1. [Page 9](javascript: void(0);)
2. OAuth Grants

# OAuth Grants

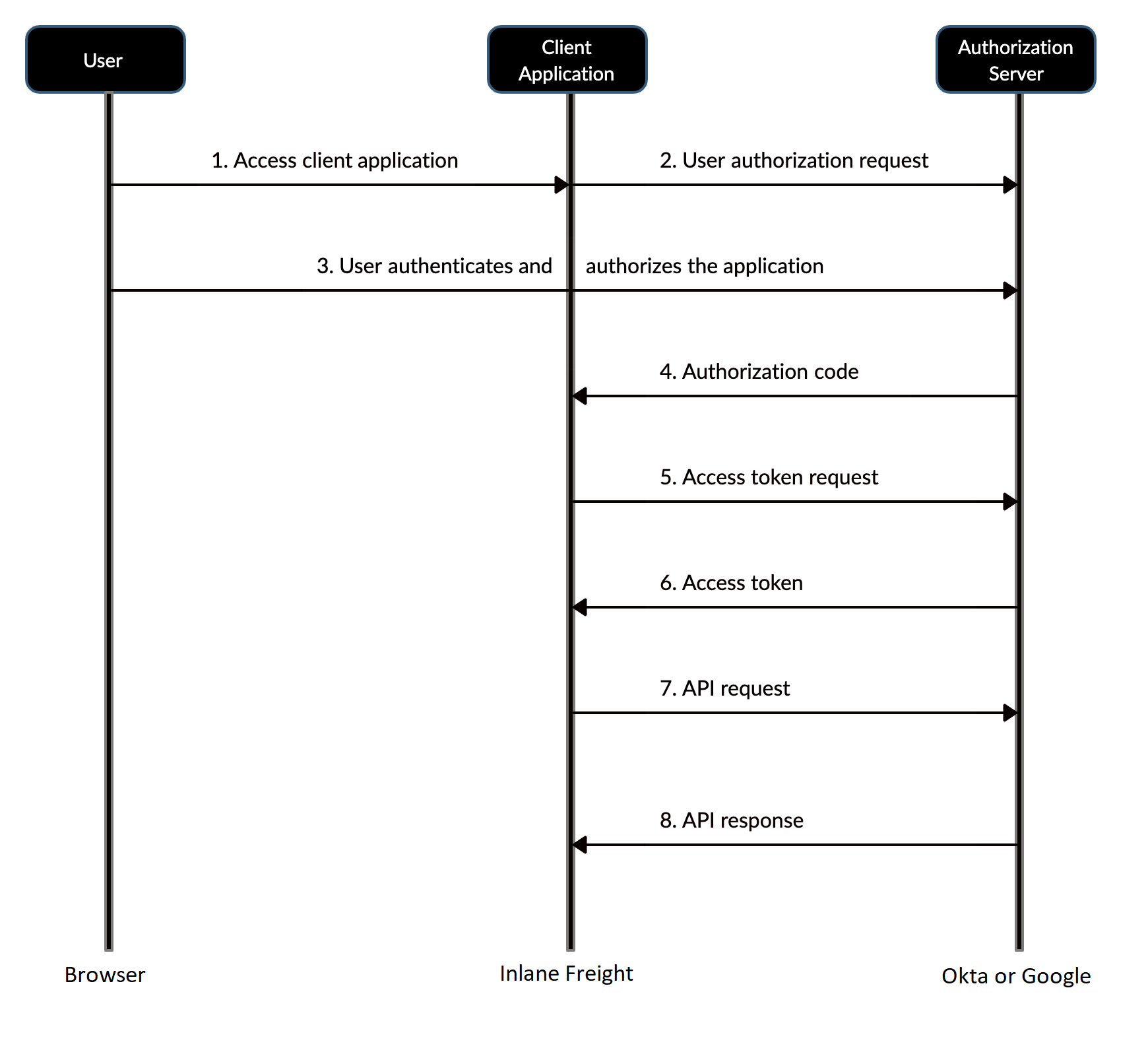
OAuth grants are categorized into five types applicable to different use cases.

* Authorization Code Grant
* Implicit Grant
* Resource Owner Credentials Grant
* Client Credentials Grant
* Refresh Token Grant

Let's examine each of these grant types based on the GitHub example we saw in the previous section.

## Authorization Code Grant

An Authorization Code Grant is generally used by applications that don't want to store confidential data, such as a user's credentials, images, and social security numbers. The flow of an Authorization Code Grant is shown below.

Let's examine what is happening here.

1. When a user clicked on the login via GitHub icon, the following request was sent to the client application.

Code: html

<http://github.com/login/oauth/authorize?response_type=code&client_id=CLIENT_ID&redirect_uri=http://inlanefreight.com/index.php?callback&scope=user&state=a45c12e87d4522>

* + **response\_type**: Indicates that the client application expects to receive an authorization code.
  + **client\_id**: The client ID that is received from the authorization server when the application is first created.
  + **redirect\_uri**: Indicates the URI to return the user to after authorization is complete.
  + **scope**: One or more scope values indicating which parts of the user's account the client application wishes to access.
  + **state**: A random string generated by the client application that is used to validate whether the request was triggered by the current user.

1. The authorization server now prompts the user to authorize the request. Once the user does this, the authorization server will redirect the user to the following URL.

Code: html

<http://inlanefreight.com/index.php?code=AUTH_CODE&state=a45c12e87d4522>

* + **code**: The authorization code returned by the authorization server.
  + **state**: The server returns the same state value that the client application sent in the previous request.

1. The client application can now exchange this authorization code for an access token.

Code: html

<https://github.com/login/oauth/access_token?grant_type=authorization_code&code=AUTH_CODE&redirect_uri=http://inlanfreight.com/index.php?callback&client_id=CLIENT_ID&client_secret=CLIENT_SECRET>

* + **grant\_type**: The grant type for this flow is authorization\_code.
  + **code**: This is the code that the client application received in the query string.
  + **redirect\_uri**: This must be identical to the redirect URI provided in the original link.
  + **client\_id**: The client ID received from the authorization server when the application is first created.
  + **client\_secret**: As this request is generated server-side, the secret is included.

1. The authorization server replies with an access token and an expiration time.

Code: json

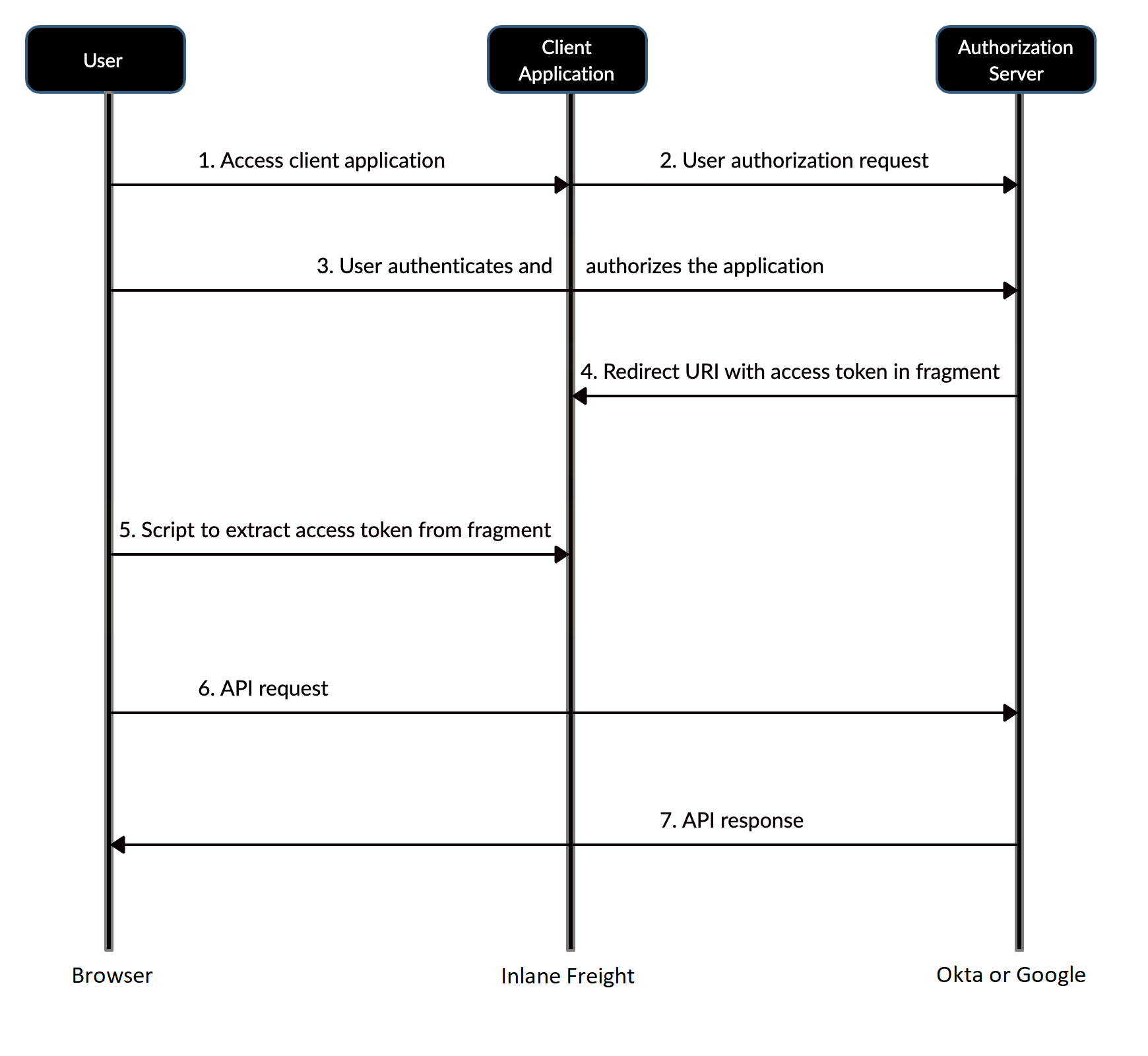
{  
 "access\_token":"RsT5OjbzRn430zqMLgV3Ia",  
 "expires\_in":3600  
}

1. Using the access token, the client application can obtain the user's profile information from the authorization server.

## Implicit Grant

The Implicit Grant type is used mainly in small single-page applications, where the application cannot secure the client secret. For example, this may be because the application code and storage are easily accessible.

An Implicit Grant is very similar to an Authorization Code Grant, with a key difference being that the authorization server returns the access token directly instead of an authorization code. Let's examine the flow of the Implicit Grant process.



1. When a user opts to log in via GitHub, the following request is sent to the client application.

Code: html

<https://github.com/login/oauth/authorize?response_type=token&client_id=CLIENT_ID&redirect_uri=http://inlanefreight.com/index.php?callback&scope=user&state=1234zyx&code_challenge=CODE_CHALLENGE&code_challenge_method=S256>

* + **response\_type**: Indicates that the client application expects to receive an access token.
  + **client\_id**: The client ID which is received from the authorization server when the first application is created.
  + **redirect\_uri**: Indicates the URI to return the user to after authorization is complete.
  + **scope**: One or more scope values indicating which parts of the user's account the client application wishes to access.
  + **state**: A random string generated by the client application, which is used to validate whether the request was triggered by the current user.
  + **code\_challenge**: The URL-safe base64-encoded SHA256 hash of the client secret.
  + **code\_challenge\_method**: Indicates the type of hashing method used by the application.

1. The authorization server prompts the user to authorize the request. Once the user has done this, the authorization server will redirect the user to the following URL.

Code: html

<http://inlanefreight.com/index.php#access_token=ACCESS_TOKEN&token_type=Bearer&expires_in=600&state=1234zyx>

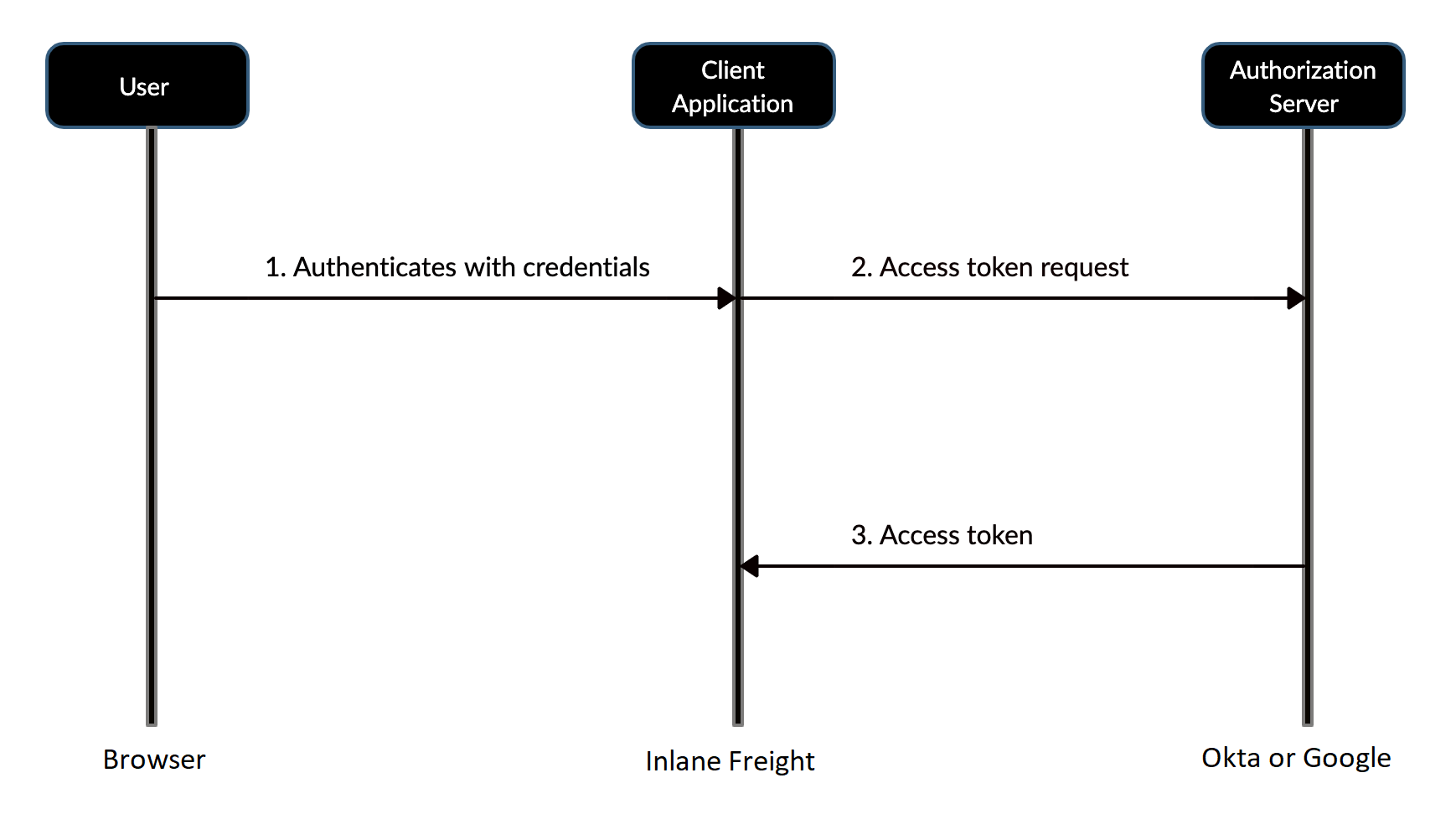
* + **access\_token**: Access token returned by the authorization server.
  + **token\_type**: Type of the token returned by the server. In this case, if it's just a string server will respond with Bearer.
  + **expires\_in**: The duration of time the access token is granted for.
  + **state**: The server returns the same state value that the client application sent in the previous request.

1. Using the access token, the client application can obtain the user's information from the authorization server.

## Resource Owner Credentials Grant

The Resource Owner Credentials Grant type is suitable in cases where the resource owner has a trust relationship with the client. Care should be taken when enabling this grant type. It should only be used when other flows are not viable. In this type, the user needs to share their credentials directly with the client, which the client then submits to the authorization server in exchange for an access token.

This grant type is used to migrate existing clients using direct authentication schemes such as HTTP Basic or Digest authentication to OAuth by converting the stored credentials to an access token. The flow of the Resource Owner Credentials Grant is shown below.



1. When the user submits credentials through the form in a client application, the following request occurs.

Code: html

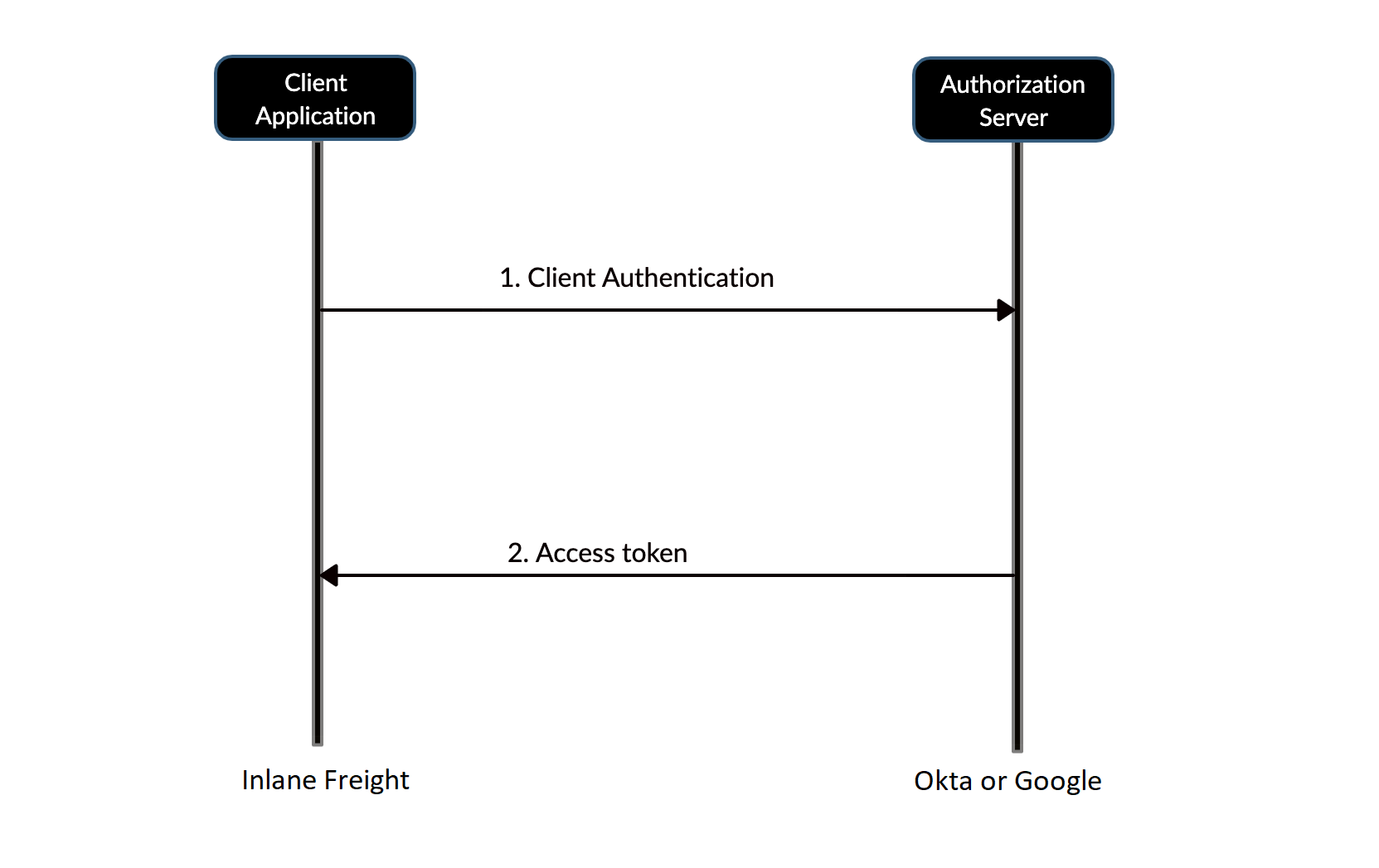
<https://github.com/login/oauth/authorize?grant_type=password&client_id=CLIENT_ID&username=USERNAME&password=PASSWORD>

* + **grant\_type**: The grant type for this flow is the user's password, which is shared with the client to receive the access token.
  + **username**: The user's username as collected by the application.
  + **password**: The user's password as collected by the application.
  + **client\_id**: The client ID received from the authorization server when the client first creates the application.

1. The authorization server replies with an access token in the same format as the other grant types.

## Client Credentials Grant

With the Client Credentials Grant type, a client application will authenticate to the authorization server and request an access token. This grant type is used in machine-to-machine authentication, where a specific user’s permissions to access the data is not required. The flow of Client Credentials Grant is shown below.



1. The client application will make a post request with the following URL.

Code: html

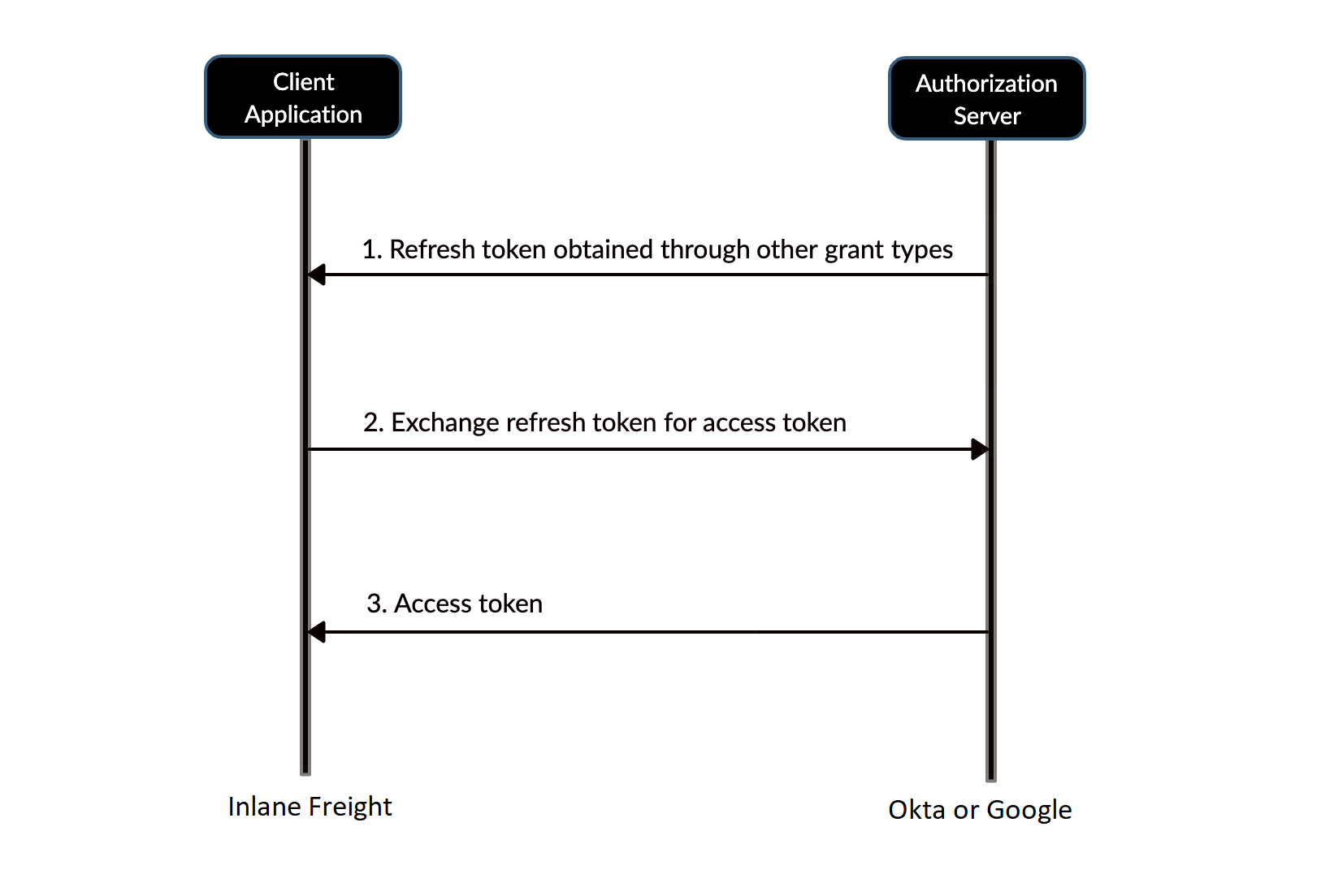
<https://github.com/login/oauth/access_token?grant_type=client_credentials&client_id=CLIENT_ID&client_secret=CLIENT_SECRET>

* + **grant\_type**: The grant type for this flow is client\_credentials.
  + **client\_id**: The client ID which is received from the authorization server when the application is first created.
  + **client\_secret**: As this request is made from server-side code, the secret is included.

1. The authorization server replies with an access token in the same format as other grant types.

## Refresh Token Grant

Clients use the Refresh Token Grant type to retrieve an access token. This allows clients to obtain a new access token without further interaction with the user. The flow of the Refresh Token Grant is as below.



1. The client application will use other grant types (authorization code grant or implicit code grant) to obtain the refresh token.
2. Client application sends a request to the authorization server to exchange the refresh token with an access token.

Code: html

<https://github.com/login/oauth/access_token?grant_type=refresh_token&refresh_token=REFRESH_TOKEN&client_id=CLIENT_ID&client_secret=CLIENT_SECRET>

* + **grant\_type**: The grant type for this flow is refresh\_token.
  + **refresh\_token**: Refresh token obtained from the authorization server using another grant type.
  + **client\_id**: The client ID which is received from the authorization server when the application is first created.
  + **client\_secret**: Since this request is made from server-side code, the secret is included.

Now that we understand how different grants of OAuth work, let's start working on exploiting OAuth.