OAuth 2.0
Developers Guide

Ping Identity, Inc. • 1001 17th Street, Suite 100, Denver, CO 80202 • 303.468.2900

Table of Contents

Contents

TABLE OF CONTENTS **2**

ABOUT THIS DOCUMENT 3

GETTING STARTED **4**

1 OVERVIEW **5**

1.1 OAUTH 2.0 OVERVIEW 5
1.2 DEVELOPER CONSIDERATIONS 5
1.2.1 Application Developer *5*
1.2.2 API Developer *6*

APPLICATION DEVELOPER CONSIDERATIONS **7**

2 GET A TOKEN **8**

2.1 OAUTH 2.0 GRANT TYPES 8
2.2 AUTHORIZATION CODE GRANT 9
2.2.1 Client Configuration *10*
2.2.2 Request authorization from user and retrieve authorization code *10*
2.2.3 Swap the authorization code for an access token *11*
2.3 IMPLICIT GRANT 13
2.3.1 Client Configuration *14*
2.3.2 Request authorization from user and retrieve access token *14*
2.4 RESOURCE OWNER PASSWORD CREDENTIALS (ROPC) 16
2.4.1 Client Configuration *16*
2.4.2 Request user authentication and retrieve access token *17*
2.5 CLIENT CREDENTIALS 19
2.5.1 Client Configuration *19*
2.5.2 Request access token *20*
2.6 EXTENSION GRANTS (I.E. SAML BEARER) 21
2.6.1 Client Configuration *21*
2.6.2 Request access token *22*

3 REFRESH A TOKEN **24**

4 USE A TOKEN **26**

API DEVELOPER CONSIDERATIONS **27**

5 VALIDATE A TOKEN **28**

5.1.1 Client Configuration *28*

6 REFERENCES **31**

About this Document

This document provides a developer overview of the OAuth 2.0 protocol. It provides an overview of the processes an application developer and an API developer need to consider to implement the OAuth 2.0 protocol.

Explanations and code examples are provided for "quick win" integration efforts. As such, they are incomplete and meant to complement existing documentation and specifications.

This document assumes familiarity with the OAuth 2.0 protocol and PingFederate. For more information about OAuth 2.0, refer to:

- PingFederate Administrator's Manual
- OAuth 2.0 RFC 6749

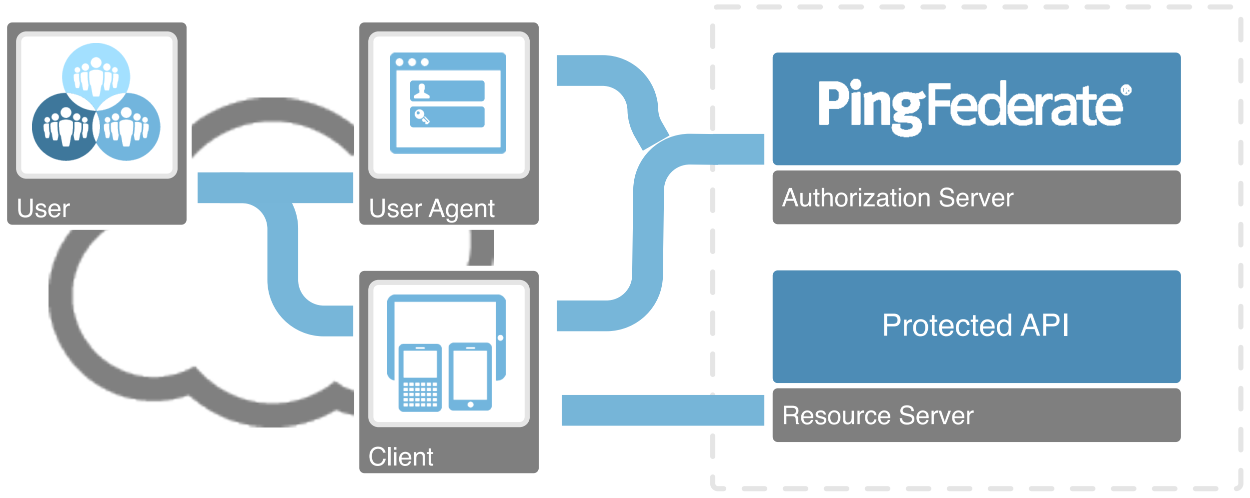
The samples described in this document use the OAuth2 Playground sample application available for download from the products page on pingidentity.com.

Note:	This document explains a number of manual processes to request and validate the OAuth tokens. While the interactions are simple, PingFederate is compatible with many 3rd party OAuth client libraries that may simplify development effort.

Getting Started

1 Overview

1.1 OAuth 2.0 Overview



The OAuth 2.0 protocol uses a number of actors to achieve the main tasks of getting, validating, and using an access token. These will be described, as well as optional steps of refreshing this access token.

The main actors involved are:

Actor	**Responsibility**
User or Resource Owner	The actual end user, responsible for authentication and to provide consent to share their resources with the requesting client.
User Agent	The user's browser. Used for redirect-based flows where the user must authenticate and optionally provide consent to share their resources.
Client	The client application that is requesting an access token on behalf of the end user.
Authorization Server (AS)	The PingFederate server that authenticates the user and/or client, issues access tokens and tracks the access tokens throughout their lifetime.
Resource Server (RS)	The target application or API that provides the requested resources. This actor will validate an access token to provide authorization for the action.

1.2 Developer Considerations

1.2.1 Application Developer

The application developer will be responsible for the user-facing elements of the process. They will need to authenticate the user and interface with the back-end APIs.

There are three main actions an application developer needs to handle to implement OAuth 2.0:

1. Get an access token
2. Use an access token
3. Refresh an access token (optional)

1.2.2 API Developer

The API developer builds the API that the application talks to. This developer is concerned with the protection of the API calls made and determining whether a user is authorized to make a specific API call.

The OAuth 2.0 process an API developer needs to handle is to:

1. Validate a token

Note:	In some cases the “API Developer” may be using a service bus or authorization gateway to manage access to APIs, and therefore the task of validating the access token would be shifted to this infrastructure.

Application Developer Considerations

2 Get a token

This section will explain how to get an OAuth2 access token (and optionally a refresh token) from the PingFederate infrastructure.

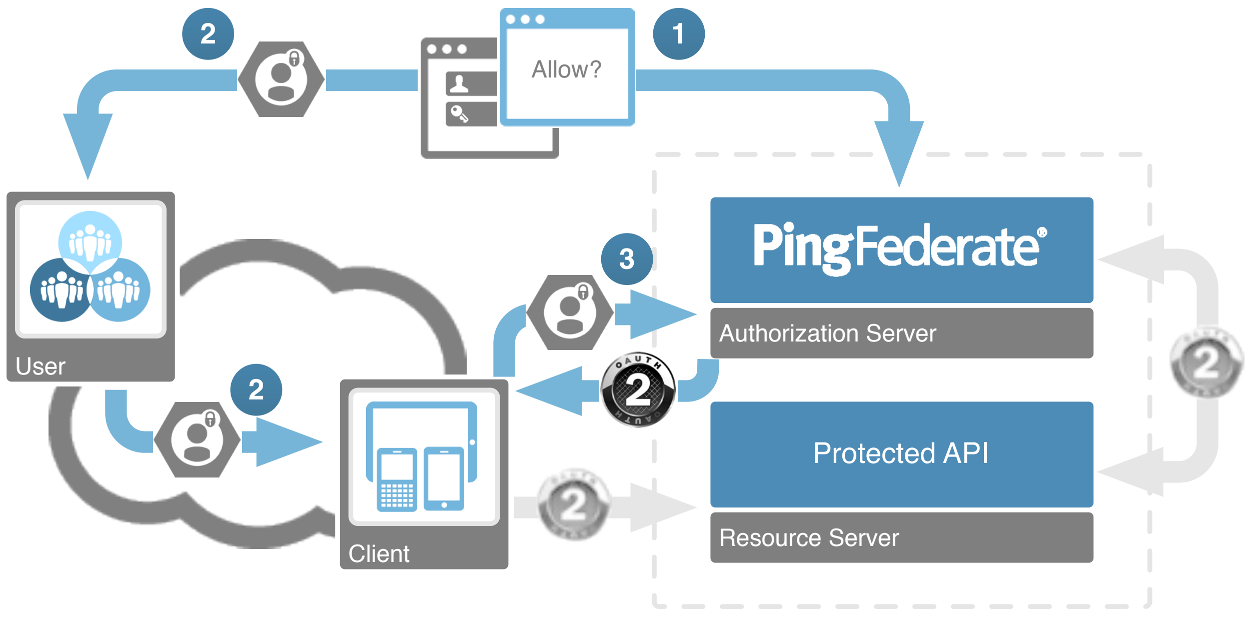
2.1 OAuth 2.0 Grant Types

OAuth 2.0 provides four standard grant types and an extension grant type that can be used to customize the authentication and authorization process depending on the application requirements.

Grant Type	Use Cases
Authorization Code	Used for most web and mobile application scenarios that want to call REST web services. Uses the user agent to transport an intermediate code, which is then exchanged for the OAuth2 tokens.
Implicit	Scenario where client is not able to safely hide the client secret (e.g. clientside JavaScript application). Uses the user agent to transport the OAuth2 tokens.
Resource Owner Password Credentials	When application needs to control the login form (e.g. native mobile app). Exchanges a username/password combination for the OAuth2 tokens.
Client Credentials	When a user is not involved. Access token only required for a service to call a REST web service. Exchanges the client credentials for an OAuth2 access token.
Extension Grant	Used to extend the OAuth 2.0 grant types for specific scenarios (e.g. the SAML bearer extension grant).

2.2 Authorization Code Grant

Authorization grant is a client redirect based flow. In this scenario, the user will be redirected to the PingFederate authorization endpoint via the user agent (i.e. web browser). This user agent will be used to authenticate the end user and allow them to grant access to the client (Step 1 below). Once the user has been authorized, an intermediate code will be granted by the authorization server and returned to the client application via the user agent (step 2). Lastly, the client will swap this code for an OAuth access token (step 3).



Capability	
Browser-based end user interaction	Yes
Can use external IDP for authentication	Yes
Requires client authentication	No*
Requires client to have knowledge of user credentials	No
Refresh token allowed	Yes
Access token is in context of end user	Yes

Note: Although the authorization code grant type does not require a client secret value, there are security implications to exchanging a code for an access token without client authentication.

2.2.1 Client Configuration

For the examples below, the following client information will be used:

Admin Label	**OAuth2 Parameter**	**Example Value**
Client ID	client_id	ac_client
Client Authentication	client_password	2Federate
Allowed Grant Types	response_type grant_type	Authorization Code • response_type of “code” • grant_type of “authorization_code” Refresh Token
Redirect URIs	redirect_uri	sample://oauth2/code/cb
Scope Settings (in AS settings) / Restrict Scopes	scope	edit

2.2.2 Request authorization from user and retrieve authorization code

To initiate the process, the client application will redirect the user to the authorization endpoint. This redirect will contain the applicable attributes URL encoded and included in the query string component of the URL.

Using the above parameters as an example, the application will redirect the user to the following URL:

```
https://localhost:9031/as/authorization.oauth2?
client_id=ac_client&
response_type=code&
scope=edit&
redirect_uri=sample%3A%2F%2Foauth2%2Fcode%2Fcb
```

This will initiate an authentication process using the browser (user agent). Once the user successfully completes the authorization request, they will be redirected with an authorization code to the redirect_uri value defined in the authorization request (if included) otherwise the user will be returned to the redirect_uri defined when the client was configured.

Note: For mobile scenarios, the redirect_uri may be a custom URL scheme that will cause the code to be returned to the native application.

Using the example above, a successful authorization request will result in the resource owner redirected to the following URL with the authorization code included as a “code” query string parameter:

```
sample://oauth2/code/cb?code=XYZ…123
```

Note:	If the authorization request also included a “state” value, this will also be included on this callback.

Note:	An error condition from the authentication / authorization process will be returned to this callback URI with “error” and “error_description” parameters.

The client will then extract the code value from the response and, optionally, verify that the state value matches the value provided in the authorization request.

2.2.3 Swap the authorization code for an access token

The final step for the client is to swap the authorization code received in the previous step for an access token that can be used to authorize access to resources. By limiting the exposure of the access token to a direct HTTPS connection between the client application and the authorization endpoint, the risk of exposing this access token to an unauthorized party is reduced.

For this to occur, the client makes a HTTP POST request to the token endpoint on the AS. This request will use the following parameters sent in the body of the request:

Item	Description
grant_type	Required to be “authorization_code”
code	The authorization code received in the previous step
redirect_uri	If this was included in the authorization request, it MUST also be included

This request should also authenticate as the pre-configured client using either HTTP BASIC authentication or by including the client_id and client_secret values in the request.

To retrieve the access token in the example, the following request will be made.

HTTP Request

```
POST https://localhost:9031/as/token.oauth2 HTTP/1.1
```

Headers

```
Content-Type:    application/x-www-form-urlencoded
Authorization:   Basic YWNfY2xpZW50OjJGZWRlcmF0ZQ==
```

Body

```
grant_type=authorization_code&code=XYZ…123
```

A successful response to this message will result in a 200 OK HTTP response and the access token (and optional refresh token) returned in a JSON structure in the body of the response.

HTTP Response

```
HTTP/1.1 200 OK
```

Headers

```
Content-Type:          application/json;charset=UTF-8
```

Body

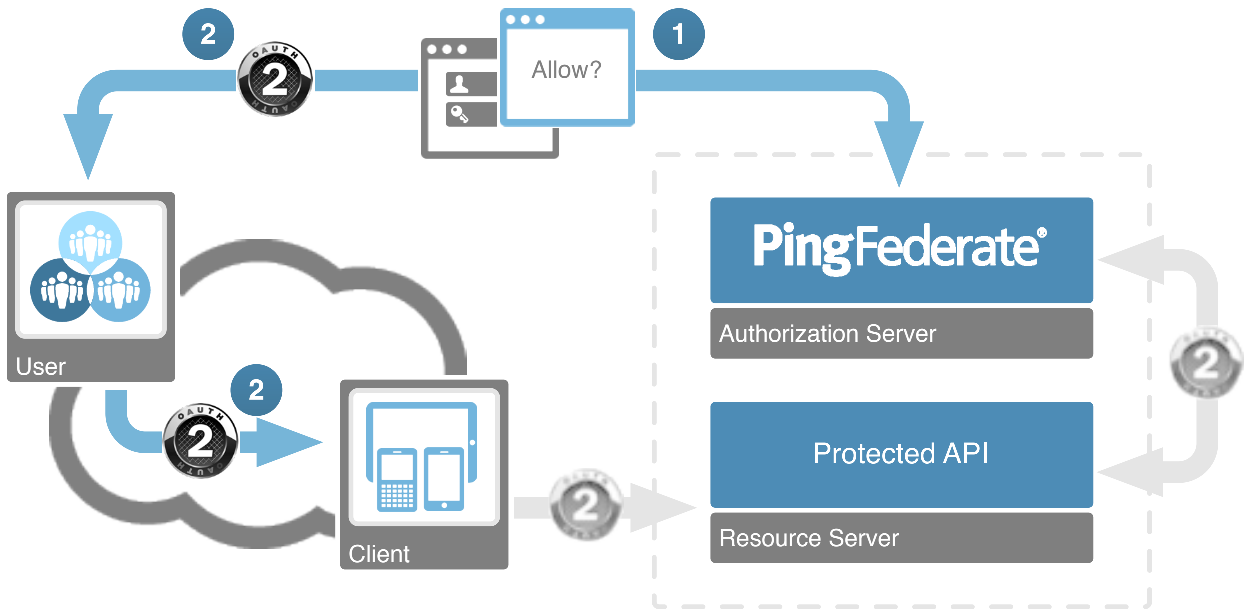
```
{
"access_token":"zzz…yyy",
"token_type":"Bearer",
"expires_in":14400,
"refresh_token":"123…789"
}
```

The application can now parse the access token and, if present, the refresh token to use for authorization to resources. If a refresh token was returned, it can be used to refresh access token once it expires.

2.3 Implicit Grant

The implicit grant is similar to an authorization code grant, however the user agent will receive an access token directly from an authorization request (rather than swapping an intermediate authorization code).

In this flow, the user requests authentication and authorization via the user agent (step 1 below). If authorized, the authorization server will redirect the user to a URL containing the access token in a URL fragment. The client can then parse this from the URL (step 2) to use for requests to protected resources.



This grant type is suitable for clients that are unable to keep a secret (i.e. client-side applications like JavaScript). The client is “mapped” to the authorization server via the redirect_uri, as there is no client secret to authenticate the client, the access token will be sent to a specific URL pre-negotiated between the client and the authorization server.

As the access token is provided to the client in the request URI, it is inherently less secure than the authorization code grant type. For this reason, an implicit grant type cannot take advantage of refresh tokens. Only access tokens can be provided via this grant type.

Capability	
Browser-based end user interaction	Yes
Can use external IDP for authentication	Yes
Requires client authentication	No
Requires client to have knowledge of user credentials	No
Refresh token allowed	No
Access token is in context of end user	Yes

2.3.1 Client Configuration

For the examples below, the following client information will be used:

Admin Label	OAuth2 Parameter	Example Value
Client ID	client_id	im_client
Client Authentication	client_password	None
Allowed Grant Types	response_type	Implicit • response_type of "token"
Redirect URIs	redirect_uri	sample://oauth2/implicit/cb
Scope Settings (in AS settings) / Restrict Scopes	scope	edit

2.3.2 Request authorization from user and retrieve access token

To initiate the process, the client application will redirect the user to the authorization endpoint. This redirect will contain the applicable attributes URL encoded and included in the query string component of the URL.

Using the above parameters as an example, the application will redirect the user to the following URL:

```
https://localhost:9031/as/authorization.oauth2?
client_id=im_client&
response_type=token&
scope=edit&
redirect_uri=sample%3A%2F%2Foauth2%2Fimplicit%2Fcb
```

This will initiate an authentication process using the browser (user agent). Once the user has authenticated and approved the authorization request, they will be redirected to the configured URI with the access token included as a fragment of the URL. A refresh token will NOT be returned to the client:

```
sample://oauth2/implicit/cb#
access_token=zzz...yyy&
token_type=bearer&
expires_in=14400
```

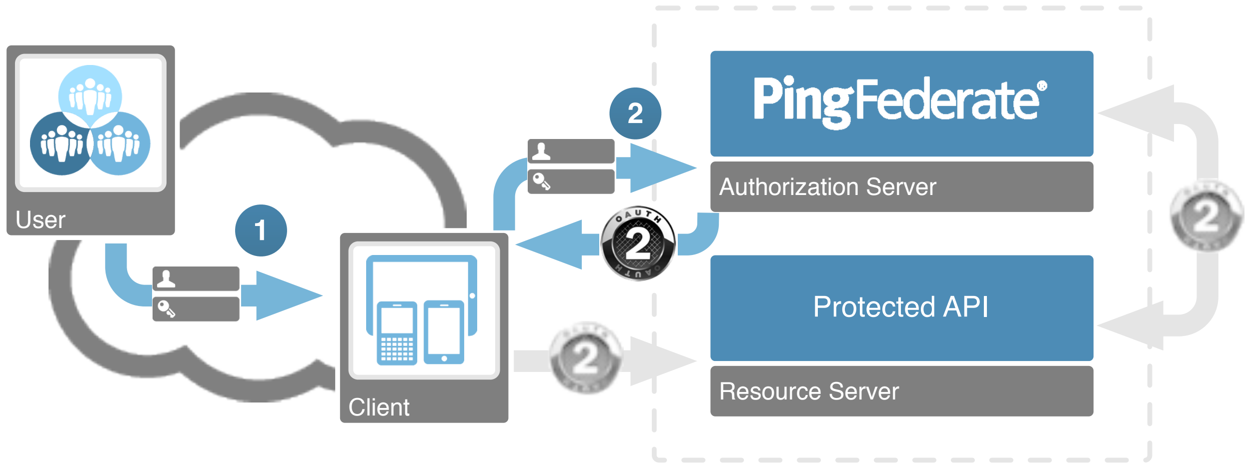
Note:	For mobile scenarios, the redirect_uri may be a custom URL scheme, which will cause the access token to be returned to the native application.

Note:	The implicit response is returned via a URL fragment. The fragment is only visible from client-side code. Therefore if you need to parse the values from server-side code, you must post the values to the server for parsing.

The application can now parse the access token from the URL fragment to use for authorization to API's.

2.4 Resource Owner Password Credentials (ROPC)

The ROPC grant type can be used in scenarios where an interactive user agent is not available, where specific design requirements warrant the use of a native application login interface, or for legacy reasons (i.e. retro-fitting a login form for OAuth2). In the ROPC grant type, the client captures the user credentials (step 1 below) and uses those credentials to swap for an access token (step 2).



Capability	
Browser-based end user interaction	No
Can use external IDP for authentication	No
Requires client authentication	No
Requires client to have knowledge of user credentials	Yes
Refresh token allowed	Yes
Access token is in context of end user	Yes

2.4.1 Client Configuration

For the examples below, the following client information will be used:

Admin Label	OAuth2 Parameter	Example Value
Client ID	client_id	ro_client
Client Authentication	client_password	2Federate
Allowed Grant Types	grant_type	Resource Owner Password Credentials • grant_type of “password” Refresh Token
Scope Settings (in AS settings) / Restrict Scopes	scope	edit

2.4.2 Request user authentication and retrieve access token

At this stage, the client displays a login form to the user and collects the credentials (e.g. username/password) and defined scope if required from the resource owner (user) and makes a HTTP POST to the token endpoint.

For the example below, the following credentials were received by the client and are used to request an access token:

Credential	Value
username	joe
password	2Federate

Note: The credentials passed via the Resource Owner Password Credential flow are processed through a PingFederate Password Credential Validator. These credentials do not have to be a username and password, they could be for example a username / PIN combination or another credential that is validated by a PCV.

HTTP Request

```
POST https://localhost:9031/as/token.oauth2 HTTP/1.1
```

Headers

```
Content-Type:       application/x-www-form-urlencoded
Authorization:      Basic cm9fY2xpZW50OjJGZWRlcmF0ZQ==
```

Body

```
grant_type=password&username=joe&password=2Federate&scope=edit
```

If successful, the client will receive a 200 OK response to this request and the access token (and optional refresh token) will be returned in a JSON structure:

HTTP Response

```
HTTP/1.1 200 OK
```

Headers

```
Content-Type:     application/json;charset=UTF-8
```

Body

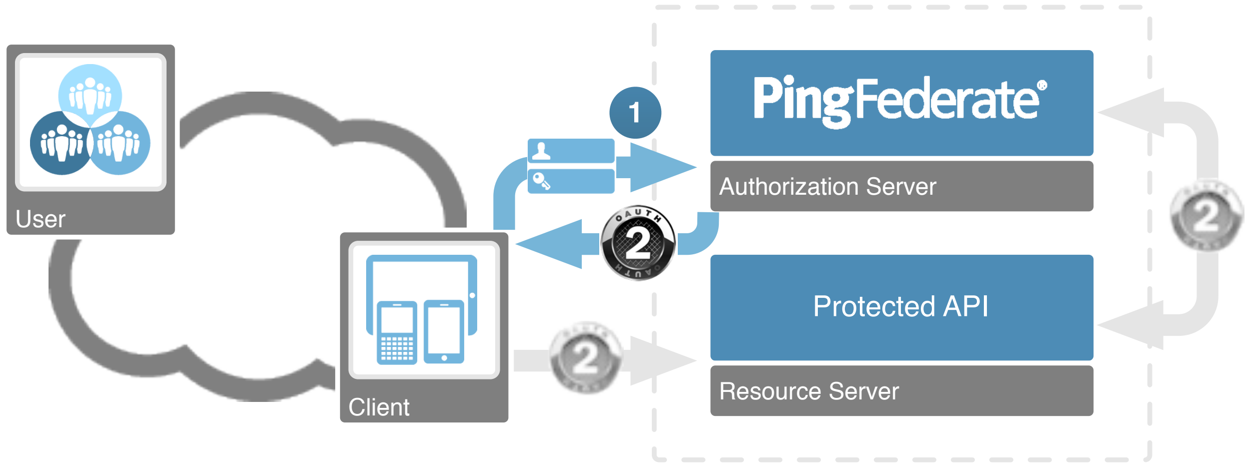
```
{
"access_token":"zzz…yyy",
"token_type":"Bearer",
"expires_in":14400,
"refresh_token":"123…789"
}
```

Note:	An error condition from the authentication / authorization process will be returned to this callback URI with "error" and "error_description" parameters.

The application can now parse the access token and, if present, the refresh token to use for authorization to resources. If a refresh token was returned, it can be used to refresh the access token once it expires.

2.5 Client Credentials

The client credentials type works in a similar way to the ROPC grant type and is used to provide an access token to a client based on the credentials or the client, not the resource owner. In this grant type, the client credentials are swapped for an access token (step 1 below).



Capability	
Browser-based end user interaction	No
Can use external IDP for authentication	No
Requires client authentication	Yes
Requires client to have knowledge of user credentials	No
Refresh token allowed	No
Access token is in context of end user	No

2.5.1 Client Configuration

For the examples below, the following client information will be used:

Admin Label	OAuth2 Parameter	Example Value
Client ID	client_id	cc_client
Client Authentication	client_password	2Federate
Allowed Grant Types	grant_type	Client Credentials • grant_type of "client_credentials"
Scope Settings (in AS settings) / Restrict Scopes	scope	edit

2.5.2 Request access token

The client makes a request (HTTP POST) to the token endpoint with the client credentials presented as HTTP Basic authentication:

HTTP Request

```
POST https://localhost:9031/as/token.oauth2 HTTP/1.1
```

Headers

```
Content-Type:     application/x-www-form-urlencoded
Authorization:    Basic Y2NfY2xpZW50OjJGZWRlcmF0ZQ==
```

Body

```
grant_type=client_credentials&scope=edit
```

Note: The client credentials can also be provided using the client_id and client_secret parameters in the contents of the POST.

The client will receive a response to this request. If successful, a 200 OK response will be received and the access token will be returned in a JSON structure. A refresh token will NOT be returned to the client.

HTTP Response

```
HTTP/1.1 200 OK
```

Headers

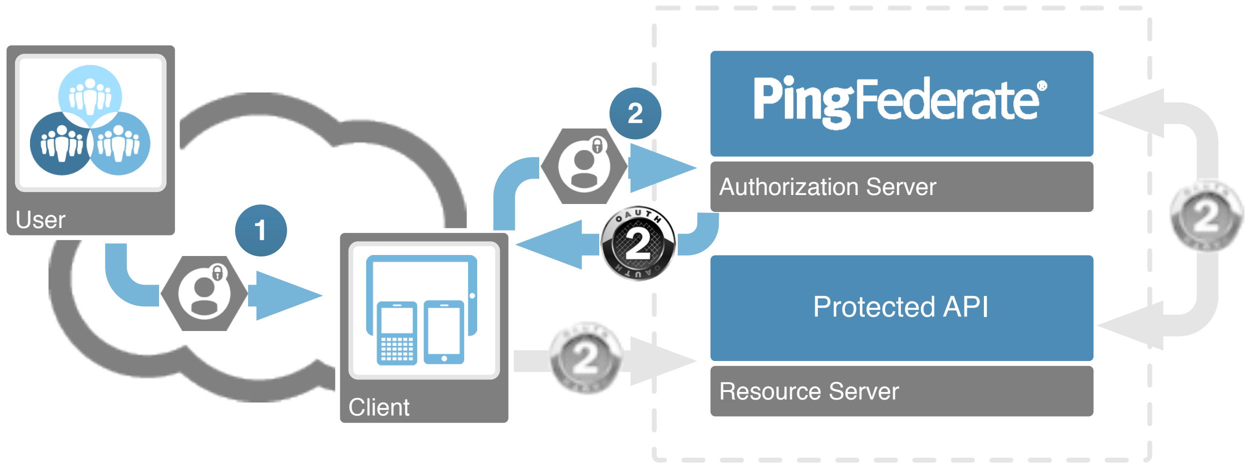
```
Content-Type:     application/json;charset=UTF-8
```

Body

```
{
"access_token":"zzz…yyy",
"token_type":"Bearer",
"expires_in":14400,
}
```

2.6 Extension grants (i.e. SAML Bearer)

The extension grant type provides support for additional grant types extending the OAuth2.0 specifications. An example is the use of the SAML 2.0 Bearer extension grant. In this grant type, a SAML assertion (indicated by step 1 below, however the process used to acquire this SAML assertion is out of scope of this document) can be exchanged for an OAuth 2.0 access token (step 2).



Capability	
Browser-based end user interaction	No*1
Can use external IDP for authentication	Yes*2
Requires client authentication	No
Requires client to have knowledge of user credentials	No
Refresh token allowed	No
Access token is in context of end user	Maybe*3

*1 – Although the grant type doesn't allow for user interaction, the process to generate the SAML assertion used in this flow can involve user interaction.
*2 – As long as the PingFederate AS is able to verify the SAML assertion, this assertion can be generated from a foreign STS.
*3 – Access token will be in the context of the subject of the SAML assertion, which may be an end-user a service or the client itself.

2.6.1 Client Configuration

For the examples below, the following client information will be used:

Admin Label	OAuth2 Parameter	Example Value
Client ID	client_id	saml_client
Client Authentication	client_password	2Federate
Allowed Grant Types	grant_type	Extension Grants • grant_type of "urn:ietf:params:oauth:grant-type:saml2-bearer"
Scope Settings (in AS settings) / Restrict Scopes	scope	edit

2.6.2 Request access token

At this stage, the client has a SAML assertion that it needs to exchange for an OAuth 2.0 access token. The process in which the client received the assertion is out of scope (i.e. bootstrap assertion, STS token exchange); however, the client would Base64 URL encode the assertion and include it in a HTTP POST to the token endpoint.

For the example below, the following SAML assertion (abbreviated for readability) was received by the client and is used to request an access token:

```
PHNhbWw6QXNzZXJ0aW9uIElEPSJTdXdCSDdiQjM3cWVmT0tycmlaZkc3Y09H
ZUMiIElzc3VlSW5zdGFudD0iMjAxNC0wMy0xMFQxNjo1MjozOS43NTRaIiBW
...
UmVmPjwvc2FtbDpBdXRobkNvbnRleHQ-PC9zYW1sOkF1dGhuU3RhdGVtZW50
Pjwvc2FtbDpBc3NlcnRpb24-
```

HTTP Request

```
POST https://localhost:9031/as/token.oauth2 HTTP/1.1
```

Headers

```
Content-Type:     application/x-www-form-urlencoded
Authorization:    Basic c2FtbF9jbGllbnQ6MkZlZGVyYXRl
```

Body

```
grant_type= urn:ietf:params:oauth:grant-type:saml2-bearer&assertion=
PHNhbWw6QXNzZXJ0aW9uIElEPSJTdXdCSDdiQjM3cWVmT0tycmlaZkc3Y09HZUMiIElzc3VlSW5zd
GFudD0iMjAxNC0wMy0xMFQxNjo1MjozOS43NTRaIiBW...UmVmPjwvc2FtbDpBdXRobkNvbnRleHQ
-PC9zYW1sOkF1dGhuU3RhdGVtZW50Pjwvc2FtbDpBc3NlcnRpb24-&scope=edit
```

Note: The client credentials can also be provided using the client_id and client_secret parameters in the contents of the POST.

The client will receive a response to this request. If successful, a 200 OK response will be received and the access token will be returned in a JSON structure. A refresh token will NOT be returned to the client.

HTTP Response

```
HTTP/1.1 200 OK
```

Headers

```
Content-Type:     application/json;charset=UTF-8
```

Body

```
{
"access_token":"zzz…yyy",
"token_type":"Bearer",
"expires_in":14400,
}
```

3 Refresh a token

If a refresh token was requested along with the access token, then the refresh token can be used to request a new access token without having to ask the user to re-authenticate. If the refresh token is still valid, then a new access token and refresh token will be returned to the client.

If the refresh token has been invalidated for any reason, then the client must require the user to re-authenticate to retrieve a new access token. The reasons for refresh tokens becoming invalid are:

- Refresh token has expired;
- Refresh token has been administratively revoked (separation / security reasons);
- User has explicitly revoked the refresh token

To refresh a token, the access token must have been requested with a grant type that supports refresh tokens (authorization code or resource owner password credentials). A request will then be made to the token endpoint with the grant_type parameter set to "refresh_token".

Note: A new access token can be requested with a scope of equal or lesser value than the original access token request. Refreshing an access token with additional scopes will return an error. If the scope parameter is omitted, then access token will be valid for the original request scope.

For this example, the authorization code client from above will be used to refresh the token

Admin Label	OAuth2 Parameter	Example Value
Client ID	client_id	ac_client
Client Authentication	client_password	2Federate
Allowed Grant Types	response_type grant_type	Authorization Code • response_type of "code" • grant_type of "authorization_code" Refresh Token
Redirect URIs	redirect_uri	sample://oauth2/code/cb
Scope Settings (in AS settings) / Restrict Scopes	scope	edit
Refresh Token	refresh_token	123...789

The following request is made by the client:

HTTP Request

```
POST https://localhost:9031/as/token.oauth2 HTTP/1.1
```

Headers

```
Content-Type:                 application/x-www-form-urlencoded
Authorization:                Basic YWNfY2xpZW50OjJGZWRlcmF0ZQ==
```

Body

```
grant_type=refresh_token&refresh_token=123…789
```

Note: A token can only be refreshed with the same or a lesser scope than the original token issued. If the token is being refreshed with the same scope as the original request, the scope parameter can be omitted. If a greater scope is required, the client must re-authenticate the user.

A successful response to this message will result in a 200 OK HTTP response and the following JSON structure in the body of the response:

HTTP Response

```
HTTP/1.1 200 OK
```

Headers

```
Content-Type:     application/json;charset=UTF-8
```

Body

```
{
"access_token":"aaa…ccc",
"token_type":"Bearer",
"expires_in":14400,
"refresh_token":"456…321"
}
```

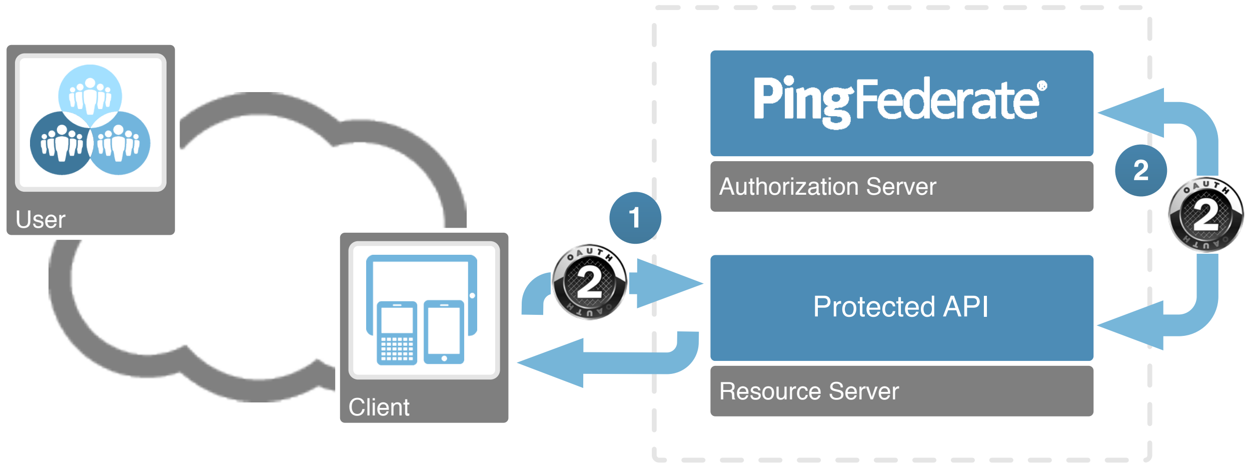
Note: Depending on the PingFederate configuration, the client could be configured to roll the refresh token returned from a refresh token request. i.e. a new refresh token is returned and the original refresh token is invalidated.

4 Use an access token

An access token can then be used as an authorization token (not to be confused with the authorization code in the authorization code grant type) to configured web services. To use an access token to access a protected resource, the access token must be passed to the resource server.

The client should use a bearer authorization method as defined in RFC 6750 to present the access token to the resource. The most common approach is to use the HTTP Authorization header and include the access token as a Bearer authorization credential; however, RFC 6750 also defines mechanisms for presenting an access token via query string and in a post body.

In the diagram below, the client presents the OAuth 2.0 access token to the protected resource (step 1). The resource then validates the access token before returning the requested resource (if authorized).



For example, to enact a GET request on a REST web service, given an access token “AAA...ZZZ”, the client makes the following HTTP request:

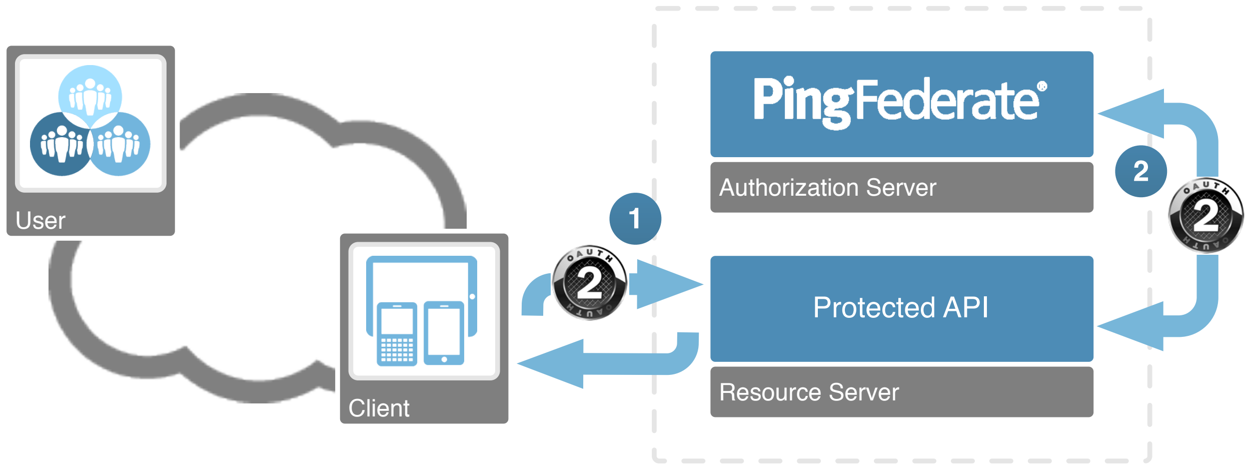
HTTP Request
`GET https://api.company.com/user HTTP/1.1`
Headers
`Authorization:   Bearer AAA...ZZZ`
Body
`<N/A>`

This will provide the access token to the resource server, which can then validate the token, verify the scope or the request, the identity of the resource owner and the client and perform the appropriate action if authorized.

API Developer Considerations

5 Validate a token

For an API developer to integrate with OAuth 2.0, the resource must accept and validate the OAuth 2.0 access token (step 1 below). Once the token has been received, the resource can then validate the access token against the PingFederate authorization server (step 2). The response from the access token validation will include attributes that the resource can use for authorization decisions.



Note:	This section will demonstrate the manual method of validating an access token through code. This effort could also be handled by an API gateway / service bus architecture.

Note:	The OAuth 2.0 specifications do not define a standard mechanism for access token validation. The process described in this section is specific to a PingFederate implementation.

5.1.1 Client Configuration

For the examples below, the following client information will be used:

Admin Label	**OAuth2 Parameter**	**Example Value**
Client ID	client_id	rs_client
Client Authentication	client_password	2Federate
Allowed Grant Types	grant_type	Access Token Validation (Client is a Resource Server) • grant_type of "urn:pingidentity.com:oauth2:grant_type:validate_bearer"

The API first needs to receive the access token from the client; this process will involve parsing the token via a process defined in RFC6750. See section 4 “Use an access token” above.

A request from a client would look similar to the following:

HTTP Request

```
GET https://api.company.com/user HTTP/1.1
```

Headers

```
Authorization:  Bearer AAA...ZZZ
```

Body

```
<N/A>
```

In order to fulfill the request, the API first extracts the access token from the authorization header, then queries the token endpoint of the PingFederate AS to validate the token:

HTTP Request

```
POST https://localhost:9031/as/token.oauth2 HTTP/1.1
```

Headers

```
Content-Type:              application/x-www-form-urlencoded
Authorization:             Basic cnNfY2xpZW50OjJGZWRlcmF0ZQ==
```

Body

```
grant_type=urn:pingidentity.com:oauth2:grant_type:validate_bearer&token=AAA..
.ZZZ
```

A successful response to this message will result in a 200 OK HTTP response and a JSON structure in the body of the response similar to the following:

HTTP Response

```
HTTP/1.1 200 OK
```

Headers

```
Content-Type:    application/json;charset=UTF-8
```

Body

```
{
"access_token": { "role":”all_access” },
"token_type":"Bearer",
"expires_in":14400,
"scope":"edit",
"client_id":"ac_client"
}
```

The resource server can then use this information to make an authorization decision and allow or deny the web request.

6 References

OAuth2 specifications & information
http://oauth.net/2

PingFederate Admin Guide
http://documentation.pingidentity.com/display/LP/Product+Documentation

Ping Identity Products and Downloads
https://www.pingidentity.com/support-and-downloads/