202`) then the resource is not yet ready. The response body contains information about how to proceed. Wait for the period defined in `pollTimeout` then call the endpoint defined in `retryUrl`. You may need to repeat this multiple times.
3. Once the endpoint returns launch ticket created (`201`) the resource is ready to be launched. The endpoint body contains field. `receiverUri` of the form `receiver://path/data` which encodes the information required to get the ica file, including the ticket.
4. Invoke the receiver URI retrieve in the above step. Citrix Workspace app intercepts this URL, reads the information out of the URL, downloads the ICA file and connects to the resource.

Refer to the Single Page Application sample to see how this has been implemented.

The flow can be seen in this sequence diagram:



Notes:

- `Citrix-ApplicationId` has been omitted from the sequence diagram, see Open API spec for the endpoint requirements.
- The sequence diagram illustrates the flow for the `resources` launch, which is the same for a `sessions` launch.

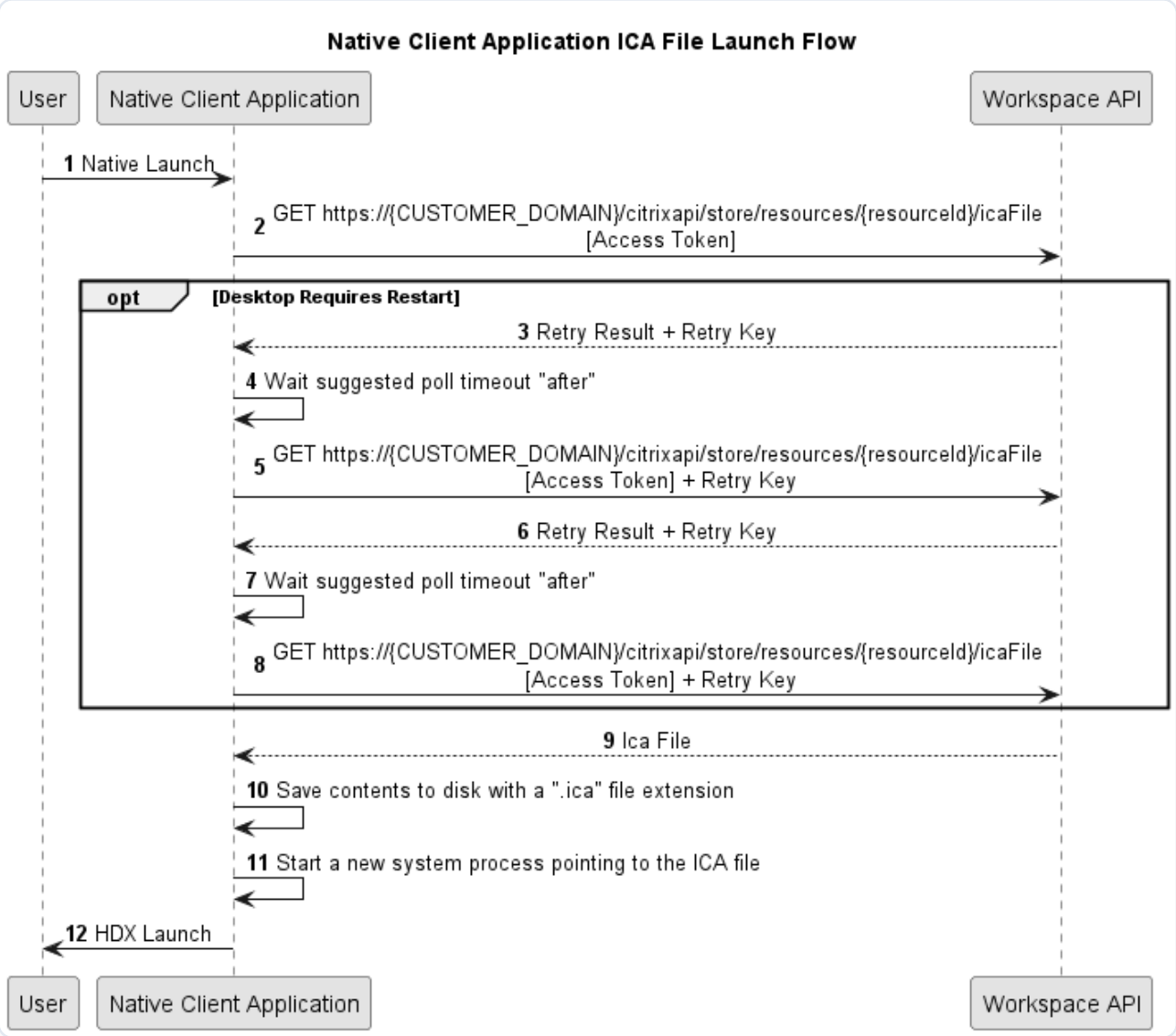
ICA File Download Launch

Use this flow if you need to retrieve an ICA file. In most case it is recommended that you use the ticketed launch as this allows Citrix Workspace app to directly retrieve the ICA file in memory, avoiding the risks of storing the ICA file on disk. The following steps are how this launch flow is executed:

1. Perform an enumeration or session list. For each resource, the response includes a property `icaFileUrl` which contains the URL to obtain the ICA file.
2. Call the endpoint discovered in the above step. If it returns retry required (`202`) then the resource is not yet ready. The response body contains information about how to proceed. Wait for the period defined in `pollTimeout` then call the endpoint defined in `retryUrl`. You may need to repeat this multiple times.

3. Once the endpoint returns the ICA file response (`200`) the resource is ready to launch. Save the contents of the ICA file to disk with a `.ica` file extension and start a new process pointing to that file. This triggers a HDX launch and connects to the resource.

Refer to the Native Client sample to see how this launch flow has been implemented.



Notes:

- `Citrix-ApplicationId` has been omitted from the sequence diagram, see Open API spec for the endpoint requirements.
- The sequence diagram illustrates the flow for the `resources` launch, which is the same for a `sessions` launch.

Open Source Samples

The following sample projects demonstrate how to interact with the APIs to authenticate, list available resources, and generate an ICA file that you can then pass to Citrix Workspace app to handle the connection.

Sample web application

Standalone web-based client that allows the user to login via the Citrix Authorization server (and your configured Identity Provider), obtain an OAuth Access and Refresh Token, and call the APIs.

Code can be found here.

Sample single-page web application

SPA that shows how to interact with the APIs to build a functional web portal. It is a pure Javascript implementation that is hosted by a C# ASP .NET server.

Includes a sample Token Management Service backend for handling tokens.

Code can be found here.

Sample native application

Windows Native Client that shows how to interact with the APIs. It is written using the C#'s WPF framework, with a simple UI to facilitate the actions required to login and begin calling APIs.

Includes a sample Token Management Service backend for handling tokens.

Code can be found here.

Security Best Practices

Securely storing Client credentials

The OAuth clients (that you would have created by following the client management guide), especially the client secret in the case of private clients, must be handled like any other secret and stored securely. The method of storing the client credentials might differ depending on your company, however it is strongly advised to use a secret storage management solution that has restricted access.

Do not hardcode the credentials.

If the client credentials are exposed or compromised, we advise to take one of the following actions:

- Perform a client secret rotation and expire the old secret
- Disable the client
- Delete the client and create a new one

For more information on these actions, view the client management guide.

Authorization Flows

To achieve the highest level of security, it is recommended to use the authorization code flow with PKCE as outlined in the authentication guide. This flow is enforced for public clients, but administrators creating private clients can choose whether they wish to use the PKCE flow or the standard authorization code flow without PKCE.

For additional security during the authorization flows, it is mandatory to use the `state` parameter properly to prevent CSRF (cross-site request forgery) attacks.

User Tokens

User tokens are issued by the Authorization Server after performing an OAuth Authorization Flow.

Access tokens are the primary tokens that are consumed by your application; they are considered short-lived and relatively safe to store locally on-device in a secure way. The lifetime of the access token is provided in the Authorization Server response.

A refresh token is provided if you have met the requirements (outlined in the client management guide). This is considered a long-lived token, and should not be stored on the client side. Instead, a server-side application can be used to perform the authentication flows and provide the client with an access token, securely storing the refresh token on the server. When the client requires a new access token, the server can then request a new one from the Authorization Server securely. This is demonstrated in our Single Page Application and Native Client open source samples via the Token Management Service.

Access and refresh tokens cannot be revoked. Each time a token refresh flow is performed, the current refresh token is invalidated and cannot be used again. A new refresh token is then returned alongside the access token, as part of a pattern known as 'Refresh Token Rotation'. As the token is constantly changing with each refresh cycle and the previous is no longer valid, the threat is reduced.