Hands-on su flussi OAuth/OIDC

Corso di formazione “Panoramica su OAuth2/OpenID Connect e sue applicazioni tramite il servizio INDIGO IAM”, 12-14 Maggio 2025, LNF

Federica Agostini, INFN CNAF
Enrico Vianello, INFN CNAF
Roberta Miccoli, INFN CNAF

Setup

Register your account in INDIGO IAM

- Browse the following link: https://iam-dev.cloud.cnaf.infn.it
- click on the INFN button
- submit the registration request and wait for an admin to approve you
- check your mailbox: you should receive a link to set the proper password
- after admin approval you can login both with username/password, or INFN AAI

Register an OAuth Client in INDIGO IAM

- click on *My clients* (left tab)
- click on *New client* (the green button)
- set at least the the client name equal to `demo_<last-name>` (a message *Change me please!* is present by default)
- click on the *Grant types* upper tab and select
  - *authorization_code*
  - *client_credentials*
  - *refresh_token*
  - *urn:ietf:params:oauth:grant-type:device_code*
- click the *Save client* green button (at the bottom page)
- once on your client details page, save your **Client id** (present in the *Main* upper tab) somewhere locally
- go to *Credentials* tab and save your **Client secret** locally

Device code flow exercise (1/4)

Device code flow exercise

1. Obtain an access token with the *device code flow* using the `verification_uri_complete` to approve your code
    - check the content of the access token
2. Obtain an access token with the *device code flow* using the `verification_uri` to approve your code
    - check the content of the access token
    - does it differ from the previous one? why?

Exercise 1: solution

Setup the following variables

```
CLIENT_ID=<your-client-id>
CLIENT_SECRET=<your-client-secret>
IAM_HOST=iam-dev.cloud.cnaf.infn.it
```

Ask for a grant in form of *device code*

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d client_id=${CLIENT_ID}
https://${IAM_HOST}/devicecode | jq
{
  "user_code": "VWR3KL",
  "device_code": " 1b6d500c-7faf-4f69-abd3-7e7a14ec7a5e ",
  "verification_uri_complete": " https://iam.test.example/device?user_code=VWR3KL ",
  "verification_uri": "https://iam.test.example/device",
  "expires_in": 600
}
```

Save the `device_code`, to be used in the token request

Click here and, after login, approve the Client (click on the *Authorize* green button)

Exercise 1: solution

Ask for an access token with the just obtained `device_code`

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d
grant_type=urn:ietf:params:oauth:grant-type:device_code  -d
device_code=1b6d500c-7faf-4f69-abd3-7e7a14ec7a5e  https://${IAM_HOST}/token | jq
{
  "access_token": " eyJraWQiOiJyc2ExIiwi... ",
  "token_type": "Bearer",
  "refresh_token": "eyJhbGciOiJub25lIn0...",
  "expires_in": 3599,
  "scope": "openid email profile offline_access",
  "id_token": "eyJraWQiOiJyc2ExIiwiYWxn..."
}
```

This `grant_type` indicates to IAM that your right to obtain an access token is due to the fact that you *own* a credential in form of **device code**

Copy this token in a variable, e.g.
`AT=eyJraWQiOiJyc2ExIiwi...`

The default scopes allowed for your client appears here and in the AT

***Tip**: the device code can be used only once and its lifetime is 10 minutes, so if you get an error try to repeat the steps more quickly*

Exercise 1: solution

Check the content of the encoded access token

```
echo $AT
eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ .eyJ3bGNnLnZlciI6IjEuMCIsInN1YiI6IjczZjE2ZDkzLTI0NDE
tNGE1MC04OGZmLTg1MzYwZDc4YzZiNSIsImF1ZCI6Imh0dHBzOlwvXC93bGNnLmNlcm4uY2hcL2p3dFwvdjFcL2Fue
SIsIm5iZiI6MTc0Njg5OTM5Miwic2NvcGUiOiJvcGVuaWQgb2ZmbGluZV9hY2Nlc3MiLCJpc3MiOiJodHRwczpcL1w
vaWFtLnRlc3QuZXhhbXBsZVwvIiwiZXhwIjoxNzQ2OTAyOTkyLCJpYXQiOjE3NDY4OTkzOTIsImp0aSI6ImM3ZmMyO
DI4LTM5OTgtNGE5ZS05NzUyLWExZDk5Mzg0YjEzMyIsImNsaWVudF9pZCI6ImRldmljZS1jb2RlLWNsaWVudCJ9  .Rd
dJJAhlKV3YARrJyqcoFoB-HhMiGnug2CD1UxP1c2rFLN2fI4EtSDbp-WKrYdvyfVFaECU8Z03SOk3ovgOI7urQDKWi
OoB7KM2Hp-Oi-eGJ4YiQcAU88ZPlm9ExHwjo23tURXMxNMW3oQ1F6Rfz8W95lpNVAk3ef4gkHCc0AVuphMTKyZhZk6
yYg-BpRoHqKyHSVKIBTY9AN3FWXijqC8_hvQSHyhGKzNmezQD9G2b05yN3cIzIrIhUgc2Pkqeekn-p4M0rMf6PplLk
aVIZZKUdDjjBGcUxo-rfXdwDvT6f33REFGf5r-96FHbSldxwd_LwEZ18M04MtPHFGFfGKw
```

Encoded header

Encoded payload

Signature

Exercise 1: solution

Decode the token and check the content of the access token payload

```
echo $AT | cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "80e5fb8d-b7c8-451a-89ba-346ae278a66f",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1746894757,
  "scope": "openid profile offline_access email",
  "iss": "https://iam.test.example/",
  "exp": 1746898357,
  "iat": 1746894757,
  "jti": "c6fbe29c-8f1b-4a15-8d08-5989166df136",
  "client_id": "device-code-client"
}
```

This is the uuid you see in your IAM profile

Those are the default access token scopes, also present in the token response

This is your CLIENT_ID

Exercise 2: solution

Ask for a *device code*

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d client_id=${CLIENT_ID}
https://${IAM_HOST}/devicecode | jq
{
  "user_code": "CC2KDW",
  "device_code": "86cf06fb-f7f6-4bc4-83fd-30a3f045cb4b ",
  "verification_uri_complete": "https://iam.test.example/device?user_code=CC2KDW",
  "verification_uri": "https://iam.test.example/device ",
  "expires_in": 600
}
```

Save the `device_code`, to be used in the token request

Click here and, after login, insert the `user_code` `CC2KDW`, then click on *Submit*. This step was skipped with the complete URI in Exercise 1. Now approve the Client (click on the *Authorize* green button) if you have clicked on *Prompt me again* in the previous exercise

Exercise 2: solution

Ask for an access token with the just obtained `device_code`

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d
grant_type=urn:ietf:params:oauth:grant-type:device_code -d
device_code=86cf06fb-f7f6-4bc4-83fd-30a3f045cb4b  https://${IAM_HOST}/token | jq
{
  "access_token": "eyJraWQiOiJyc2ExIiwi...",
  "token_type": "Bearer",
  "refresh_token": "eyJhbGciOiJub25lIn0...",
  "expires_in": 3599,
  "scope": "openid email profile offline_access",
  "id_token": "eyJraWQiOiJyc2ExIiwiYWxn..."
}
```

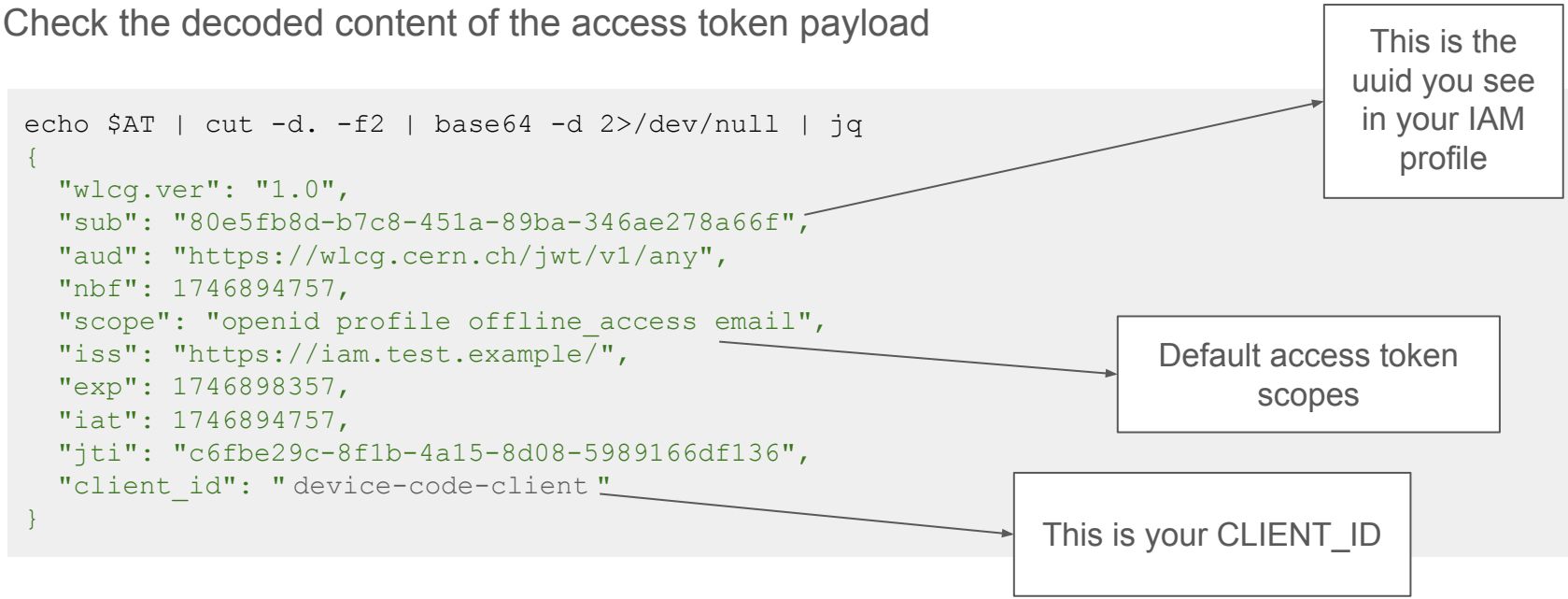
Copy this token in a variable, e.g.
`AT=eyJraWQiOiJyc2ExIiwi...`

***Tip**: the device code can be used only once and its lifetime is 10 minutes, so if you get an error try to repeat the steps more quickly*

Exercise 2: solution

Check the decoded content of the access token payload

```
echo $AT | cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "80e5fb8d-b7c8-451a-89ba-346ae278a66f",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1746894757,
  "scope": "openid profile offline_access email",
  "iss": "https://iam.test.example/",
  "exp": 1746898357,
  "iat": 1746894757,
  "jti": "c6fbe29c-8f1b-4a15-8d08-5989166df136",
  "client_id": "device-code-client"
}
```



The access token claims which differ with respect to the previous ones are the expiration of the AT (**exp**), the time it was issued at (**iat**) and the token identifier (**jti**)

Refresh token flow exercise (2/4)

Refresh token flow exercise

1. Obtain a refresh token (RT) using the device code flow
    - request the `openid` and `offline_access` scopes
    - check the content of the refresh token
2. Obtain an access token (AT) using the refresh token flow and the RT issued in the previous bullet point
    - check the content of the AT
3. Request another AT with the same RT and check the difference with the previous one
    - request for the the `openid` scope
4. Request another AT with the same RT and check the difference with the previous one
    - request for the the `email` scope
    - what do you think it will happen?

Exercise 1: solution

Setup the following variables

```
CLIENT_ID=<your-client-id>
CLIENT_SECRET=<your-client-secret>
IAM_HOST=iam-dev.cloud.cnaf.infn.it
```

Ask for a *device code*

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d scope="openid offline_access" -d
client_id=${CLIENT_ID} https://${IAM_HOST}/devicecode | jq
{
  "user_code": "WEHIR-",
  "device_code": " 1b6d500c-7faf-4f69-abd3-7e7a14ec7a5e ",
  "verification_uri_complete": " https://iam.test.example/device?user_code=WEHIR- ",
  "verification_uri": "https://iam.test.example/device",
  "expires_in": 600
}
```

The `offline_access` scope indicates you want to obtain a refresh token

Save the `device_code`, to be used in the token request

Click here and approve the user code

Exercise 1: solution

Obtained an AT and RT refresh token with the `device_code`

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d
grant_type=urn:ietf:params:oauth:grant-type:device_code -d
device_code=02739aa8-afba-490c-9712-39da986ce779  https://${IAM_HOST}/token | jq
{
  "access_token": "eyJraWQiOiJyc2ExIiwi...",
  "token_type": "Bearer",
  "refresh_token": "eyJhbGciOiJub25lIn0...",
  "expires_in": 3599,
  "scope": "openid offline_access",
  "id_token": "eyJraWQiOiJyc2ExIiwiYWxn..."
}
```

Copy the refresh token in a variable, e.g.
`RT=eyJraWQiOiJyc2ExIiwi...`

You have just obtained the originally requested scopes

They will also appear in this access token

Exercise 1: solution

Check the content of the decoded refresh token

```
echo $RT
eyJhbGciOiJub25lIn0.eyJleHAiOjE3NDcxNTg1OTIsImp0aSI6IjU5M2UwMDNmLWY5ZTYtNDNiZC1hYjI3LWFiYj
QxNDlmMmRhNyJ9.
```

Encoded header

Encoded payload

In IAM, **no token signature** is present, because for now the refresh token is saved in the database and when it is used, we can just check if it's present there rather than verify the signature

On the other hand, this RT can be used only by IAM to issue an access token (i.e. other authorization servers will luckily return a 401|3 error)

Exercise 1: solution

Decode the token and check the content of the refresh token payload

```
echo $RT | cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "exp": 1747158592,
  "jti": "593e003f-f9e6-43bd-ab27-abb4149f2da7"
}
```

It only presents an identifier, plus an expiration (default 30 days in IAM, but may be changed by admins)

Exercise 2: solution

This exercise requires you have obtained a refresh token in Exercise 1, which is saved in the RT variable

Ask for an access token with the just obtained `refresh_token`

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=refresh_token -d
refresh_token=${RT}  https://${IAM_HOST}/token | jq
{
  "access_token": "eyJraWQiOiJyc2ExIiwi... ",
  "token_type": "Bearer",
  "refresh_token": "eyJhbGciOiJub25lIn0...",
  "expires_in": 3599,
  "scope": "openid offline_access ",
  "id_token": "eyJraWQiOiJyc2ExIiwiYWxn..."
}
```

This `grant_type` indicates to IAM that your right to obtain an access token is due to the fact that you *own* a credential in form of **refresh token**

Copy the access token in a variable, e.g. `AT=eyJraWQiOiJyc2ExIiwi...`

The token response contains the same scopes as the ones originally requested (in Exercise 1, with the device code flow), identified by the **scope** key in this JSON

Exercise 2: solution

Insect the access token (decoded) payload

```
echo $AT | cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "73f16d93-2441-4a50-88ff-85360d78c6b5",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1746901582,
  "scope": "openid offline_access",
  "iss": "https://iam.test.example/",
  "exp": 1746905182,
  "iat": 1746901582,
  "jti": "bcebf573-bf6b-4ac6-a9e9-646d868b3731",
  "client_id": "device-code-client"
}
```

You are still recognized as the one requesting the token (through your UUID) even if 2 flows were performed this time

Also the token contains the originally requested scopes

Exercise 3: solution

This exercise requires you have obtained a refresh token in Exercise 1, which is saved in the RT variable

Ask for an access token with the `openid` scope using the `refresh_token` flow

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=refresh_token -d
refresh_token=${RT} -d scope="openid" https://${IAM_HOST}/token | jq
{
  "access_token": " eyJraWQiOiJyc2ExIiwi... ",
  "token_type": "Bearer",
  "refresh_token": "eyJhbGciOiJub25lIn0...",
  "expires_in": 3599,
  "scope": "openid",
  "id_token": "eyJraWQiOiJyc2ExIiwiYWxn..."
}
```

Copy the access token in a variable, e.g. `AT=eyJraWQiOiJyc2ExIiwi...`

`openid` is present in the token response and allowed since it's a subset of the originally granted scopes

Exercise 3: solution

Insect the access token (decoded) payload

```
echo $AT | cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "73f16d93-2441-4a50-88ff-85360d78c6b5",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1746902762,
  "scope": "openid",
  "iss": "https://iam.test.example/",
  "exp": 1746906362,
  "iat": 1746902762,
  "jti": "08735800-e4f8-4c6d-ad44-f36ed51dd3a5",
  "client_id": "device-code-client"
}
```

The access token claim which differ with respect to the previous ones is the **scope**, together with the AT expiration (**exp**), time when it was issued (**iat**) and the token identifier (**jti**)

Exercise 4: solution

This exercise requires you have obtained a refresh token in Exercise 1, which is saved in the RT variable

Ask for an access token with the `email` scope using the `refresh_token` flow

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=refresh_token -d
refresh_token=${RT} -d scope="email" https://${IAM_HOST}/token | jq
{
  "error": "invalid_scope",
  "error_description": "Up-scoping is not allowed."
}
```

The email scope has never been approved by the user !

Client credentials flow exercise (3/4)

Client credentials flow exercise

1. Obtain an access token with the *client credentials* flow
    - check the content of the AT
2. Obtain an access token with the `openid` and `offline_access` scopes
    - do you expect to receive also a refresh token?
3. Obtain an access token with the `email` scopes
    - why you don’t have the same error as with the refresh token flow?
4. Obtain an access token with the https://storm.test.example audience
    - check the content of the AT

Exercise 1: solution

Setup the following variables

```
CLIENT_ID=<your-client-id>
CLIENT_SECRET=<your-client-secret>
IAM_HOST=iam-dev.cloud.cnaf.infn.it
```

Ask for a grant in form of *client credential*

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=client_credentials
https://${IAM_HOST}/token | jq
{
  "access_token": "eyJraWQiOiJyc2ExIiwiYWx...",
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": "openid profile email offline_access"
}
```

Copy this token in a variable, e.g.
`AT=eyJraWQiOiJyc2ExIiwi...`

The default scopes allowed for your client appears here and in the AT

Exercise 1: solution

Insect the access token (decoded) payload

```
echo $AT | cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "client-cred",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1746904998,
  "scope": "openid profile email offline_access",
  "iss": "https://iam.test.example/",
  "exp": 1746908598,
  "iat": 1746904998,
  "jti": "1da04872-b730-491a-b4fe-48c15142e597",
  "client_id": "client-cred"
}
```

Also the token contains the originally requested scopes

The client credential flow does not required user's intervention, it acts as service account which asks for token by itself

- then, the token **sub** is equal to the **client_id**
- the token will never contain user's information (email, username, IAM group membership, etc)

Exercise 2: solution

Ask for a token with the `offline_access` scope

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=client_credentials -d
scope="offline_access" https://${IAM_HOST}/token | jq
{
  "access_token": "eyJraWQiOiJyc2ExIiwiYWxnIjoi...",
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": "offline_access"
}
```

The `offline_access` scope is returned in the token response, but NO refresh token is issued

In fact, a new access token can be requested any time from a client credentials client (without needing a long-lived RT)

Exercise 3: solution

Ask for a token with the `email` scope

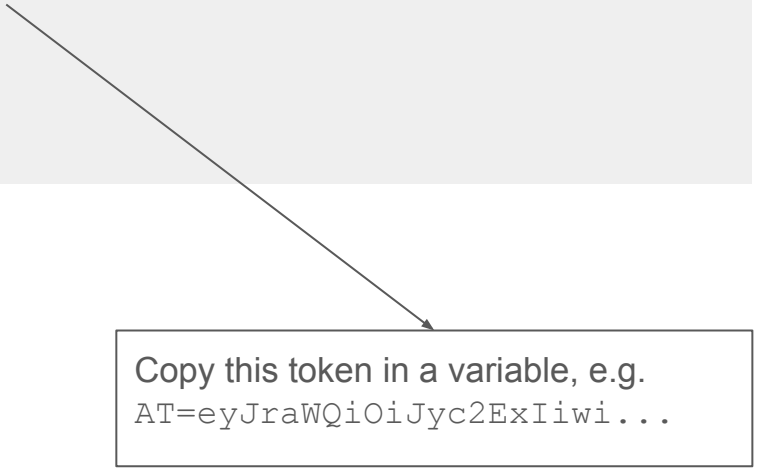
```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=client_credentials -d
scope="email" https://${IAM_HOST}/token | jq
{
  "access_token": "eyJraWQiOiJyc2ExIiwiYWxnIjoiUlM...",
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": "email"
}
```

The `email` scope is returned in the token response since it is not bounded to a client consent phase (i.e. the client is able to ask any scope it is allowed to receive)

Exercise 4: solution

Ask for a token with the `https://storm.test.example` audience

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=client_credentials -d
audience="https://storm.test.example" https://${IAM_HOST}/token | jq
{
  "access_token": "eyJraWQiOiJyc2ExIiwiYWxnIjoiUl...",
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": "openid profile offline_access email"
}
```



Exercise 4: solution

Insect the access token (decoded) payload

```
$ echo $AT | cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "client-cred",
  "aud": "https://storm.test.example ",
  "nbf": 1746906312,
  "scope": "openid profile offline_access email",
  "iss": "https://iam.test.example/",
  "exp": 1746909912,
  "iat": 1746906312,
  "jti": "f26f3350-d971-4daf-b513-494a7a9e9d52",
  "client_id": "client-cred"
}
```

The **aud** claim present in the AT must be the same as the one the token is intended for

Here, we could be authorized by a StoRM WebDAV service for instance

Token exchange flow exercise (4/4)

Prerequisite

Register a new OAuth Client in IAM

- set at least the the client name equal to **`demo_exchange_<last-name>`**
- click the *Save client* green button (at the bottom page)
- once on your client details page, save your **Client id** (present in the *Main* upper tab) somewhere locally
- go to *Credentials* tab and save your **Client secret** locally
- communicate the Client id to an IAM Admin (basically, teachers) and wait for them to add the `urn:ietf:params:oauth:grant-type:token-exchange` grant type to your client

The next token exchange flow exercises require that you request the

subject token → with the **demo_<last-name>** client
access token → with the **demo_exchange_<last-name>** client

Token exchange flow exercise

1. Obtain an access token with the `openid` and `profile` scopes
    - request the subject token with *client credentials flow* and `email` and `offline_access` scopes
    - check the content of the access token
2. Obtain an access token with the `openid` and `profile` scopes, but
    - request the subject token with *device code flow* and `email` and `offline_access` scopes
    - check the content of the access token
3. Obtain an access token with the https://storm.test.example audience and `email` scope
    - request the subject token with https://fts.test.example audience
    - find a real use-case for this exercise
    - check the content of the access token
4. Try to obtain an access token without specifying the requested scopes

Exercise 1: solution

Setup the following variables

```
SUB_CLIENT_ID=<demo-client-id>
SUB_CLIENT_SECRET=<demo-client-secret>
CLIENT_ID=<demo-exchange-client-id>
CLIENT_SECRET=<demo-exchange-client-secret>
IAM_HOST=iam-dev.cloud.cnaf.infn.it
```

Get a `SUBJECT_TOKEN` with the `email` and `offline_access` scopes

```
SUBJECT_TOKEN=$(curl -s -L -u ${SUB_CLIENT_ID}:${SUB_CLIENT_SECRET} -d
grant_type=client_credentials -d scope="email offline_access" https://${IAM_HOST}/token |
jq -r .access_token |  tr -d '"')
```

Here the client credential flow is used

Exercise 1: solution

Ask for a token with the `openid` and `profile` scopes (which were not included in the SUBJECT_TOKEN)

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d
grant_type=urn:ietf:params:oauth:grant-type:token-exchange -d scope="openid profile" -d
subject_token=${SUBJECT_TOKEN}  https://${IAM_HOST}/token | jq
{
  "access_token": " eyJraWQiOiJyc2ExIiwiYWxnIjoi... ",
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": "openid profile",
  "issued_token_type": "urn:ietf:params:oauth:token-type:jwt"
}
```

This `grant_type` indicates to IAM that your right to obtain an access token is due to the fact that you *own* a credential in form of AT (the **subject token**)

Copy this token in a variable, e.g.
`AT=eyJraWQiOiJyc2ExIiwi...`

Despite the refresh token flow, the requested scopes are returned as token response even if not originally granted – this is allowed by the *token exchange* and it's the reason why an IAM admin has to modify your client in order to enable this flow

Exercise 1: solution

Insect the access token (decoded) payload

```
echo $AT | cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "client-cred",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "act": {
     "sub": "token-exchange-actor"
  },
  "nbf": 1746910139,
  "scope": "openid profile",
  "iss": "https://iam.test.example/",
  "exp": 1746913739,
  "iat": 1746910139,
  "jti": "fe0476ce-6980-45fa-9faf-d21d390b2563",
  "client_id": "token-exchange-actor"
}
```

Since the subject token does not carry any user information, here also there is no reference to the user

Exercise 2: solution

Get a device code with the `email` and `offline_access` scopes

```
curl -s -L -u ${SUB_CLIENT_ID}:${SUB_CLIENT_SECRET} -d scope="email offline_access" -d
client_id=${SUB_CLIENT_ID} https://${IAM_HOST}/devicecode | jq
{
  "user_code": "WVZQPW",
  "device_code": " 4f464aa6-aba2-4584-83cd-84d3e81c8a1d ",
  "verification_uri_complete": "https://iam.test.example/device?user_code=WVZQPW",
  "verification_uri": "https://iam.test.example/device",
  "expires_in": 600
}
```

Approve the user's code and get an access token (SUBJECT_TOKEN)

```
SUBJECT_TOKEN=$(curl -s -L -u ${SUB_CLIENT_ID}:${SUB_CLIENT_SECRET} -d
grant_type=urn:ietf:params:oauth:grant-type:device_code -d
device_code= 4f464aa6-aba2-4584-83cd-84d3e81c8a1d  https://${IAM_HOST}/token | jq -r
.access_token |  tr -d '"')
```

Exercise 2: solution

Ask for a token with the `openid` and `profile` scopes

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d
grant_type=urn:ietf:params:oauth:grant-type:token-exchange -d scope="openid profile" -d
subject_token=${SUBJECT_TOKEN} https://${IAM_HOST}/token | jq
{
  "access_token": " eyJraWQiOiJyc2ExIiwiYWxnIjoiU... ",
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": "openid profile",
  "id_token": "eyJraWQiOiJyc2ExIiwiYWxnIjoi...",
  "issued_token_type": "urn:ietf:params:oauth:token-type:jwt"
}
```

Copy this token in a variable, e.g.
`AT=eyJraWQiOiJyc2ExIiwi...`

Now we get also an ID token, since a user (myself) has been previously authenticated

Exercise 2: solution

Inspect the access token (decoded) payload

```
echo $AT | cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "73f16d93-2441-4a50-88ff-85360d78c6b5 ",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "act": {
     "sub": "token-exchange-actor "
  },
  "nbf": 1746912119,
  "scope": "openid profile",
  "iss": "https://iam.test.example/",
  "exp": 1746915719,
  "iat": 1746912119,
  "jti": "94176e69-ed75-428a-b068-6d1ba951b38f",
  "client_id": " token-exchange-actor "
}
```

This is my UUID, propagated from the SUBJECT_TOKEN

This is the client requesting the access token

No ID of the Client requesting the subject token is present in the AT as this is an example of impersonation !

Exercise 3: solution

This exercise is useful to simulate a job transmitted by a client → FTS → storage (storm)

Get a `SUBJECT_TOKEN` with the `https://fts.test.example` audience and the client credential flow to simulate a job submitted to FTS

```
SUBJECT_TOKEN=$(curl -s -L -u ${SUB_CLIENT_ID}:${SUB_CLIENT_SECRET} -d
grant_type=client_credentials -d audience= https://fts.test.example
https://${IAM_HOST}/token | jq -r .access_token |  tr -d '"')
```

Ask for a token with a `https://storm.test.example` audience and `email` scope (to simulate FTS doing a refresh flow such to have the proper audience)

```
AT=$(curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d
grant_type=urn:ietf:params:oauth:grant-type:token-exchange -d scope="email" -d
subject_token=${SUBJECT_TOKEN} -d audience= https://storm.test.example
https://${IAM_HOST}/tok
en | jq -r .access_token |  tr -d '"')
```

Exercise 3: solution

Insect the access token (decoded) payload

```
echo $AT | cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "client-cred",
  "aud": "https://storm.test.example ",
  "act": {
     "sub": "token-exchange-actor"
  },
  "nbf": 1746913264,
  "scope": "email",
  "iss": "https://iam.test.example/",
  "exp": 1746916864,
  "iat": 1746913264,
  "jti": "e61e86d5-2798-4a24-81fe-215e0ab2ea99",
  "client_id": "token-exchange-actor"
}
```

The new token is audience-restricted for StoRM

Exercise 4: solution

Get a `SUBJECT_TOKEN` with the default scopes and the *client credential* flow

```
SUBJECT_TOKEN=$(curl -s -L -u ${SUB_CLIENT_ID}:${SUB_CLIENT_SECRET} -d
grant_type=client_credentials https://${IAM_HOST}/token | jq -r .access_token |  tr -d
'"')
```

Ask for an access token with the default `scopes`

```
curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d
grant_type=urn:ietf:params:oauth:grant-type:token-exchange -d
subject_token=${SUBJECT_TOKEN} https://${IAM_HOST}/token | jq
{
  "error": "invalid_request",
  "error_description": "The scope parameter is required for a token exchange request!"
}
```

The token exchange requires that the scopes are explicitly requested