Introduction to OAuth and its applications

CNAF SD - Federica Agostini

12/01/2023

Outline

- Brief introduction to OAuth
- JSON Web Tokens
- OAuth grant types
  - authorization code flow
  - device code flow
  - refresh token flow
  - client credentials
  - token exchange
- oidc-agent
- Usage in WLCG

A brief introduction to OAuth

Core technologies in AAI

- **OAuth 2**
  - A standard framework for **delegated authorization**
  - Widely adopted in industry
  - Main specification is RFC 6749
- **OpenID Connect (OIDC)**
  - An **authentication** layer built on top of OAuth 2
  - Core specification
- **JSON Web Tokens (JWTs)**
  - A **compact, URL-safe** means of representing attributes (**claims**) to be transferred between two or more parties
  - Main specification is RFC 7519

BUNDESREPUBLIK DEUTSCHLAND
FEDERAL REPUBLIC OF GERMANY / REPUBLIQUE FEDERALE D'ALLEMAGNE
PERSONALAUSWEIS
IDENTITY CARD / CARTE D'IDENTITE

T22000129

Name/Surname/Nom
MUSTERMANN
GEB. GABLER

Vornamen/Given names/Prénoms
ERIKA

Geburtstag/Date of birth/Date de naissance
12.08.1964

Staatsangehörigkeit/Nationality/Nationalité
DEUTSCH

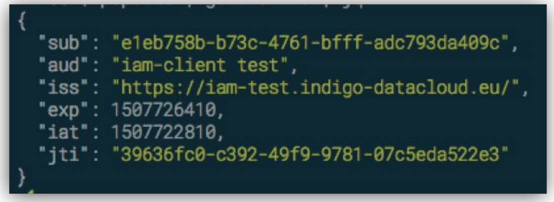
Geburtsort/Place of birth/Lieu de naissance
BERLIN

Gültig bis/Date of expiry/Date d'expiration
31.10.2020

938568

Unterschrift der Inhaberin/des Inhabers - Signature of bearer - Signature de la titulaire/du titulaire

MUSTER

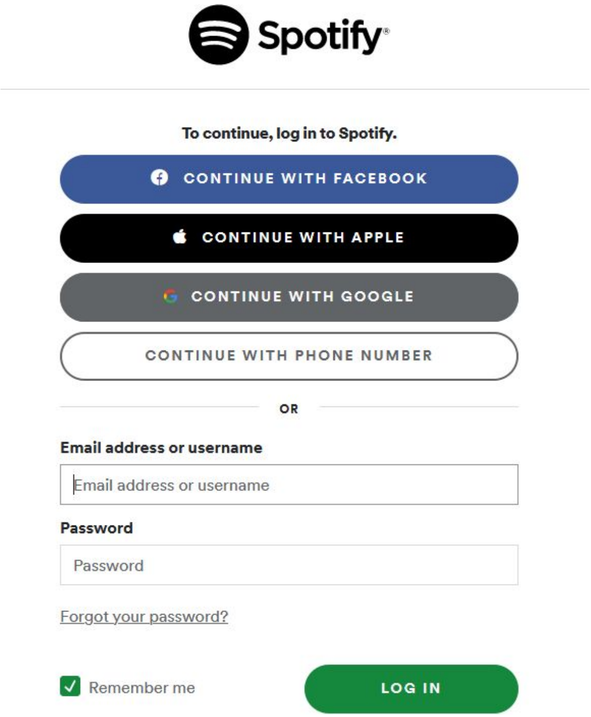


```
{
  "sub": "e1eb758b-b73c-4761-bfff-adc793da409c",
  "aud": "iam-client test",
  "iss": "https://iam-test.indigo-datacloud.eu/",
  "exp": 1507726410,
  "iat": 1507722810,
  "jti": "39636fc0-c392-49f9-9781-07c5eda522e3"
}
```

OAuth 2 features that matter

- OAuth uses **authorization tokens** between users and service providers to **prove an identity without sharing password data**
- Heavily relies on web technology
- Allows for example to login on a service using another social account
- Defines authorization workflows for web, desktop and mobile applications

Example: https://accounts.spotify.com/en/login



To continue, log in to Spotify.

Email address or username: Email address or username

Password: Password

Forgot your password?

☑ Remember me

OAuth 2 features that matter

- Enables **Single Sign-On** (SSO), based on strong authentication mechanisms, including multi-factor authentication
- Gives users more control over their data → they can **selectively grant access** to the scopes an application ask for
- **Mandates the use of TLS** (Transport Layer Security)
- **Easy to implement**
- Tokens can be (and usually are) **self-contained**, *i.e.* their **integrity and validity** can be verified **without calling back the token issuer**

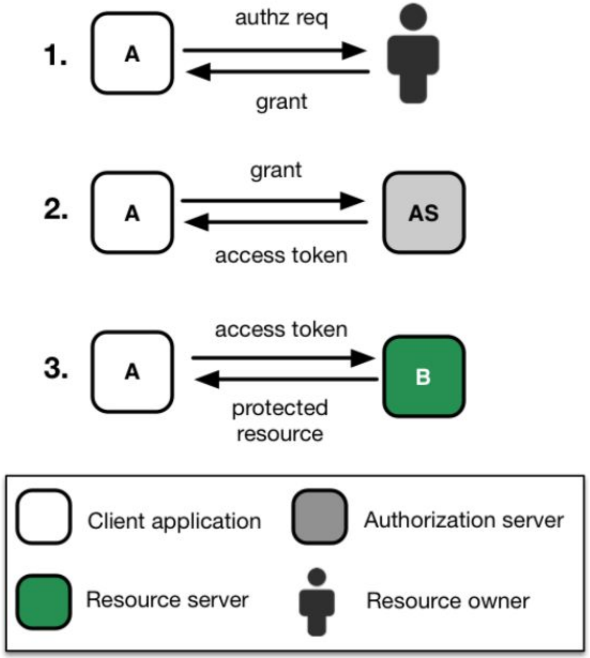
OAuth 2 roles

- **Resource owner**
  - A **user** that owns resources hosted at a service
- **Client**
  - An **application** that wants to have delegated access to user resources
  - It has to be registered on the Authorization Server
  - *Relying Party* (RP) in OIDC
- **Authorization Server (AS)**
  - A service that authenticates users and Clients
  - It **issues tokens** to Clients that can be used to access user resources
  - *OpenID Provider* (OP) in OIDC
- **Resource Server (RS)**
  - A service that **holds protected resources** (*e.g.,* user data)
  - It grants access based on tokens issued by the Authorization Server and presented by a Client
  - It has to validate the access token
  - Not mandatory to register a RS on the Authorization Server

The Authorization Server may be the same as the Resource Server

Authorization flow in theory

1. Authorization request to the resource owner
   - The Client (**A**) requests authorization from the resource owner to access a resource within a defined **scope**
     - the authorization request can be performed indirectly via the Authorization Server (**AS**)
   - The Client receives an **authorization grant**, which is a credential representing the resource owner's authorization
     - it depends on the authorization flow (aka *grant type*) used by the Client to perform the authorization requests
2. Authorization request to the AS token endpoint
   - The Client requests for an **access token** by authenticating with the AS and presenting the authorization grant
     - additional tokens can be requested at this stage



Authorization flow in theory

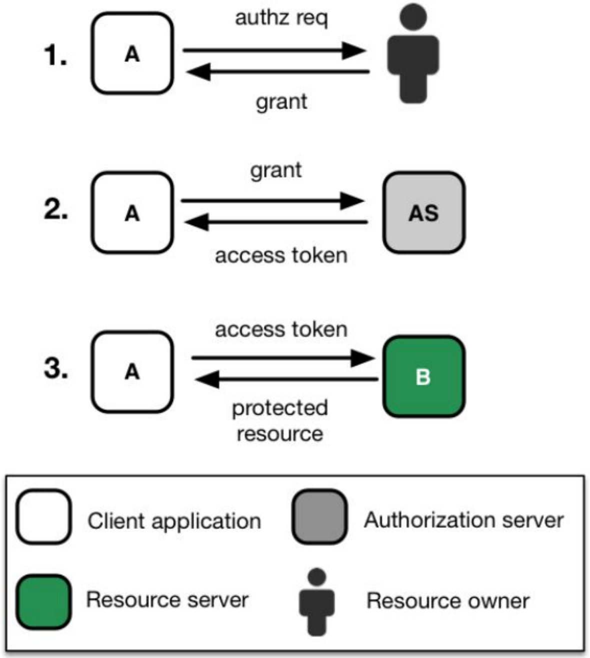
3. Access to the protected resource

   ○ The Client requests the protected resource from the Resource Server (**B**) and authenticates by presenting the access token

   ○ The RS validates the access token, and if valid, serves the request

   ○ Access is granted/denied according to the contents of the access token

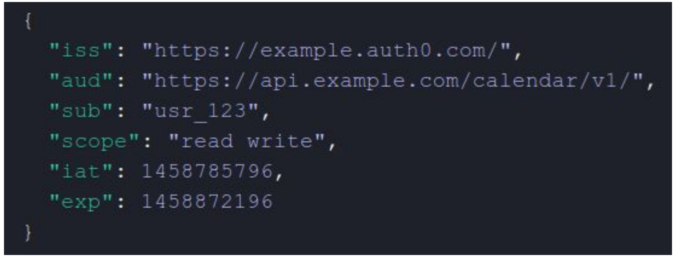
     ■ local policies that map token claims into permissions may be applied by the RS



OAuth/OIDC token types

Access Token (AT)

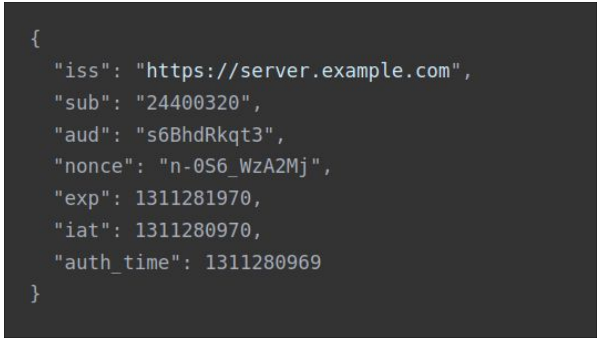
- Defined within OAuth 2
- Is a string that the Client uses to make requests to the Resource Server
  - do not have to be in any particular format
- AT may be *bearer tokens*, meaning that those who hold the token can use it



```
{
  "iss": "https://example.auth0.com/",
  "aud": "https://api.example.com/calendar/v1/",
  "sub": "usr_123",
  "scope": "read write",
  "iat": 1458785796,
  "exp": 1458872196
}
```

ID token

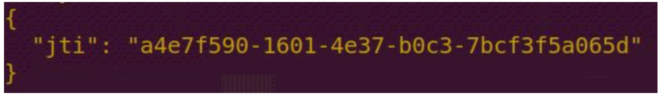
- Defined within OIDC
- Is a JWT intended to be read by the OAuth Client, which is the *audience* of the token
- May also contain information about the user such as their name or email address
  - client applications can use it to build a user profile to personalize the user experience



```
{
  "iss": "https://server.example.com",
  "sub": "24400320",
  "aud": "s6BhdRkqt3",
  "nonce": "n-0S6_WzA2Mj",
  "exp": 1311281970,
  "iat": 1311280970,
  "auth_time": 1311280969
}
```

Refresh token (RT)

- Defined within OAuth 2
- Is a string that the OAuth Client can use to get a new AT without the user's interaction
- Must not allow the Client to gain any access beyond the scope of the original grant



```
{
  "jti": "a4e7f590-1601-4e37-b0c3-7bcf3f5a065d"
}
```

Examples of scopes

Standard commonly used OAuth/OIDC scopes

- **openid** signal that the Client wants to receive authentication information about the user
- **profile** used to request profile information (name, address, *etc*)
- **email** used to request access to the user's email (name, address)
- **offline_access** used to request refresh tokens, needed to renew access tokens

WLCG-defined scopes (detailed later)

- **storage.read**, **storage.create**, **storage.modify**, **storage.stage** used to manage access to WLCG storage
- **compute.read**, **compute.modify**, **compute.create**, **compute.cancel** used to manage access to WLCG computing resources
- **wlcg.groups** used to request the inclusion of group information in tokens

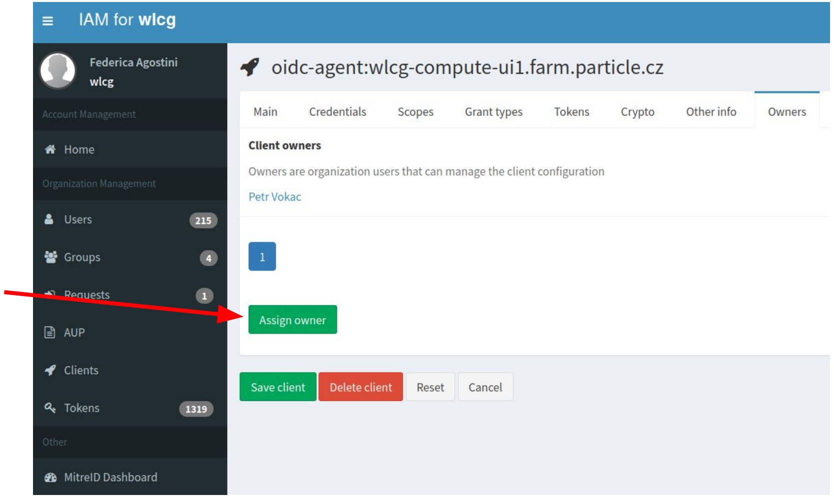
OAuth Client registration

- Clients which interact with an Authorization Server need to be **registered**
- When a client is registered, it is assigned a unique identifier (**client_id**) and a **credential**, either
  - a password (**client_secret**), or
  - an assertion (in the form of a **JWT**)

  Credentials are required in most of the OAuth/OIDC flows or to access specific endpoints, where different privileges may be assigned to different Clients
- Client registration is necessary to integrate any application that needs to "drive" an authorization flow
  - *e.g.*, if your web app needs to authenticate users through a "Login" button, you need to register a Client
- Registration is not needed for Resource Servers (*e.g.*, a REST API)
  - Resource Servers need to validate the ATs before to allow/deny access to protected resources

OAuth Client registration in IAM

- In INDIGO IAM versions >= 1.8.0:
  - users own their newly created clients
  - users can reclaim their old clients through the registration token (once)
  - admins can assign to one or more users the ownership of a client
- A list of System Scopes may be defined in IAM so that each newly registered Client has
  - default access to them, if they are declared as **default**
  - access needs to be granted by an admin, if they are declared as **restricted**
  - access has to be explicitly requested, but there is no need of intervention from an admin
- Client registration in IAM through
  - web application
  - API

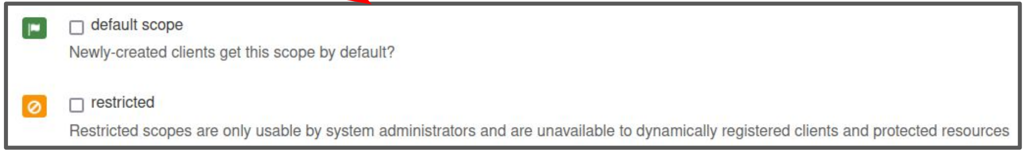


oidc-agent:wlcg-compute-ui1.farm.particle.cz

Client owners

Owners are organization users that can manage the client configuration

Petr Vokac



□ default scope

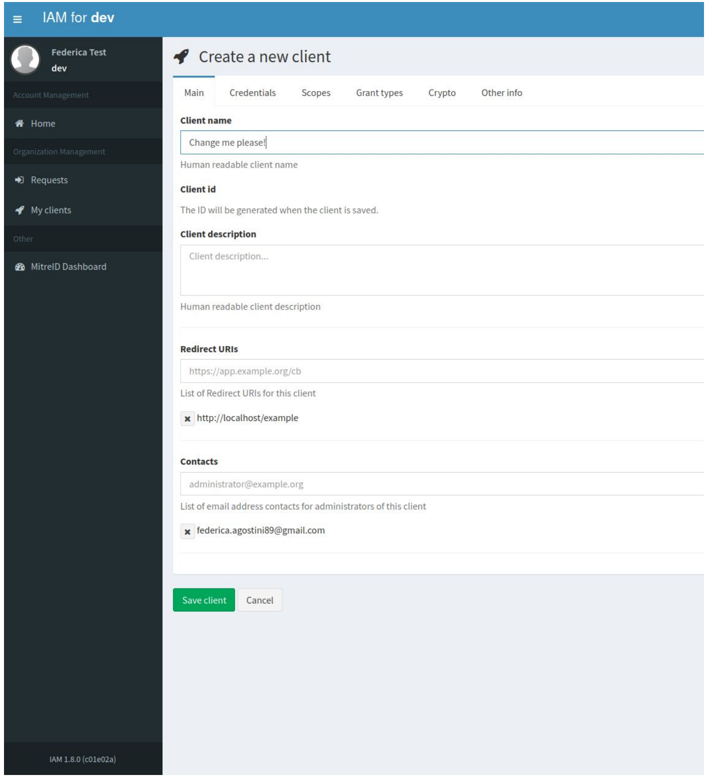
Newly-created clients get this scope by default?

□ restricted

Restricted scopes are only usable by system administrators and are unavailable to dynamically registered clients and protected resources

Client registration in IAM through web interface

- Documentation here
- One can create a *New client* from the *My Client* link on the left navigation bar of the IAM dashboard
- Minimum information required for a web app which needs to authenticate users through a “Login with IAM” button is
  - **Client name** choose a name for your Client
  - **Redirect URIs** one or more URIs for your web app (it is required when the *authorization code flow* is enabled – see later)
- Then, IAM will generate a `client_id` and `client_secret` which have to be saved into your web app
- Select the `offline_access` scope from the *Scopes* tab if you want to request the RT for the Client being created



Create a new client

Client name

Change me please!

Human readable client name

Client id

The ID will be generated when the client is saved.

Client description

Client description...

Human readable client description

Redirect URIs

https://app.example.org/cb

List of Redirect URIs for this client

☒ http://localhost/example

Contacts

administrator@example.org

List of email address contacts for administrators of this client

☒ federica.agostini89@gmail.com

Client registration in IAM through API

- IAM supports anonymous OAuth Client registration
    - a Client is **not linked to an account**
    - used for example by oidc-agent
- In IAM one can use the client-registration API

Example of client registration in IAM through API

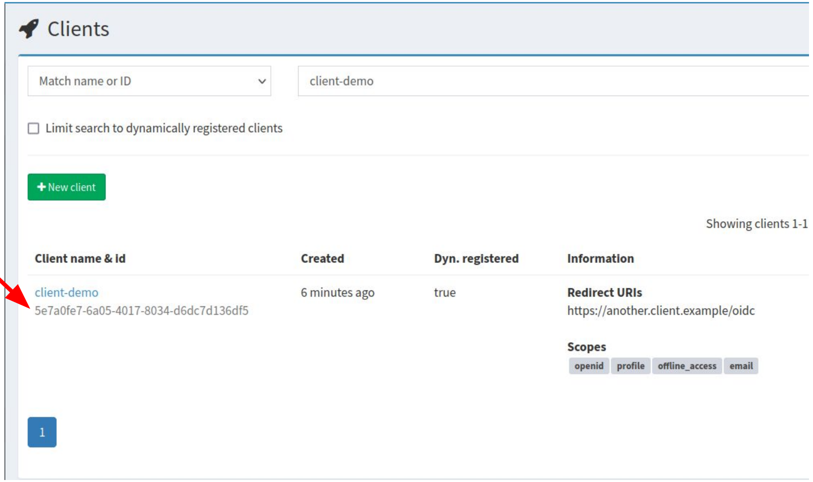
Prepare a JSON file with the Client details, for instance

```
$ cat client_req.json
{
  "redirect_uris": [
      "https://another.client.example/oidc"
  ],
  "client_name": "client-demo",
  "contacts": [
      "test@iam.test"
  ],
  "token_endpoint_auth_method": "client_secret_basic",
  "scope": "address phone openid email profile offline_access",
  "grant_types": [
      "refresh_token",
      "authorization_code"
  ],
  "response_types": [
      "code"
  ]
}
```

Example of client registration in IAM through API

POST HTTP request to the IAM client-registration API

```
$ curl https://wlcg.cloud.cnaf.infn.it/iam/api/client-registration -H "Content-Type: application/json" -d @client_req.json 2>/dev/null |
jq
{
  "client_id": "90b4f677-2551-4852-935e-8f785c583572",
  "client_secret": "xxx",
  "client_name": "client-demo",
  "redirect_uris": [
        "https://another.client.example/oidc"
  ],
  "contacts": [
        "test@iam.test"
  ],
  "grant_types": [
        "authorization_code",
        "refresh_token"
  ],
  "response_types": [
        "code"
  ],
  "token_endpoint_auth_method": "client_secret_basic",
  "scope": "openid profile offline_access email",
  "reuse_refresh_token": true,
  "dynamically_registered": true,
  "clear_access_tokens_on_refresh": true,
  "require_auth_time": false,
  "registration_access_token": "xxx”,
  "registration_client_uri": "https://wlcg.cloud.cnaf.infn.it/iam/api/client-registration/90b4f677-2551-4852-935e-8f785c583572",
  "created_at": 1669116921824
}
```



□ Limit search to dynamically registered clients

Showing clients 1-1

Client name & id	Created	Dyn. registered	Information
client-demo 5e7a0fe7-6a05-4017-8034-d6dc7d136df5	6 minutes ago	true	**Redirect URIs** https://another.client.example/oidc **Scopes** openid profile offline_access email

OAuth/OIDC grant types

Authorization grant types

=

Authorization Flows

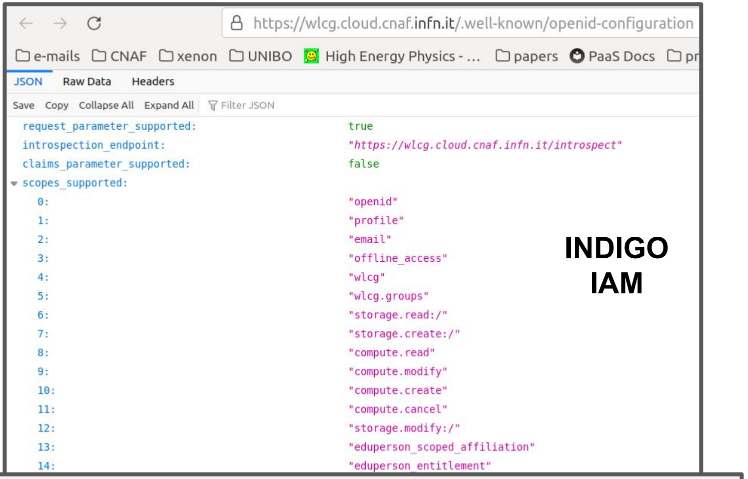
=

Ways for an application to get tokens

OAuth/OIDC provider metadata

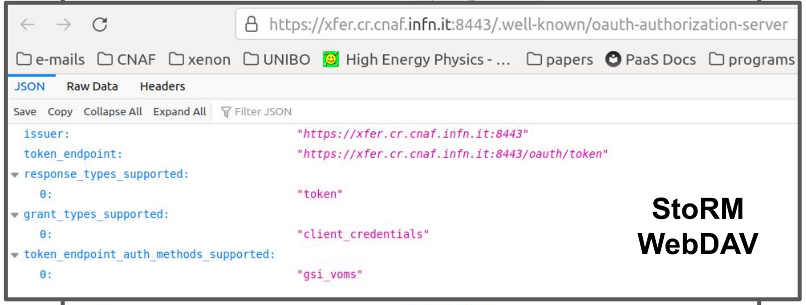
- OAuth & OIDC provide a standard way to expose the AS/OP configuration to Clients
- Information is published at **a well-known endpoint** for the server
  - `.well-known/openid-configuration` (in OIDC)
  - `.well-known/oauth-authorization-server` (in OAuth)
- Clients can use this information to know about
  - location of key material used to sign/encrypt tokens
  - supported grant types/authorization flows
  - endpoint locations
  - supported claims
  - …

  and implement **automatic Client configuration**
- Example metadata document:
  - https://wlcg.cloud.cnaf.infn.it/.well-known/openid-configuration
  - https://xfer.cr.cnaf.infn.it:8443/.well-known/openid-configuration
  - https://xfer.cr.cnaf.infn.it:8443/.well-known/oauth-authorization-server
  - https://accounts.google.com/.well-known/openid-configuration



```
request_parameter_supported:       true
introspection_endpoint:            "https://wlcg.cloud.cnaf.infn.it/introspect"
claims_parameter_supported:        false
scopes_supported:
  0:                               "openid"
  1:                               "profile"
  2:                               "email"
  3:                               "offline_access"
  4:                               "wlcg"
  5:                               "wlcg.groups"
  6:                               "storage.read:/"
  7:                               "storage.create:/"
  8:                               "compute.read"
  9:                               "compute.modify"
  10:                              "compute.create"
  11:                              "compute.cancel"
  12:                              "storage.modify:/"
  13:                              "eduperson_scoped_affiliation"
  14:                              "eduperson_entitlement"
```

INDIGO IAM



```
issuer:                                 "https://xfer.cr.cnaf.infn.it:8443"
token_endpoint:                         "https://xfer.cr.cnaf.infn.it:8443/oauth/token"
response_types_supported:
  0:                                    "token"
grant_types_supported:
  0:                                    "client_credentials"
token_endpoint_auth_methods_supported:
  0:                                    "gsi_voms"
```

StoRM WebDAV

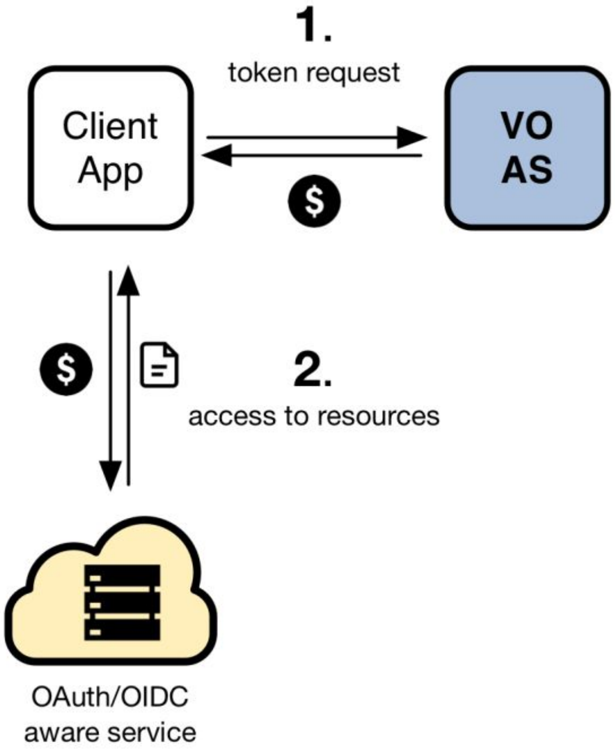
Token-based AuthN/Z

In order to access resources/services, a **Client application** needs an **Access Token**

The token is obtained from a **VO** (through an OAuth Authorization Server) using standard **OAuth/OIDC** flows

Authorization is then **performed at the services**, leveraging info extracted from the token:

- **Identity attributes**: *e.g.*, groups/roles
- **Scopes**: capabilities linked to access tokens at token creation time



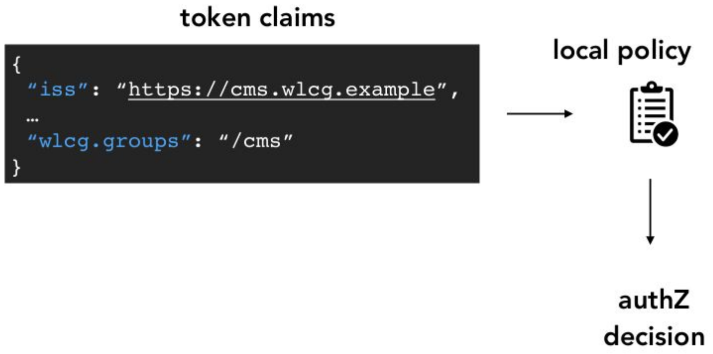
Identity-based vs Scope-based Authorization

Identity-based authorization

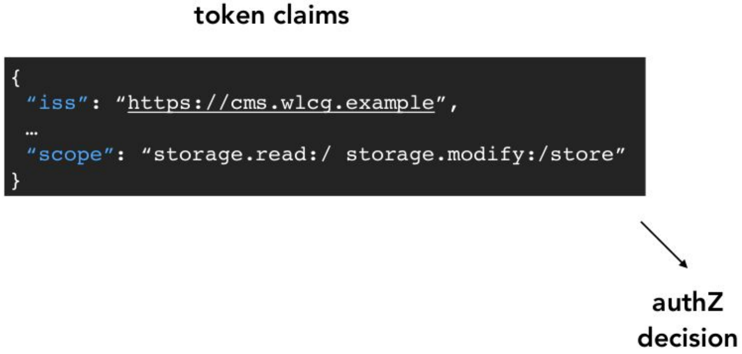
- the token brings information about attribute entitlement (*e.g.*, group/role membership)
- the service maps these attributes to a local authorization policy

Scope-based authorization

- the token brings information about which actions should be authorized at a service
- the service needs to understand these capabilities and honor them
- the authorization policy is managed at the VO level (*i.e.*, IAM)



```
{
  “iss”: “https://cms.wlcg.example”,
  …
  “wlcg.groups”: “/cms”
}
```



```
{
  “iss”: “https://cms.wlcg.example”,
  …
  “scope”: “storage.read:/ storage.modify:/store”
}
```

In practice …

- The central AS provides **attributes** that can be used for authorization at services, *e.g.*:
  - groups/roles, *e.g.*: `/escape`, `/cms/analysis`, `/atlas/production`
  - scopes, *e.g.*: `storage.read:/escape`, `compute.create`
- This information is exposed to services via **signed JWTs**, possibly obtained via **OAuth/OIDC** protocol message exchanges (*aka* flows)
- Services can then **grant or deny access** to functionality based on this information. Examples:
  - allow read access on the `/cms` namespace to all members of the `/cms` group
  - allow read access on the `/cms` namespace to anyone with the capability `storage.read:/cms`
- For instance, in StoRM WebDAV one can set authorization policies based on JWT subject/groups/issuer
  - capability-based access following the WLCG JWT profile requirements is natively supported

JSON Web Tokens (JWT)

OAuth bearer token usage

- RFC 6750
- It defines how to use tokens in HTTP requests to access protected resources on Resource Servers
- Any party in possession of a bearer token (a "bearer") can use it to **get access** to the associated resources (without demonstrating possession of a cryptographic key)
- OAuth bearer token must be used in combination with **TLS over HTTP**
- Typically, tokens are sent in the **Authorization HTTP header**, as in the following example HTTP request

```
GET / HTTP/1.1
Host: apache.test.example
Authorization: Bearer eyJraWQiOiJy…rYI
User-Agent: curl/7.65.3
Accept: */*
```

The token!

JSON Web Tokens: definition

- RFC 7519
- **JSON Web Token** is a compact, self-contained way of securely transmitting information between parties in a JSON object
- The payload of the JWT is encoded in token claims
- JWTs are typically **signed** and, if confidentiality is a requirement, can be **encrypted**
- A JWT is represented as a sequence of **URL-safe parts** separated by period (“.”) characters. Each part contains a **base64url-encoded value**.
- The number of parts in the JWT is dependent upon the representation of the resulting JSON Web Signature (JWS) using the JWS Compact Serialization or JSON Web Encryption (JWE) using the JWE Compact Serialization

JWT: Header.Body.Signature

Example of encoded token

eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJ3bGNnLnZlciI6IjEuMCIsInN1YiI6IjBmZDc2YjNjLWMzZjEtNDI4MC1iZTNjLTVlYmVhZDgxYzZkNiIsImF1ZCI6Imh0dHBzOlwvXC93bGNnLmNlcm4uY2hcL2p3dFwvdjFcL2FueSIsIm5iZiI6MTY2OTEyNzI3Nywic2NvcGUiOiJzdG9yYWdlLnJlYWQ6XC8iLCJpc3MiOiJodHRwczpcL1wvd2xjZy5jbG91ZC5jbmFmLmluZm4uaXRcLyIsImV4cCI6MTY2OTEzMDg3NywiaWF0IjoxNjY5MTI3Mjc3LCJqdGkiOiI5ZDE0NGRhMC1hMTQ5LTQwZTItYWM3NS01MjM0YzFjOTcyODIiLCJjbGllbnRfaWQiOiJlYjllMWNjMi1mNWUxLTRhNGItYjk2Ny1iY2NlYTI2NmYwOWIifQ.YbsCossZBloBxJBgk9D-IdVuAzm67rl_MVVdp8j4bXicLgPCM-6Wdze2VMzR6Nw0KMCBXhs59e5glgq0Fr5kagrpPjuua2sHX5ul84SNvlgoKMwSn_NIDXSO9flaDIIuelrSgT1qOTSiMV5M_U4VpWjOimpYm9fxmLSSIZT59MU

JWT: Header.Body.Signature

Example of decoded token

Header

```
$ echo $AT | cut -d. -f1 |
base64 -d 2>/dev/null | jq

{
  "kid": "rsa1",
  "alg": "RS256"
}
```

Payload

```
$ echo $AT | cut -d. -f2 | base64 -d
2>/dev/null | jq

{
  "wlcg.ver": "1.0",
  "sub":
"0fd76b3c-c3f1-4280-be3c-5ebead81c6d6",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1669127273,
  "scope": "storage.read:/",
  "iss": "https://wlcg.cloud.cnaf.infn.it/",
  "exp": 1669130873,
  "iat": 1669127273,
  "jti":
"2222be79-e218-442b-9389-c741c5b95da2",
  "client_id":
  "eb9e1cc2-f5e1-4a4b-b967-bccea266f09b"
}
```

Signature

```
$ echo $AT | cut -d. -f3

Zcamp7C40T4oygiO9_ua6oASnE
TYvkZhr8x_OredqLQagryptTwl
iDJRcCA2L8Uff_Tyh8KxKJsc1e
k86pGEZnkckFcfKscNJQyg8qKt
4plTDpxUkMV0ficF--IFOK3ACl
u18kWSG1pc85IGl8r64qF5e46o
fHjblGDnQAz06bc
```

JWT: Header.Body.Signature

Example of decoded token

Header

```
$ echo $AT | cut -d. -f1 |
base64 -d 2>/dev/null | jq

{
  "kid": "rsa1",
  "alg": "RS256"
}
```

Payload

```
$ echo $AT | cut -d. -f2 | base64 -d
2>/dev/null | jq
```

Useful JWT decoders

- https://jwt.io/
- https://github.com/troyharvey/jwt-cli
- *etc*

```
  "iat": 1669127273,
  "jti":
"2222be79-e218-442b-9389-c741c5b95da2",
  "client_id":
  "eb9e1cc2-f5e1-4a4b-b967-bccea266f09b"
}
```

Signature

```
$ echo $AT | cut -d. -f3

Zcamp7C40T4oygiO9_ua6oASnE
TYvkZhr8x_OredqLQagryptTwl
iDJRcCA2L8Uff_Tyh8KxKJsc1e
k86pGEZnkckFcfKscNJQyg8qKt
4plTDpxUkMV0ficF--IFOK3ACl
u18kWSG1pc85IGl8r64qF5e46o
fHjblGDnQAz06bc
```

JWT claim names

Typical registered claim names (*i.e.* a set of basic claims defined by the JWT standard)

- **“iss”** (Issuer): the principal (AS/OP) that issued the JWT (*e.g.*, IAM WLCG)
- **“sub”** (Subject): the principal that is the subject of the JWT (*e.g.*, a unique ID linked to an IAM account)
- **“aud”** (Audience): identifies the recipients that the JWT is intended for (*e.g.*, RUCIO)
- **“exp**” (Expiration time): identifies the expiration time after which the JWT MUST NOT be accepted by resources
- **“nbf”** (Not before): identifies the time before which the JWT MUST NOT be accepted by resources
- **“iat”** (Issued at): identifies the time at which the JWT was issued
- **“jti”** (JWT ID): provides a unique identifier for the JWT

Additional IAM claims

- **“client_id”**: ID of the client which requests the token
- **“scope”**: depends on the IAM profile in use (more explanation on the profiles in the *WLCG* slides)
    - `iam` => not included
    - `wlcg` => included
    - `aarc` => not included
- **“groups”**: list of groups the user is member of (available with `iam` profile)
- **“wlcg.groups”**: list of groups the user is member of, but has to be explicitly requested with the `wlcg.groups` scope (available with `wlcg` profile)
- **“eduperson_entitlement”**: list of groups the user is member of, but has to be explicitly requested with the `eduperson_entitlement`scope (available with `aarc` profile)
- *etc*

JWT validation

- Section 4 of RFC 9068
- Validating a JWT means:
  - **check** that the **current time** is before the time represented by the “exp” claim (delays of few minutes are allowed to account for clock skew)
  - check the **token signature** using the algorithm specified in the JWT "alg" Header Parameter
    - the well-known endpoint of the AS shares its public/symmetric key through the *jwks_uri* field
    - the JWKS can be cached (in IAM the *max-age* is configurable, default is 6 hours)
  - if validation is performed by the Resource Server, the **“aud”** claim must contain a resource indicator value corresponding to the **resource itself**
- OAuth 2 foresees that the AS implements an introspection endpoint which does the job (RFC 7662)
- Anyhow, a callback to the AS for the token validation can (and should) be avoided
  - many libraries support the token validation
  - we strongly advise to implement your own token validation
  - example on github

OAuth grant types

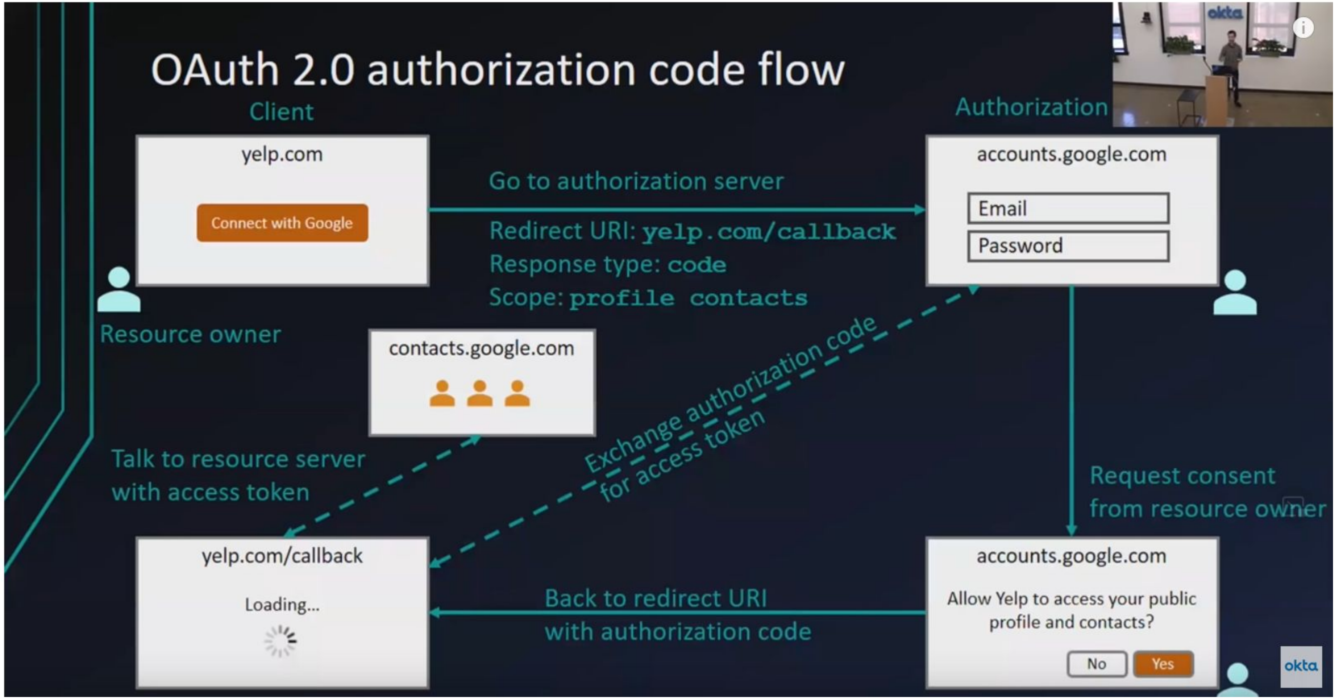
- authorization code flow
- device code flow
- refresh token flow
- client credentials
- token exchange

Authorization code flow

- Section 4.1 of RFC 6749 (OAuth 2)
- Section 3.1 of the OpenID Connect spec
- The recommended flow for server-side applications that can maintain the confidentiality of client credentials
    - but recommended for any client when combined with PKCE
- Allows an application to obtain tokens to act **on behalf of a user** for a potentially unbounded amount of time **within the limits of allowed scopes**

Authorization code flow

OAuth 2.0 and OpenID Connect video from OktaDev



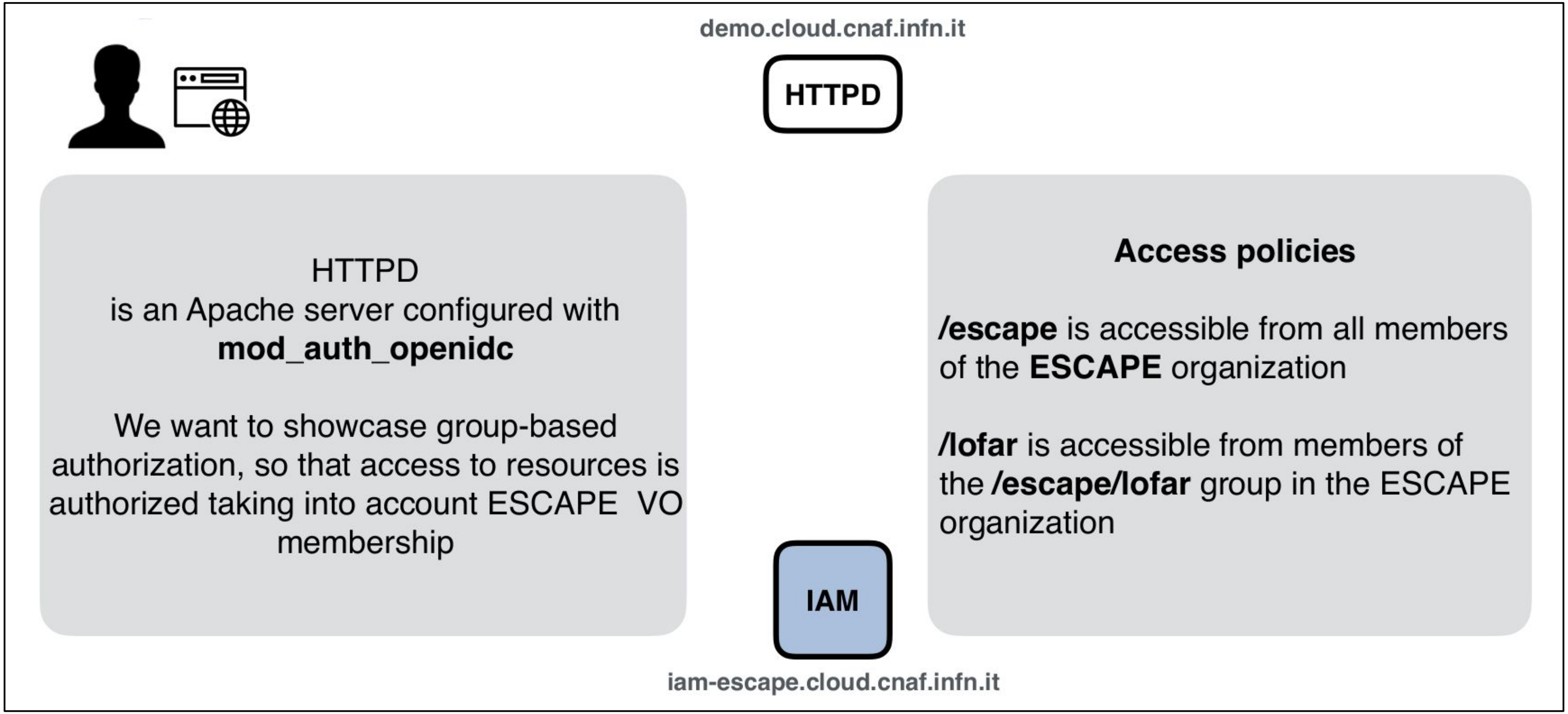
OAuth 2.0 authorization code flow
Client
Authorization
yelp.com
Connect with Google
Go to authorization server
Redirect URI: yelp.com/callback
Response type: code
Scope: profile contacts
accounts.google.com
Email
Password
Resource owner
contacts.google.com
Exchange authorization code for access token
Talk to resource server with access token
Request consent from resource owner
yelp.com/callback
Loading...
Back to redirect URI with authorization code
accounts.google.com
Allow Yelp to access your public profile and contacts?
No
Yes
okta

Authorization code flow in practice

- In practice, many OAuth/OIDC client libraries implement all the above **behind the scenes**
- As an example, Apache and nginx modules require the following information to enable a working OIDC integration
  - The OIDC provider discovery/metadata URL
  - Client credentials
- The library then takes care of exchanging messages with the OP, implementing verification checks, and provides the obtained authentication/authorization information to the protected web application
  - typically via env variables or HTTP headers
- Source code of an Apache integration demo by Andrea here

Integration Demo setup

From ESCAPE AAI Webinar



HTTPD
is an Apache server configured with
mod_auth_openidc

We want to showcase group-based authorization, so that access to resources is authorized taking into account ESCAPE VO membership

Access policies

/escape is accessible from all members of the **ESCAPE** organization

/lofar is accessible from members of the **/escape/lofar** group in the ESCAPE organization

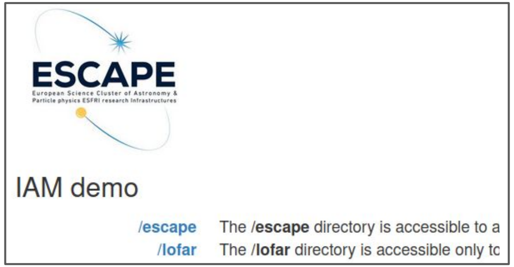
Apache mod_auth_openidc configuration

From ESCAPE AAI Webinar

```
ServerName demo.cloud.cnaf.infn.it

<VirtualHost _default_:80>

  OIDCProviderMetadataURL https://iam-escape.cloud.cnaf.infn.it/.well-known/openid-configuration
  OIDCClientID demo_client
  OIDCClientSecret ***
  OIDCScope "openid email profile"
  OIDCRedirectURI https://demo.cloud.cnaf.infn.it/oidc/redirect_uri
  OIDCCryptoPassphrase *****

  <Location /escape>
    …
    AuthType openid-connect
    Require valid-user
  </Location>

  …

  <Location /lofar>
    …
    AuthType openid-connect
    Require claim groups:/escape/lofar
  </Location>

</VirtualHost>
```

Demo application in action

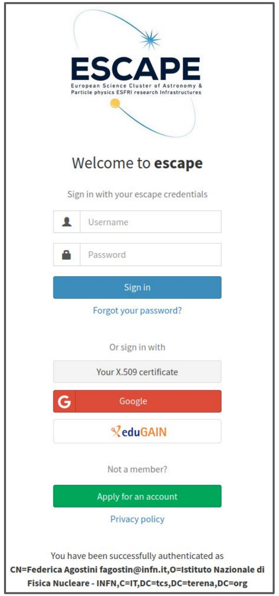
https://demo.cloud.cnaf.infn.it



IAM demo

/escape The /**escape** directory is accessible to a

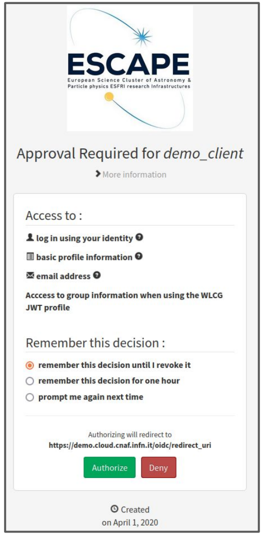
/lofar The /**lofar** directory is accessible only tc



Welcome to **escape**

Sign in with your escape credentials

You have been successfully authenticated as **CN=Federica Agostini fagostin@infn.it,O=Istituto Nazionale di Fisica Nucleare - INFN,C=IT,DC=tcs,DC=terena,DC=org**



Approval Required for *demo_client*

› More information

Access to :

log in using your identity

basic profile information

email address

Acccess to group information when using the WLCG JWT profile

Remember this decision :

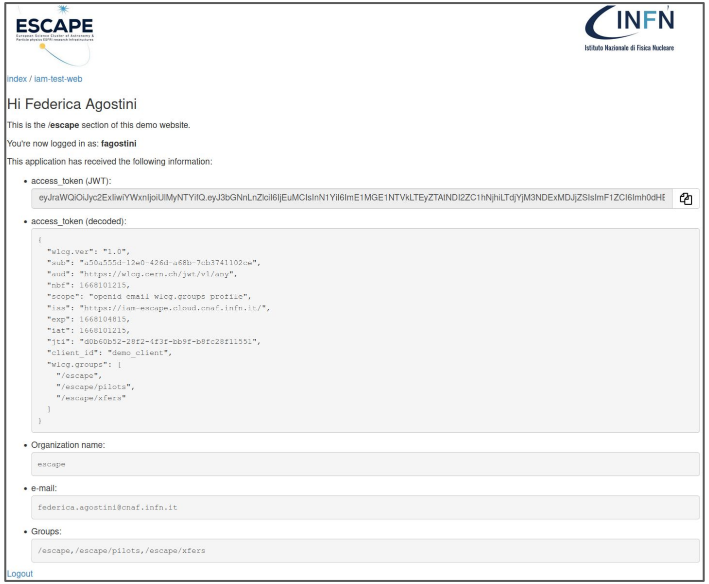
● **remember this decision until I revoke it**

○ **remember this decision for one hour**

○ **prompt me again next time**

Authorizing will redirect to **https://demo.cloud.cnaf.infn.it/oidc/redirect_uri**

Created on April 1, 2020



index / iam-test-web

Hi Federica Agostini

This is the /**escape** section of this demo website.

You're now logged in as: **fagostini**

This application has received the following information:

- access_token (JWT):

```
eyJraWQiOiJyc2ExliwiYWxnIjoiUlMyNTYifQ.eyJ3bGNnLnZlciI6IjEuMCIsInN1YiI6ImE1MGE1NTVkLTEyZTAtNDI2ZC1hNjhiLTdjYjM3NDExMDJjZSIsImF1ZCI6Imh0dHE
```

- access_token (decoded):

```
{
  "wlcg.ver": "1.0",
  "sub": "a50a555d-12e0-426d-a68b-7cb3741102ce",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1668101215,
  "scope": "openid email wlcg.groups profile",
  "iss": "https://iam-escape.cloud.cnaf.infn.it/",
  "exp": 1668104815,
  "iat": 1668101215,
  "jti": "d0b60b52-28f2-4f3f-bb9f-b8fc28f11551",
  "client_id": "demo_client",
  "wlcg.groups": [
    "/escape",
    "/escape/pilots",
    "/escape/xfers"
  ]
}
```

- Organization name:

```
escape
```

- e-mail:

```
federica.agostini@cnaf.infn.it
```

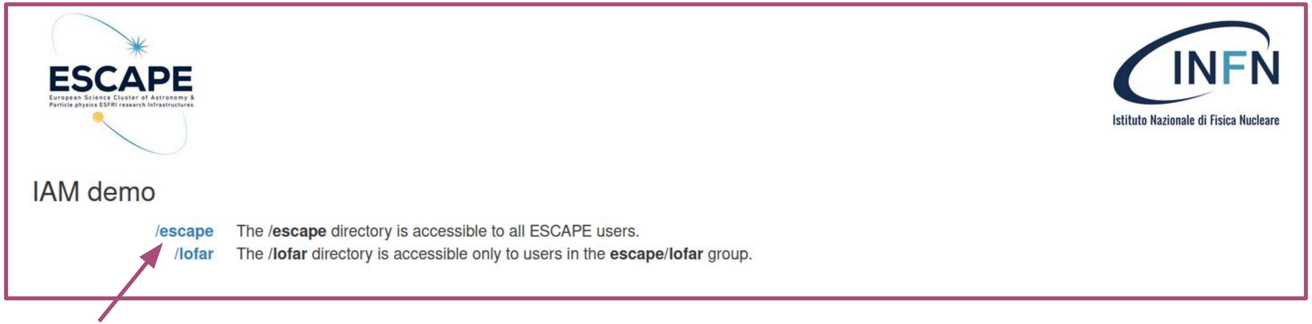
- Groups:

```
/escape,/escape/pilots,/escape/xfers
```

Logout

What happens behind the scenes

- `GET https://demo.cloud.cnaf.infn.it`



ESCAPE
European Science Cluster of Astronomy & Particle physics ESFRI research Infrastructures
INFN
Istituto Nazionale di Fisica Nucleare

IAM demo

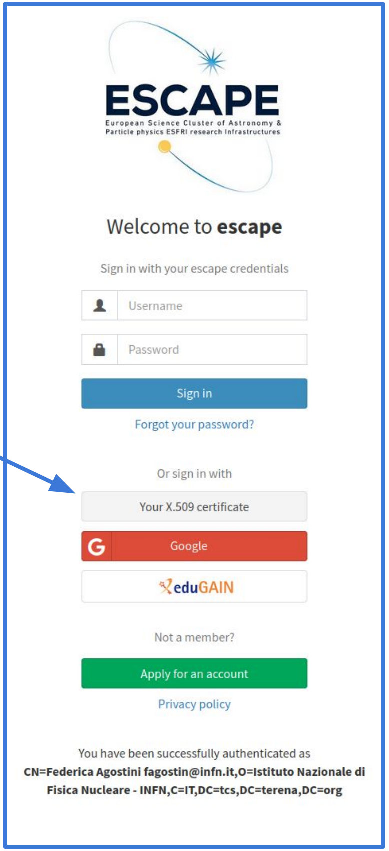
/escape The /**escape** directory is accessible to all ESCAPE users.

/lofar The /**lofar** directory is accessible only to users in the **escape/lofar** group.

- `GET https://demo.cloud.cnaf.infn.it/escape`
- `302`, redirect to `https://iam-escape.cloud.cnaf.infn.it/authorize?response_type=code&scope=openid%20profile%20email%20wlcg.groups&client_id=demo_client&state=UbSXWU5MyWvFvaWnoQwOkVwUM_M&redirect_uri=https%3A%2F%2Fdemo.cloud.cnaf.infn.it%2Foidc%2Fredirect_uri&nonce=q5dHehbHMSM1b7CGm6lWR8m26RynGPgwpmimr7rpesY`
- `GET https://iam-escape.cloud.cnaf.infn.it/authorize?response_type=code&scope=openid%20profile%20email%20wlcg.groups&client_id=demo_client&state=UbSXWU5MyWvFvaWnoQwOkVwUM_M&redirect_uri=https%3A%2F%2Fdemo.cloud.cnaf.infn.it%2Foidc%2Fredirect_uri&nonce=q5dHehbHMSM1b7CGm6lWR8m26RynGPgwpmimr7rpesY`
- `302`, redirect to `https://iam-escape.cloud.cnaf.infn.it/login`

What happens behind the scenes

- `GET https://iam-escape.cloud.cnaf.infn.it/login`
- `200` => user has to authenticate (I am using x509 in this example)
- `GET https://iam-escape.cloud.cnaf.infn.it/dashboard?x509ClientAuth=true`
- `302`, redirect to `https://iam-escape.cloud.cnaf.infn.it/authorize?response_type=code&scope=openid%20profile%20email%20wlcg.groups&client_id=demo_client&state=UbSXWU5MyWvFvaWnoQwOkVwUM_M&redirect_uri=https%3A%2F%2Fdemo.cloud.cnaf.infn.it%2Foidc%2Fredirect_uri&nonce=q5dHehbHMSM1b7CGm6lWR8m26RynGPgwpmimr7rpesY`


ESCAPE
European Science Cluster of Astronomy & Particle physics ESFRI research Infrastructures

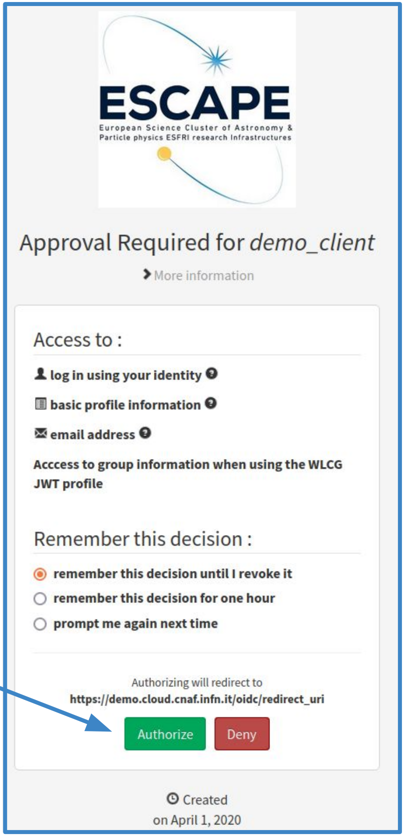
Welcome to **escape**

Sign in with your escape credentials

You have been successfully authenticated as
CN=Federica Agostini fagostin@infn.it,O=Istituto Nazionale di Fisica Nucleare - INFN,C=IT,DC=tcs,DC=terena,DC=org

What happens behind the scenes

- `GET https://iam-escape.cloud.cnaf.infn.it/authorize?response_type=code&scope=openid%20profile%20email%20wlcg.groups&client_id=demo_client&state=UbSXWU5MyWvFvaWnoQwOkVwUM_M&redirect_uri=https%3A%2F%2Fdemo.cloud.cnaf.infn.it%2Foidc%2Fredirect_uri&nonce=q5dHehbHMSM1b7CGm6lWR8m26RynGPgwpmimr7rpesY`
- `200` => user has to authorize the Client app to access their data
- `POST https://iam-escape.cloud.cnaf.infn.it/authorize`
  - IAM generates an **authorization code** and sends it back to the web app using an HTTP redirect
- `303`, redirect to `https://demo.cloud.cnaf.infn.it/oidc/redirect_uri?code=jjBikzc_C_vWe9_ho8iqUO&state=UbSXWU5MyWvFvaWnoQwOkVwUM_M`



ESCAPE

European Science Cluster of Astronomy & Particle physics ESFRI research Infrastructures

Approval Required for *demo_client*

› More information

Access to :

log in using your identity

basic profile information

email address

Acccess to group information when using the WLCG JWT profile

Remember this decision :

● remember this decision until I revoke it

○ remember this decision for one hour

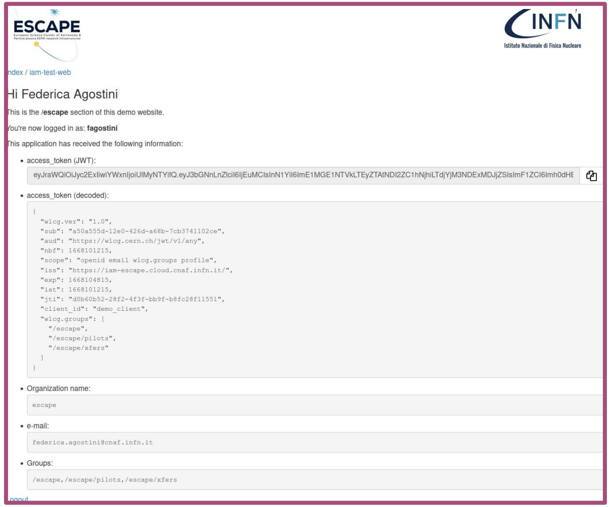
○ prompt me again next time

Authorizing will redirect to
https://demo.cloud.cnaf.infn.it/oidc/redirect_uri

Created
on April 1, 2020

What happens behind the scenes

- `GET https://demo.cloud.cnaf.infn.it/oidc/redirect_uri?code=jjBikzc_C_vWe9_ho8iqUO&state=UbSXWU5MyWvFvaWnoQwOkVwUM_M`
  - The demo app exchanges the authorization code with an **access** and **id token** in the back channel
- `302`, redirect to `https://demo.cloud.cnaf.infn.it/escape`
- `200` => users has access to their resource



ndex / iam-test-web

Hi Federica Agostini

his is the /**escape** section of this demo website.

ou're now logged in as: **fagostini**

his application has received the following information:

- access_token (JWT):

```
eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJ3bGNnLnZlciI6IjEuMCIsInN1YiI6ImE1MGE1NTVkLTEyZTAtNDI2ZC1hNjhiLTdjYjM3NDExMDJjZSIsImF1ZCI6Imh0dHE
```

- access_token (decoded):

```
{
  "wlcg.ver": "1.0",
  "sub": "a50a555d-12e0-426d-a68b-7cb3741102ce",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1668101215,
  "scope": "openid email wlcg.groups profile",
  "iss": "https://iam-escape.cloud.cnaf.infn.it/",
  "exp": 1668104815,
  "iat": 1668101215,
  "jti": "d0b60b52-28f2-4f3f-bb9f-b8fc28f11551",
  "client_id": "demo_client",
  "wlcg.groups": [
    "/escape",
    "/escape/pilots",
    "/escape/xfers"
  ]
}
```

- Organization name:

```
escape
```

- e-mail:

```
federica.agostini@cnaf.infn.it
```

- Groups:

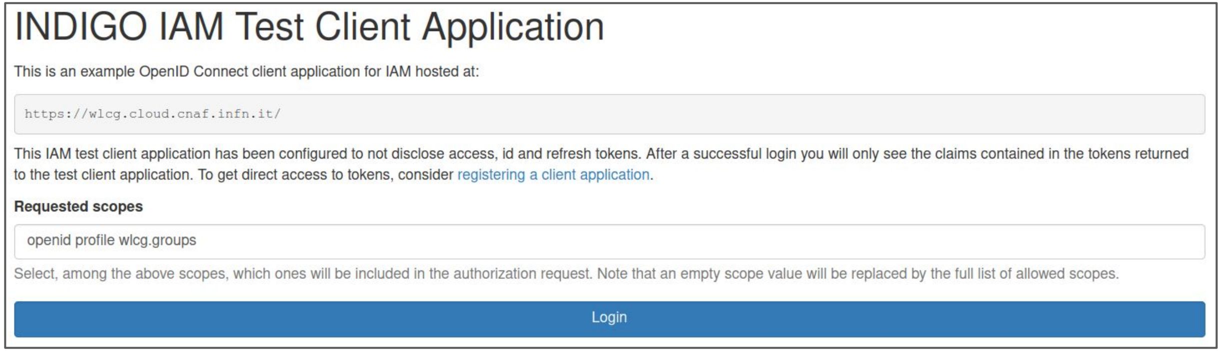
```
/escape,/escape/pilots,/escape/xfers
```

ogout

Clone your demo app

- A web server is necessary to reproduce the authorization code flow (*i.e.* not feasible with `curl`)
- Many examples can be found on github, *e.g.*
  - iam-test-client
  - sample-oauth2-client
  - invenio-oauth-client
  - flask-dance
  - *etc*
- Clone it and enjoy !

https://wlcg.cloud.cnaf.infn.it/iam-test-client/



INDIGO IAM Test Client Application

This is an example OpenID Connect client application for IAM hosted at:

```
https://wlcg.cloud.cnaf.infn.it/
```

This IAM test client application has been configured to not disclose access, id and refresh tokens. After a successful login you will only see the claims contained in the tokens returned to the test client application. To get direct access to tokens, consider registering a client application.

Requested scopes

openid profile wlcg.groups

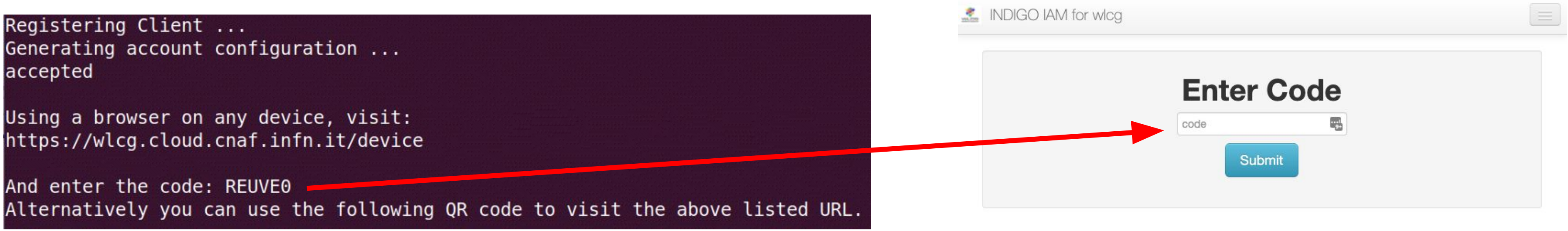
Select, among the above scopes, which ones will be included in the authorization request. Note that an empty scope value will be replaced by the full list of allowed scopes.

Device code flow

- RFC 8628
- Used in place of the authorization code flow when the Client can not easily trigger a browser-based authorization
  - the authorization to access protected resources happens on a separate device
- Requirements for the device code flow:
  - the device is able to display or otherwise communicate an URI and code sequence to the user
  - the user has a secondary device (*e.g.*, personal computer or smartphone) from which they can process the request
- Since the protocol supports Clients that can not receive incoming requests, the Clients poll the authorization server repeatedly until the end user completes the approval process

Device code flow

- The authorization grant is a **code**, *e.g.* a sequence of 6 letters/numbers in IAM
  - The code has to be requested at the device code endpoint exposed by the AS
  - The device code endpoint can be retrieved from the *well-known* endpoint
  - The code is used to obtain an AT
- Supported by `oidc-agent`



```
Registering Client ...
Generating account configuration ...
accepted

Using a browser on any device, visit:
https://wlcg.cloud.cnaf.infn.it/device

And enter the code: REUVE0
Alternatively you can use the following QR code to visit the above listed URL.
```

Example of a device code request

- Client credentials are needed to get a device code
- The *audience* request parameter can be used to suggest an audience for the requested access token
- The device code endpoint in IAM it is `/devicecode`

```
$ curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d client_id=${CLIENT_ID} -d
scope="${CLIENT_SCOPES}" ${DEVICECODE_ENDPOINT} > response.json
$ device_code=$(jq -r .device_code response.json)
$ user_code=$(jq -r .user_code response.json)
$ verification_uri=$(jq -r .verification_uri response.json)
$ verification_uri_complete=$(jq -r .verification_uri_complete response.json)
```

Useful script

Example of a device code request

- After authentication, copy in the browser the obtained `user_code`
  - URI of the code verification is `${verification_uri}`
  - use `${verification_uri_complete}` instead, if you do not want to copy-and-paste the code
- Then, you will be prompted to the consent page
- The `device_code` is finally used in the POST request to the token endpoint
  - it can be retrieved from the *well-known* endpoint. In IAM it is `/token`

```
$ curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d
grant_type=urn:ietf:params:oauth:grant-type:device_code -d audience=${AUDIENCE} -d
device_code=${device_code} ${TOKEN_ENDPOINT}
```

Useful script

Real device code request

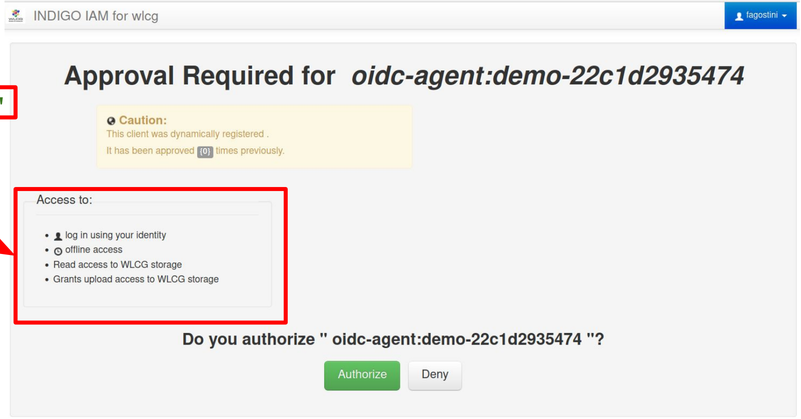
- Register a Client with the device code grant type enabled
    - I have used `oidc-agent` (next slides), selecting WLCG IAM as token issuer/AS
    - my `client_id` is `eb9e1cc2-f5e1-4a4b-b967-bccea266f09b`
    - among the scopes allowed for my client, there is `openid offline_access storage.read:/ storage.create:/`
- POST request to the device code endpoint

```
$ curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d client_id=${CLIENT_ID} -d scope="openid
offline_access storage.read:/ storage.create:/" https://wlcg.cloud.cnaf.infn.it/devicecode >
code_response.json
$ device_code=$(jq -r .device_code code_response.json)
$ user_code=$(jq -r .user_code code_response.json)
$ verification_uri=$(jq -r .verification_uri code_response.json)
$ echo ${user_code}
EMGFZA
$ echo ${verification_uri}
https://wlcg.cloud.cnaf.infn.it/device
```

Real device code request

- I have copied the code **EMGFZA** in the box which appears at https://wlcg.cloud.cnaf.infn.it/device
- Then, I gave permissions to the Client to access my data
- And now I can ask for tokens

```
$ curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=urn:ietf:params:oauth:grant-type:device_code -d
audience=myAudience -d device_code=${device_code} https://wlcg.cloud.cnaf.infn.it/token > token_response.json
$ jq -r .access_token token_response.json | tr -d '"' | cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "0fd76b3c-c3f1-4280-be3c-5ebead81c6d6",
  "aud": "myAudience",
  "nbf": 1669306655,
  "scope": "storage.read:/ storage.create:/ openid offline access",
  "iss": "https://wlcg.cloud.cnaf.infn.it/",
  "exp": 1669310255,
  "iat": 1669306655,
  "jti": "f05aaef8-efc4-46f4-be94-bf440c318f42",
  "client_id": "eb9e1cc2-f5e1-4a4b-b967-bccea266f09b"
}
```



Approval Required for *oidc-agent:demo-22c1d2935474*

Caution:
This client was dynamically registered .
It has been approved {0} times previously.

Access to:

- log in using your identity
- offline access
- Read access to WLCG storage
- Grants upload access to WLCG storage

Do you authorize " oidc-agent:demo-22c1d2935474 "?

Real device code request

- A **refresh token** has been issued to the Client (together with the AT) due to the `offline_access` scope requested during the device code flow
- An **id token** has been issued due to the `openid` scope requested during the device code flow

```
$ jq -r . token_response.json
{
  "access_token":
"eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJ3bGNnLnZlciI6IjEuMCIsInN1YiI6IjBmZDc2YjNjLWMzZjEtNDI4MC1iZTNjLTVlYmVhZDgxYzZkNiIsImF1
ZCI6Im15QXVkaWVuY2UiLCJuYmYiOjE2NjkzMDY2NTUsInNjb3BlIjoic3RvcmFnZS5yZWFkOlwvIHN0b3JhZ2UuY3JlYXRlOlwvIG9wZW5pZCBvZmZsaW5lX2FjY2Vz
cyIsImlzcyI6Imh0dHBzOlwvXC93bGNnLmNsb3VkLmNuYWYuaW5mbi5pdFwvIiwiZXhwIjoxNjY5MzEwMjU1LCJpYXQiOjE2NjkzMDY2NTUsImp0aSI6ImYwNWFhZWY4
LWVmYzQtNDZmNC1iZTk0LWJmNDQwYzMxOGY0MiIsImNsaWVudF9pZCI6ImViOWUxY2MyLWY1ZTEtNGE0Yi1iOTY3LWJjY2VhMjY2ZjA5YiJ9.B_gsWbro3GF9ZqClABe
tpZIn2p61OIGTbO9n18PjP5UiodqhrdEubv9EKj5kWZZfSFhlzszlvmBziT9IZIelnx5CocYXkqzRalK0IJq-c4rzWB_o-9QgJR84FdgxN5sY6OdMpxcp9N75gweuSJf
F0_ZZ9bLIgLWHzBnv4nTsKaw",
  "token_type": "Bearer",
  "refresh_token": "eyJhbGciOiJub25lIn0.eyJqd…",
  "expires_in": 3599,
  "scope": "storage.read:/ storage.create:/ openid offline_access",
  "id_token":
"eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJ3bGNnLnZlciI6IjEuMCIsInN1YiI6IjBmZDc2YjNjLWMzZjEtNDI4MC1iZTNjLTVlYmVhZDgxYzZkNiIsImF1
ZCI6ImViOWUxY2MyLWY1ZTEtNGE0Yi1iOTY3LWJjY2VhMjY2ZjA5YiIsImtpZCI6InJzYTEiLCJpc3MiOiJodHRwczpcL1wvd2xjZy5jbG91ZC5jbmFmLmluZm4uaXRc
LyIsImV4cCI6MTY2OTMwNzI1NSwiaWF0IjoxNjY5MzA2NjU1LCJqdGkiOiIxZjc3Y2MxYy01NmU2LTRmODQtODgzZC00NjYzNzI5MWY5ZDgifQ.LmyxnazIlAHzo16pZ
AgwLc-P7qjazgMtPMn_5xqcE5HJa2jf0H-QrnDPwQ0NSfmEEEuu6r48l2d6CeEBZOZf2SfoXZt6mXXR4wLX0lTPfj66Qj0lefd8r64bc_rONiw2y1qUesMUPHLxHqOwG
0cUyQKCAo9_KE6MvLSE57LezJU"
}
```

Device code flow exercise

- Reproduce the device code flow using the `verification_uri_complete` value in place of the `verification_uri`
- Use the AT obtained by IAM to access a resource server (*e.g.*, WebDAV)
  - in this case you can use groups, included in the token, requesting the `wlcg.groups` scope
  - hint: the *audience* claim must be the resource itself

Refresh token flow

- Section 1.5 of RFC 6749
- Mechanism to implement the ability for an application to act on behalf of a user and get tokens without user's interaction
- Used by a Client to refresh an AT that is about to expire
  - the authorization grant is a **refresh token**, which is obtained in a former authorization flow
- Authenticated POST request to the AS token endpoint
  - **Client credentials** and a valid **RT** must be provided by the caller
  - Produces a new AT and possibly an updated RT

Refresh token flow

- The scope request parameter should be used to attenuate the token privileges, by requesting a subset of the scopes linked to the first user authorization grant
- A refresh token request can be performed in order to **change the *audience*** claim in place of the *token exchange* flow (shown in next slides)
- RTs are Client specific, *i.e.*, a refresh token issued to Client A cannot be used by Client B
  - instead, this use case is supported by the *token exchange* flow

Refresh tokens in IAM

- In IAM, the refresh token flow can be enabled or disabled per Client
- RTs may have an expiration date, or be unbounded in validity. This depends on the Client configuration
    - tokens validity settings in IAM can only be changed by administrators
- In IAM, the default RT lifetime is infinite (*i.e.*, the RT does not expire)
    - within WLCG it has been asked to set the default lifetime of RT to 30 days
    - the WLCG IAM instance already supports this default
- RTs can be revoked/invalidated using a standard OAuth API

Refresh tokens: use cases

- *How long do we want a user “session” to last?*
  That’s the lifetime of the refresh token

- Example: users on a CLI should not be prompted for login more than once a week
  RT lifetime: **a week**

- Vault requests one week of validity for the RT lifetime

Example of a refresh token request

- Client credentials and a valid refresh token are needed to get a new access token
- The *audience* request parameter can be used to suggest an audience for the requested access token
- The token endpoint can be retrieved from the *well-known* endpoint
  - in IAM it is `/token`

```
$ curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=refresh_token -d
scope="${CLIENT_SCOPES}" -d audience=${AUDIENCE} -d
refresh_token=${REFRESH_TOKEN} ${TOKEN_ENDPOINT}
```

Useful script

Client registration for refresh token request

Register a Client with the `refresh_token` grant type enabled

- I have used `oidc-agent` (next slides), selecting WLCG IAM as token issuer/AS
- my `client_id` is `eb9e1cc2-f5e1-4a4b-b967-bccea266f09b`
- copy the RT stored in the local configuration for my oidc-agent client
  - it is visible with the command `oidc-add -p <client-alias>`
- the list of scopes allowed for my client are: `openid`, `offline_access`, `storage.read:/`, `storage.create:/`, `compute.read`, `compute.modify`

Real refresh token request

Authenticated POST request to the token endpoint

- I am not prompted to the consent page at this stage, as the user is considered to be offline
- the consent to access their data was given in the previous OAuth flow
  - the one which issued the RT
  - in case of my oidc-agent Client, it was the device code flow

```
$ curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=refresh_token -d audience=myAudience -d
refresh_token=${REFRESH_TOKEN} https://wlcg.cloud.cnaf.infn.it/token | jq -r .access_token |  tr -d '"' |
cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "0fd76b3c-c3f1-4280-be3c-5ebead81c6d6",
  "aud": "myAudience",
  "nbf": 1669390346,
  "scope": "storage.create:/ openid offline_access compute.read storage.read:/ compute.modify",
  "iss": "https://wlcg.cloud.cnaf.infn.it/",
  "exp": 1669393946,
  "iat": 1669390346,
  "jti": "d25af1fe-cc3a-4112-af80-2589c2f1b6af",
  "client_id": "eb9e1cc2-f5e1-4a4b-b967-bccea266f09b"
}
```

If I do not specify the list of requested scopes, all the ones allowed for my client are returned in the AT

Refresh token flow exercise

- Obtain a RT using the device code flow
  - hint: include `offline_access` among the requested scopes
- Obtain an AT using the refresh token flow and the RT issued in the previous bullet point
- Use the AT to access a resource server (*e.g.*, WebDAV)

Client credential flow

- Section 4.4 of RFC 6749
- A Client enabling the client credential flow is able to request tokens for itself
    - The client can request an AT using only its client credentials
- Client authentication is required (*i.e.* not enabled for public clients)
- The client authentication is used as authorization grant
    - No additional authorization request is needed
    - The authorization grant does not require intervention of a user (*i.e.* no login requested)
    - The consent page is not shown → a user does not have to authorize the Client app to access their data

Client credentials: use cases

- Used to obtain tokens not linked to user identities → they are linked to the service itself
- The AT issued with the client credentials request contains **scope-based attributes** only, *i.e.*
  - the *group* claim is not present
  - the *sub* claim is the `client_id` (NOT the user's uuid)
  - authorization is based on the scopes which are present in the AT → it has to be honoured by Resource Servers without introducing further authorization policies
- An automated script can ask for a new AT when the previous one is about to expire
  - the client credentials needed for this flow have to be maintained confidentials, stored on a secure server with restricted access
- Useful for service accounts, when the authorization is based on capabilities, *e.g.*
  - Rucio
  - FTS
  - *etc*

Example of a client credentials request

- Client credentials are the only required parameter to get an access token
- The *audience* request parameter can be used to suggest an audience for the requested access token
- The token endpoint can be retrieved from the *well-known* endpoint
    - in IAM it is `/token`

```
$ curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d
grant_type=client_credentials -d scope="${CLIENT_SCOPES}" -d
audience="${AUDIENCE}" ${TOKEN_ENDPOINT}
```

Useful script

Client registration for client credentials request

Register a Client with the `client_credentials` grant type enabled

- just for simplicity, I have reused the `oidc-agent` Client
- my `client_id` is `eb9e1cc2-f5e1-4a4b-b967-bccea266f09b`
- enabled the *client_credentials* grant below the *Grant types* tab
- I will request scopes which do not provide user informations → they are meaningless in this flow and would not introduce further claims in the token
  - *i.e.*: `storage.read:/`, `storage.create:/`, `compute.read`, `compute.modify`

Real client credentials request

Authenticated POST request to the token endpoint

- I am not prompted to the consent page at this stage, since the client authentication is used as authorization grant

```
$ curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=client_credentials -d
scope="compute.read compute.modify" -d audience=myAudience
https://wlcg.cloud.cnaf.infn.it/token | jq -r .access_token |  tr -d '"' | cut -d. -f2 |
base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "eb9e1cc2-f5e1-4a4b-b967-bccea266f09b",
  "aud": "myAudience",
  "nbf": 1669829197,
  "scope": "compute.modify compute.read",
  "iss": "https://wlcg.cloud.cnaf.infn.it/",
  "exp": 1669832797,
  "iat": 1669829197,
  "jti": "57a2e594-f212-4183-b6fa-e9b6b9402393",
  "client_id": "eb9e1cc2-f5e1-4a4b-b967-bccea266f09b"
}
```

The sub claim of the AT is the `client_id` (NOT the user's uuid)

Real client credentials request

According with RFC 6749, a RT should not be issued in a client credential request

- when the `offline_access` scope is requested, it appears in the AT, but no RT is issued

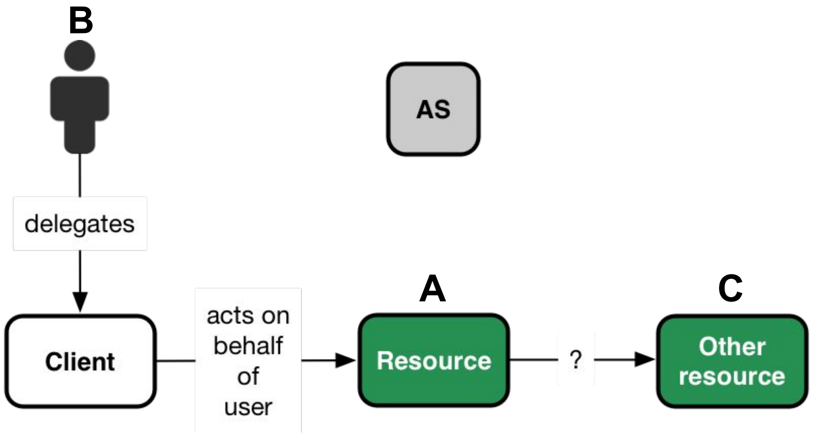
```
$ curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=client_credentials -d
scope="compute.read compute.modify offline_access" -d audience=myAudience
https://wlcg.cloud.cnaf.infn.it/token | jq
{
  "access_token":
"eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJ3bGNnLnZlciI6IjEuMCIsInN1YiI6ImViOWUxY2MyLWY1ZTEtNGE0Yi
1iOTY3LWJjY2VhMjY2ZjA5YiIsImF1ZCI6Im15QXVkaWVuY2UiLCJuYmYiOjE2Njk4OTA3NzIsInNjb3BlIjoiY29tcHV0ZS5t
b2RpZnkgY29tcHV0ZS5yZWFkIG9mZmxpbmVfYWNjZXNzIiwiaXNzIjoiaHR0cHM6XC9cL3dsY2cuY2xvdWQuY25hZi5pbmZuLm
l0XC8iLCJleHAiOjE2Njk4OTQzNzIsImlhdCI6MTY2OTg5MDc3MiwianRpIjoiZjg1Y2ZkOWQtOGRjNS00YjAwLThiYmYtODQ5
ODQwNjA4NWUzIiwiY2xpZW50X2lkIjoiZWI5ZTFjYzItZjVlMS00YTRiLWI5NjctYmNjZWEyNjZmMDliIn0.bnMwu4juY2npv9
cTlRtxXHSf9pJFUzpDAX9JUbSymdPxGMP-hE36z2lB0i4Y1c1yCnpz5Nou7j_0IL-0Wg89LXDvFCY5o4aX5rOb2lff0Hvwl0u2
pOMspyFCDSPO_in_fttC1mHwNSJbEyrtSa8PDBFM9Lew3LHUpFC0Z1P1HDM",
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": "compute.read compute.modify offline_access"
}
```

Client credential flow exercise

- Register a Client in the WLCG IAM with the *client_credentials* grant enabled and include the `wlcg.groups` among the allowed scopes
- Ask for an AT using the client credential flow and requests the `wlcg.groups` scope
  - check if the *wlcg.groups* claim appears in the AT
- Use this AT to access to https://xfer.cr.cnaf.infn.it:8443/wlcg
  - are you allowed?
  - if not, why?
- Enable the `storage.read:/` scope for your client and try to access to https://xfer.cr.cnaf.infn.it:8443/wlcg

Token exchange

- RFC 8693
- This flow has been designed to satisfy the needs to access resources hosted by other downstream services on behalf of the user
  - allows a Resource Server **A** to make calls to a backend service **C** on behalf of the requesting user **B**
  - the RS is an OAuth 2 Client of the **AS**
- Allows a Client to request the exchange of an AT with another AT (and potentially a RT to renew such AT)
- Preferably used when the exchanged token and the new token are requested by two different Clients
- Useful to implement **controlled delegation of privileges** between two registered client applications



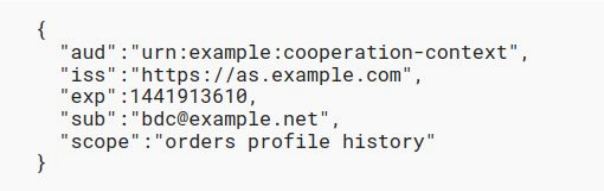
Token exchange

- The new token might be an AT that
  - is more **narrowly scoped** for the downstream service
  - has an **audience** different from the original token (which corresponds to the RS)
- In order to request a token exchange, a Client must be configured with the `urn:ietf:params:oauth:grant-type:token-exchange` grant type enabled
- Terminology:
  - **subject token** represents the subject access token that the Client wants to exchange
  - **actor token** represents the new token issued during a token exchange flow
- The ***act*** claim is a JSON object which identifies the acting party to whom authority has been delegated. It provides a representation of a delegation chain
  - members in the JSON object are claims that identify the actor
  - *i.e.*, it contains at least the *sub* claim
  - a chain of delegation can be expressed by nesting one *act* claim within another: the least recent actor is the most deeply nested

Impersonation vs. delegation

Impersonation

- When a subject A impersonates B, A has all the rights of B and it is indistinguishable fr B
- When A interacts within any other entity, A is B within the scope authorized by the toke
- Basically, the process **allows a subject to change to a different subject**
  - an application or API cannot determine by looking at the token if the subject is the entity that was actually logged in or not

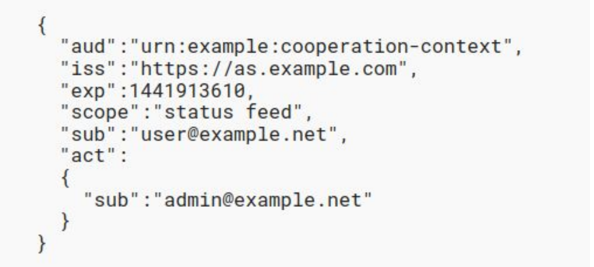


```
{
  "aud":"urn:example:cooperation-context",
  "iss":"https://as.example.com",
  "exp":1441913610,
  "sub":"bdc@example.net",
  "scope":"orders profile history"
}
```

From RFC 8693

Delegation

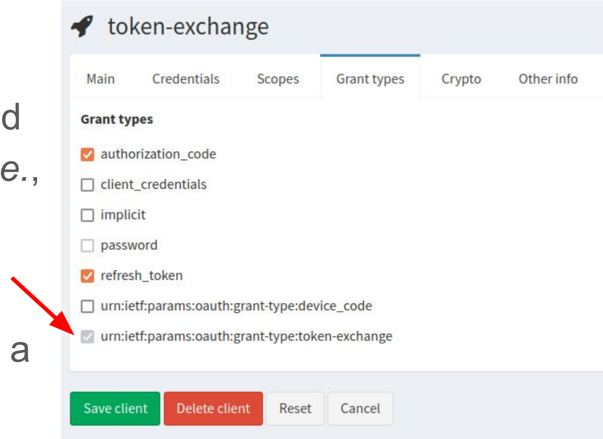
- With delegation A still has its own identity, which is separated from B
- When A interacts within another entity, it is explicit that A is representing B, because B has delegated some of its rights to A
  - the token contains explicit information that one subject delegates its rights to another entity
  - the subject can decide to **only delegate certain rights** to another subject



```
{
  "aud":"urn:example:cooperation-context",
  "iss":"https://as.example.com",
  "exp":1441913610,
  "scope":"status feed",
  "sub":"user@example.net",
  "act":
  {
    "sub":"admin@example.net"
  }
}
```

Token exchange in IAM

- The current IAM implementation (v1.8.0) does not support delegation
  - the scopes requested during a token exchange have to be enabled also by the Client which requested the subject token
- The token exchange grant is disabled by default for dynamically registered Clients; it can only be enabled by administrators for few trusted Clients (*i.e.*, VO central services)
- `offline_access` privileges can be delegated across trusted Client applications using token exchange (*i.e.* IAM allows to exchange an AT for a longer-lived RT)
  - a token obtained with a token exchange cannot be further exchanged by the same Client
  - the lifetime of the RT depends on the Client configuration
  - a token exchange request from the same Client which requested the subject token is forbidden if `offline_access` is included among the requested scopes. This prevents the Client to extend indefinitely the lifetime of an exchanged RT



Grant types

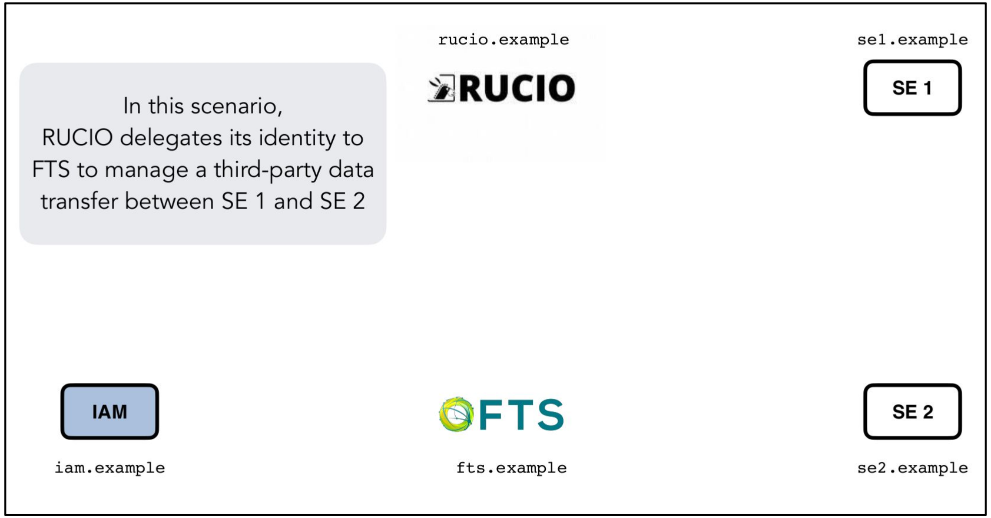
☑ authorization_code
☐ client_credentials
☐ implicit
☐ password
☑ refresh_token
☐ urn:ietf:params:oauth:grant-type:device_code
☑ urn:ietf:params:oauth:grant-type:token-exchange

Token exchange: use case

Example: moving some of my files with RUCIO + FTS

- I give RUCIO permission to act on my behalf
- RUCIO then delegates this task to FTS, which still acts on my behalf to trigger third-party transfers across Storage Elements
  - Here two different Client apps act on my behalf (RUCIO and FTS)
- Different **scopes** are needed at different level of the infrastructure
- Token exchange **allows to provide tokens with minimum privileges** to each service without requiring that big fat tokens are used at the top of the chain

From WLCG CE Hackathon



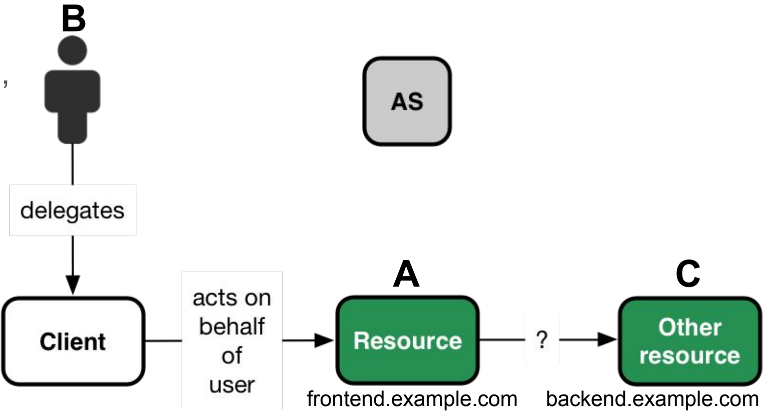
In this scenario, RUCIO delegates its identity to FTS to manage a third-party data transfer between SE 1 and SE 2

The token exchange flow

From section 2.3 of RFC 8693

User **B** wants to access the resource **C**.
Since resource **A** is an OAuth Client of the **AS** enabled for token exchanges,
B requests access to A using a bearer token issued by AS

```
GET /resource HTTP/1.1
Host: frontend.example.com
Authorization: Bearer accVkjcJyb4BWCxGsndESCJQbdFMogUC5PbRDqceLTC
```



The token exchange flow

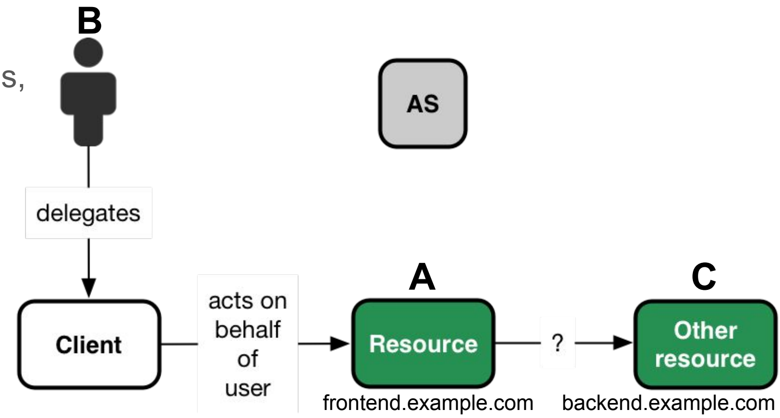
From section 2.3 of RFC 8693

User **B** wants to access the resource **C**.
Since resource **A** is an OAuth Client of the **AS** enabled for token exchanges,
B requests access to A using a bearer token issued by AS

```
GET /resource HTTP/1.1
Host: frontend.example.com
Authorization: Bearer accVkjcJyb4BWCxGsndESCJQbdFMogUC5PbRDqceLTC
```

Then A requests for a token exchange properly scoped for resource C

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Authorization: Basic cnMwODpsb25nLXNlY3VyZS1yYW5kb20tc2VjcmV0
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&resource=https%3A%2F%2Fbackend.example.com%2Fapi
&subject_token=accVkjcJyb4BWCxGsndESCJQbdFMogUC5PbRDqceLTC
&subject_token_type=
 urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
```



The token exchange flow

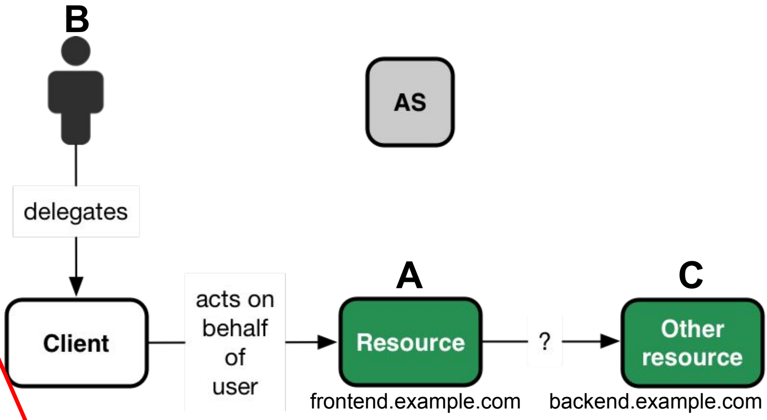
From section 2.3 of RFC 8693

User **B** wants to access the resource **C**.
Since resource **A** is an OAuth Client of **AS** enabled for token exchanges,
B requests access to A using a bearer token issued by AS

```
GET /resource HTTP/1.1
Host: frontend.example.com
Authorization: Bearer accVkjcJyb4BWCxGsndESCJQbdFMogUC5PbRDqceLTC
```

Then A requests for a token exchange properly scoped for resource C

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Authorization: Basic cnMwODpsb25nLXNlY3VyZS1yYW5kb20tc2VjcmV0
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&resource=https%3A%2F%2Fbackend.example.com%2Fapi
&subject_token=accVkjcJyb4BWCxGsndESCJQbdFMogUC5PbRDqceLTC
&subject_token_type=
 urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
```



The bearer token becomes the subject token

The token exchange flow

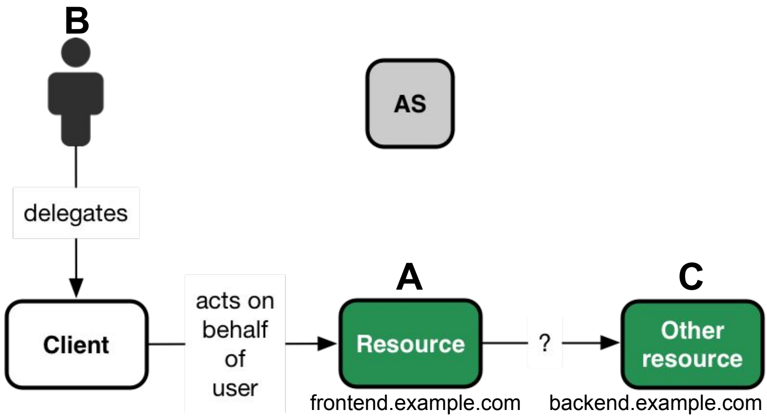
From section 2.3 of RFC 8693

User **B** wants to access the resource **C**.
Since resource **A** is an OAuth Client of **AS** enabled for token exchanges,
B requests access to A using a bearer token issued by AS

```
GET /resource HTTP/1.1
Host: frontend.example.com
Authorization: Bearer accVkjcJyb4BWCxGsndESCJQbdFMogUC5PbRDqceLTC
```

Then A requests for a token exchange properly scoped for resource C

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Authorization: Basic cnMwODpsb25nLXNlY3VyZS1yYW5kb20tc2VjcmV0
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&resource=https%3A%2F%2Fbackend.example.com%2Fapi
&subject_token=accVkjcJyb4BWCxGsndESCJQbdFMogUC5PbRDqceLTC
&subject_token_type=
 urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
```



HTTP basic authentication to AS using the credentials of the OAuth Client A

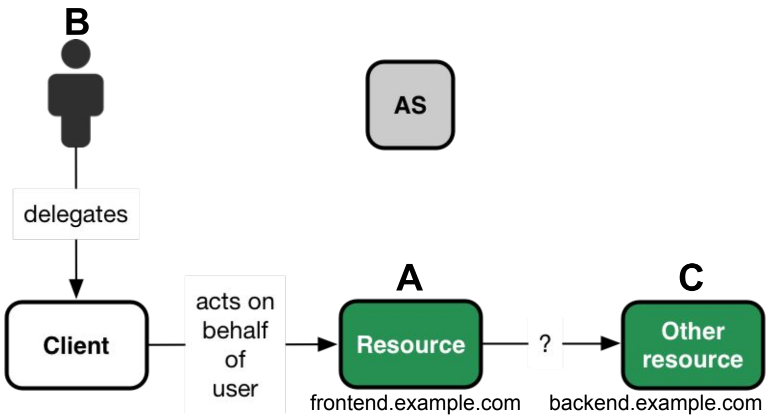
resource parameter indicates the location of the backend service (similar to *audience*)

The token exchange flow

From section 2.3 of RFC 8693

The **AS** validates the Client credentials and the subject token, and issues a new access token to **A**

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store

{
 "access_token":"eyJhbGciOiJFUzI1NiIsImtpZCI6IjllciJ9.eyJhdWQiOiJo
   dHRwczovL2JhY2tlbmQuZXhhbXBsZS5jb20iLCJpc3MiOiJodHRwczovL2FzLmV
   4YW1wbGUuY29tIiwiZXhwIjoxNDQxOTE3NTkzLCJpYXQiOjE0NDE5MTc1MzMsIn
   N1YiI6ImJkY0BleGFtcGxlLmNvbSIsInNjb3BlIjoiYXBpIn0.40y3ZgQedw6rx
   f59WlwHDD9jryFOr0_Wh3CGozQBihNBhnXEQgU85AI9x3KmsPottVMLPIWvmDCM
   y5-kdXjwhw",
 "issued_token_type":
     "urn:ietf:params:oauth:token-type:access_token",
 "token_type":"Bearer",
 "expires_in":60
}
```



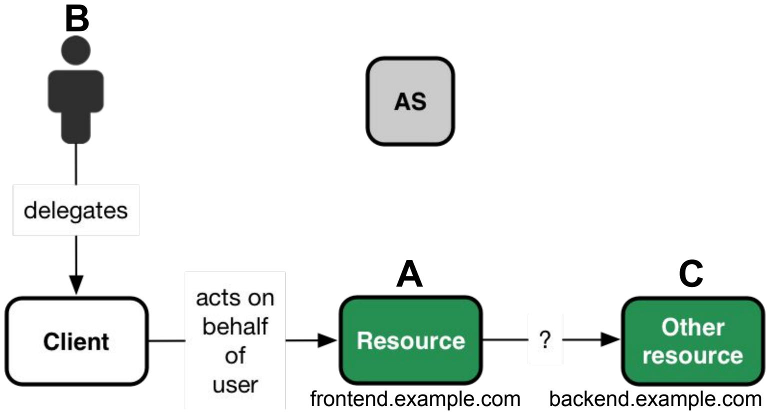
The access token is of *Bearer* type, meaning opaque to the Client (it only has to be sent in another HTTP request)

The token exchange flow

From section 2.3 of RFC 8693

Now **A** can finally use the newly acquired AT to access the backend server **C** using HTTP bearer authentication

```
GET /api HTTP/1.1
Host: backend.example.com
Authorization: Bearer eyJhbGciOiJFUzI1NiIsImtpZCI6IjllciJ9.eyJhdWQ
    iOiJodHRwczovL2JhY2tlbmQuZXhhbXBsZS5jb20iLCJpc3MiOiJodHRwczovL2
    FzLmV4YW1wbGUuY29tIiwiZXhwIjoxNDQxOTE3NTkzLCJpYXQiOjE0NDE5MTc1M
    zMsInN1YiI6ImJkY0BleGFtcGxlLmNvbSIsInNjb3BlIjoiYXBpIn0.40y3ZgQe
    dw6rxf59WlwHDD9jryFOr0_Wh3CGozQBihNBhnXEQgU85AI9x3KmsPottVMLPIW
    vmDCMy5-kdXjwhw
```



Example of a token exchange request

- Client authentication may be required by the AS
- A valid **subject token** is needed to get a new AT
- The *audience* request parameter can be used to suggest an audience for the requested AT
  - in IAM it is used in place of the *resource* parameter
- The token endpoint can be retrieved from the *well-known* endpoint
  - in IAM it is `/token`

```
$ curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d
grant_type=urn:ietf:params:oauth:grant-type:token-exchange -d
scope=${CLIENT_SCOPES} -d audience=${AUDIENCE} -d
subject_token=${SUBJECT_TOKEN} ${TOKEN_ENDPOINT}
```

Useful script

Client registration for token exchange request

- I got the subject token using the `oidc-agent` Client, where:
  - client id: `eb9e1cc2-f5e1-4a4b-b967-bccea266f09b`
  - scopes allowed: `openid, offline_access, storage.read:/, storage.create:/, compute.read, compute.modify`
  - then, the subject token is obtained with (`oidc-token` command explained in next slides)

```
SUBJECT_TOKEN=$(oidc-token -s "compute.read compute.modify" demo)
```

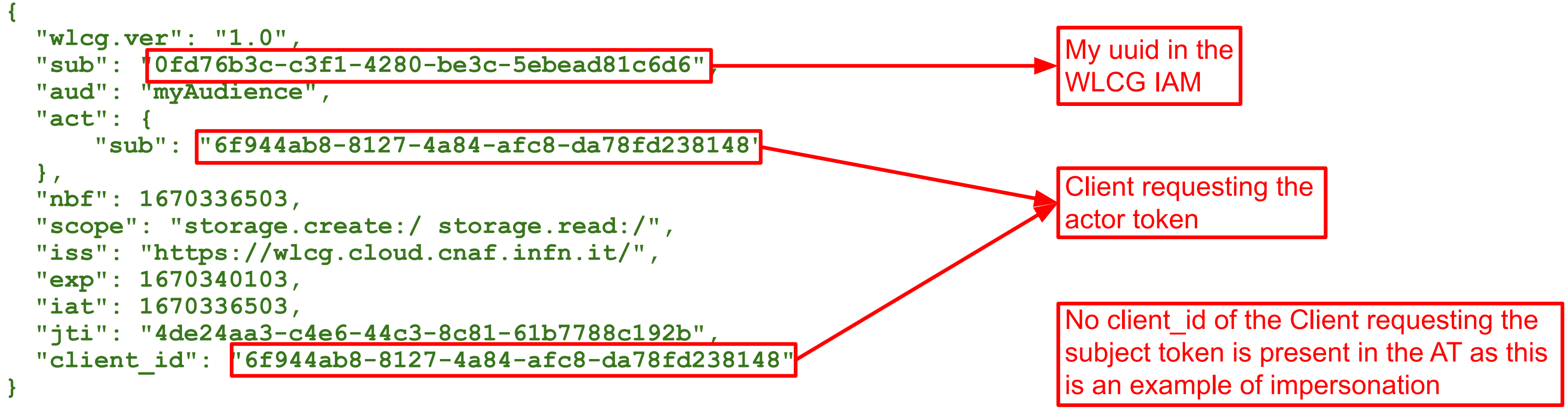
- I have registered a new client in the WLCG IAM instance, with the **token exchange** grant type (`urn:ietf:params:oauth:grant-type:token-exchange`) enabled
  - client id: `6f944ab8-8127-4a84-afc8-da78fd238148`
  - scopes allowed: `openid, offline_access, profile, storage.read:/, storage.create:/, compute.read, compute.create, compute.modify, wlcg.groups`

Real token exchange request

Authenticated POST request to the token endpoint

- I am not prompted to the consent page at this stage, since the subject token is used as authorization grant

```
$ curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=urn:ietf:params:oauth:grant-type:token-exchange
-d scope="storage.read:/ storage.create:/" -d audience=myAudience -d subject_token=${SUBJECT_TOKEN}
https://wlcg.cloud.cnaf.infn.it/token | jq -r .access_token |  tr -d '"' | cut -d. -f2 | base64 -d
2>/dev/null | jq
```



```
{
  "wlcg.ver": "1.0",
  "sub": "0fd76b3c-c3f1-4280-be3c-5ebead81c6d6",
  "aud": "myAudience",
  "act": {
      "sub": "6f944ab8-8127-4a84-afc8-da78fd238148"
  },
  "nbf": 1670336503,
  "scope": "storage.create:/ storage.read:/",
  "iss": "https://wlcg.cloud.cnaf.infn.it/",
  "exp": 1670340103,
  "iat": 1670336503,
  "jti": "4de24aa3-c4e6-44c3-8c81-61b7788c192b",
  "client_id": "6f944ab8-8127-4a84-afc8-da78fd238148"
}
```

My uuid in the WLCG IAM

Client requesting the actor token

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

Token exchange flow exercise

Prerequisites:

- Register a Client with oidc-agent enabling the `compute.read` and `compute.modify` scopes
- Register a new client in the same IAM instance, enabling the `offline_access`, `compute.read`, `compute.modify` and `compute.create` scopes
  - enable the client credentials grant to your Client configuration
  - ask the IAM admin administrator to enable also the token exchange grant
- obtain a subject token (named `SUBJECT_TOKEN_OIDC`) using the oidc-token command and asking for the `compute.read` scope
- obtain another subject token (named `SUBJECT_TOKEN_CLIENT_CRED`) using the client credentials flow and authenticating with the second client to the token endpoint

Token exchange flow exercise

- Ask for an AT using the token exchange flow and the `SUBJECT_TOKEN_OIDC`; request the
  - `compute.read`
  - `compute.read`, `compute.modify`
  - `compute.read`, `compute.create`

  scopes and try to predict which output would you get from the AS/IAM in each request
- Ask for an AT using the token exchange flow and the `SUBJECT_TOKEN_CLIENT_CRED` including the `offline_access` scope
  - understand and explain the output you get

Deprecated grant types

- Implicit
  - section 4.2 of RFC 6749
  - simplified authorization code flow optimized for web browser-based Clients (*e.g.*, JavaScript apps)
    - the AS issues directly an AT to the Client; code exchange is bypassed
    - the AS does not authenticate the Client (*i.e.* the Client is public); Client identity may be verified via the redirect URI used to deliver the access token
- Resource owner password credentials
  - section 4.3 of RFC 6749
  - the resource owner password credentials (*i.e.*, username and password) are used as authorization grant to obtain an AT
  - this flow prevents the typical delegation pattern OAuth has been designed for
- Public clients
  - Clients incapable of maintaining the confidentiality of their credentials (*e.g.*, executing on the device used by the resource owner), and incapable of secure Client authentication via any other means
  - not really deprecated, but discouraged in IAM

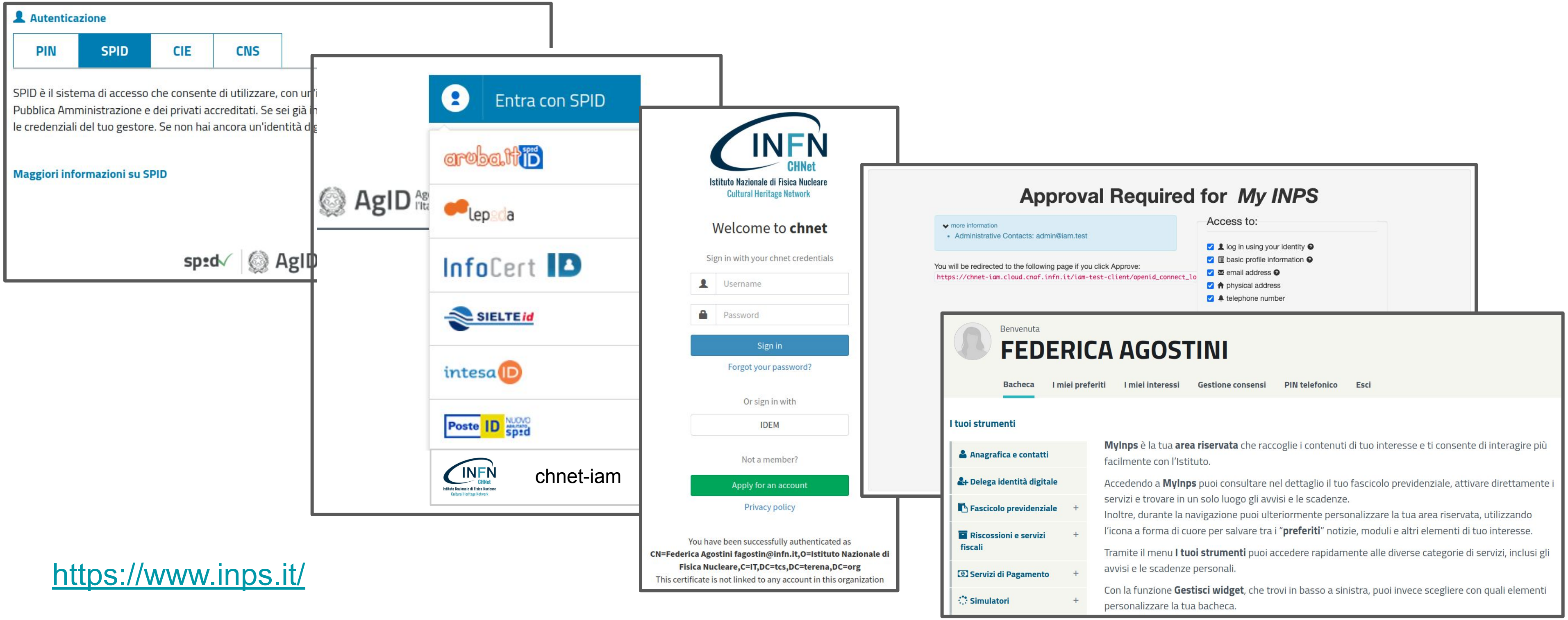
What’s new in OAuth2.1

- OAuth 2.1 (draft)
  - it is a draft with the aim of consolidating and simplifying the most commonly used features of OAuth 2.0
- New features of OAuth 2.1
  - PKCE (Proof Key for Code Exchange) is required for all OAuth Clients using the authorization code flow
    - it should be used in OAuth 2 by public Clients in order to prevent interception attack
  - redirect URIs must be compared using exact string matching
    - prevents using wildcards in the URI
  - the **implicit** grant is omitted from this specification
  - the **resource owner password credentials** grant is omitted
  - using bearer tokens in the query string of URIs is forbidden
  - refresh tokens for public Clients must either be sender-constrained or one-time use

OpenID Connect Federation 1.0

- Spec (draft 25 at December 2, 2022)
- OpenID Connect 1.0 extends OAuth 2 to provide a standard identity layer. It allows Clients to verify the identity of the end-user and to obtain basic profile information based on the authentication performed by an OP
  - *i.e.* who the user is and how it was authenticated → contained in the ID token
  - allows to establish login sessions (SSO)
- The OpenID Connect federation is a draft to standardize the concept of multilateral federation using the OIDC protocol, as it is now for SAML (*e.g.* EduGain)
  - A federation can be expressed as an agreement between parties that trust each other
  - The federation trust chains rely on cryptographically signed JWT
  - An Entity in the federation must be able to trust that other Entities it is interacting with belong to the same federation
  - In an OIDC federation, Entities are represented by OPs and RPs
  - The specification describes the technical trust infrastructure needed to build a dynamic and distributed trust network
- G. De Marco from Developers Italia (developers of SPID, CIE, *etc*) is one of the authors of the OIDC federation draft
  - Series of seminars hosted by GARR are available here

OIDC Federation: how it could work



Autenticazione

PIN | SPID | CIE | CNS

SPID è il sistema di accesso che consente di utilizzare, con un'i
Pubblica Amministrazione e dei privati accreditati. Se sei già i
le credenziali del tuo gestore. Se non hai ancora un'identità dig

Maggiori informazioni su SPID

Entra con SPID

aruba.it spid ID
lepida
InfoCert ID
SIELTE id
intesa ID
Poste ID NUOVO ABILITATO spid
INFN CHNet Istituto Nazionale di Fisica Nucleare Cultural Heritage Network

chnet-iam

INFN CHNet
Istituto Nazionale di Fisica Nucleare
Cultural Heritage Network

Welcome to **chnet**

Sign in with your chnet credentials

Username

Password

Sign in

Forgot your password?

Or sign in with

IDEM

Not a member?

Apply for an account

Privacy policy

You have been successfully authenticated as
CN=Federica Agostini fagostin@infn.it,O=Istituto Nazionale di Fisica Nucleare,C=IT,DC=tcs,DC=terena,DC=org
This certificate is not linked to any account in this organization

Approval Required for *My INPS*

more information
- Administrative Contacts: admin@iam.test

You will be redirected to the following page if you click Approve:
https://chnet-iam.cloud.cnaf.infn.it/iam-test-client/openid_connect_lo

Access to:

☑ log in using your identity
☑ basic profile information
☑ email address
☑ physical address
☑ telephone number

Benvenuta

FEDERICA AGOSTINI

Bacheca | I miei preferiti | I miei interessi | Gestione consensi | PIN telefonico | Esci

I tuoi strumenti

- Anagrafica e contatti
- Delega identità digitale
- Fascicolo previdenziale +
- Riscossioni e servizi fiscali +
- Servizi di Pagamento +
- Simulatori +

MyInps è la tua **area riservata** che raccoglie i contenuti di tuo interesse e ti consente di interagire più facilmente con l'Istituto.

Accedendo a **MyInps** puoi consultare nel dettaglio il tuo fascicolo previdenziale, attivare direttamente i servizi e trovare in un solo luogo gli avvisi e le scadenze.
Inoltre, durante la navigazione puoi ulteriormente personalizzare la tua area riservata, utilizzando l'icona a forma di cuore per salvare tra i "**preferiti**" notizie, moduli e altri elementi di tuo interesse.

Tramite il menu **I tuoi strumenti** puoi accedere rapidamente alle diverse categorie di servizi, inclusi gli avvisi e le scadenze personali.

Con la funzione **Gestisci widget**, che trovi in basso a sinistra, puoi invece scegliere con quali elementi personalizzare la tua bacheca.

https://www.inps.it/

JWT-based client authentication

- RFC 7523
- The OAuth and OIDC protocols support **JWT-based client-authentication**, *i.e.*
    - Clients authenticate to the token issuer sending a **signed JWT**
    - it replaces the client credentials (`client_id`, `client_secret`)
- The AS inspects the JWT, resolves the `client_id` and verifies the JWT using either a **shared secret** or a **public key** linked to the Client configuration
- Clients need to know how to generate and sign a JWT
- Pros: time-limited Client credentials under the control of the Client

JWT client authN methods

In order to use a JWT Bearer Token as client authentication method, the following query parameters have to be added to the token request:

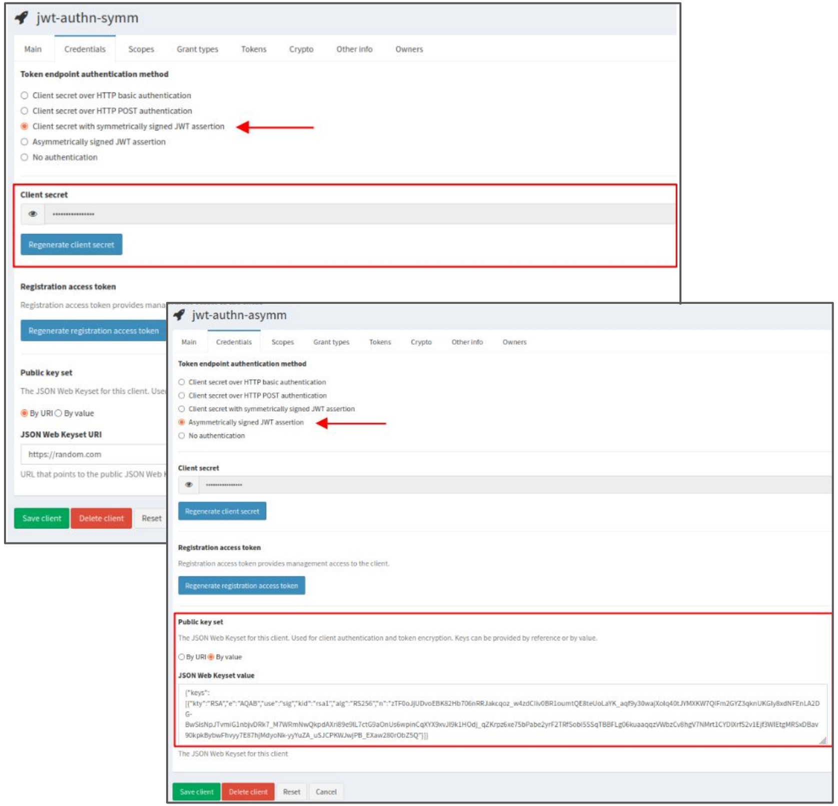
- *client_assertion_type*, whose value is `urn:ietf:params:oauth:client-assertion-type:jwt-bearer`
- *client_assertion*, which must contain a JWT
- *scope* (if omitted the AS returns all the ones allowed by the Client)

```
POST /token HTTP/1.1
Host: iam.local.io
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&
client_assertion_type=urn%3Aietf%3Aparams%3Aoauth%3A
client-assertion-type%3Ajwt-bearer&
client_assertion=eyJhbGciOiJSUzI1NiIsImtpZCI6IjIyIn0.eyJpc3Mi[...].cC4hiUPo[...]
```

JWT assertion signature

In IAM users can choose between two types of JWT assertion signatures (documentation here):

- a **symmetrically-signed JWT assertion**, signed with the `client_secret`
  - `client_secret_jwt` shared secret scheme, Clients that have received a client secret from the AS create a JWT using an HMAC SHA algorithm (*i.e.*, HMAC SHA-256)
- an **asymmetrically-signed JWT assertion,** signed with a RSA private key
  - `private_key_jwt` Clients that have registered a public key sign a JWT using the corresponding private key
  - IAM retrieves the RSA public key used to validate the JWT assertion from a JSON Web Keyset that can be provided during Client registration
    - by URI, or
    - by value



jwt-authn-symm

Token endpoint authentication method

○ Client secret over HTTP basic authentication
○ Client secret over HTTP POST authentication
● Client secret with symmetrically signed JWT assertion
○ Asymmetrically signed JWT assertion
○ No authentication

Client secret

••••••••••••••••

Registration access token

Registration access token provides mana

Public key set

The JSON Web Keyset for this client. Use

● By URI ○ By value

JSON Web Keyset URI

https://random.com

URL that points to the public JSON Web

jwt-authn-asymm

Token endpoint authentication method

○ Client secret over HTTP basic authentication
○ Client secret over HTTP POST authentication
○ Client secret with symmetrically signed JWT assertion
● Asymmetrically signed JWT assertion
○ No authentication

Client secret

••••••••••••••••

Registration access token

Registration access token provides management access to the client.

Public key set

The JSON Web Keyset for this client. Used for client authentication and token encryption. Keys can be provided by reference or by value.

○ By URI ● By value

JSON Web Keyset value

```
{"keys":
[{"kty":"RSA","e":"AQAB","use":"sig","kid":"rsa1","alg":"RS256","n":"zTF0oJjUDvoEBK82Hb706nRRJakcqoz_w4zdCIlv0BR1oumtQE8teUoLaYK_aqf9y30wajXolq40tJYMXKW7QIFm2GYZ3qknUKGly8xdNFEnLA2D
G-
BwSisNpJTvmiG1nbjvDRk7_M7WRmNwQkpdAXri89e9lL7ctG9aOnUs6wpinCqXYX9xvJl9k1HOdj_qZKrpz6xe75bPabe2yrF2TRfSobI5SSqTBBFLg06kuaaqqzVWbzCv8hgV7NMrt1CYDIXrfS2v1Ejf3WIEtgMRSxDBav
90kpkBybwFhvyy7E87hjMdyoNk-yyYuZA_uSJCPKWJwjPB_EXaw280rObZ5Q"}]}
```

The JSON Web Keyset for this client

JWT assertion

Example of symmetrically signed decoded assertion:

```
## Header
{
  "alg": "HS256"
}
## Payload
{
  "sub": "181f26f9-4562-4919-b718-759241485335",
  "aud": "https://iam.local.io/token",
  "nbf": 1649162752,
  "iss": "181f26f9-4562-4919-b718-759241485335",
  "exp": 1651754752,
  "iat": 1649162752,
  "jti": "120240aa-e389-4a55-8384-f4d7a54c2633"
}
```

Example of asymmetrically signed decoded assertion:

```
## Header
{
  "alg": "RS256",
  "kid": "rsa1"
}
## Payload
{
  "sub": "bdb6ca15-be9c-470a-81dc-69d30dabb340",
  "aud": "https://iam.local.io/token",
  "nbf": 1649162752,
  "iss": "bdb6ca15-be9c-470a-81dc-69d30dabb340",
  "exp": 1651754752,
  "iat": 1649162752,
  "jti": "f4392c1e-6d6a-423e-8e5e-5d114585f750"
}
```

JWT-based client authentication example

Example of an HTTP POST request to the token endpoint where the Client is authenticated with **JWT assertion** and is authorized via the **client credentials** OAuth 2 flow.

```
$ JWTA=eyJhbGciOiJI[...]I6IkpXVCJ9.eyJpc3[...]wfQ.3g9o80SyE[...]W_0dNpwg
$ curl -d client_assertion=${JWTA} -d
client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer -d
grant_type=client_credentials -d scope=storage.read:/ https://iam.local.io/token | jq
```

```
{
  "access_token": "eyJraWQiOiJyc2ExIiwiY...",
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": "storage.read:/"
}
```

Main WLCG possible use cases

- Reduced risk of exposed Client credentials
  - JWT-based auth is a requirement for high security OpenID-connect use, *e.g.*, the Financial Grade API OpenID Connect profile
- Time-limited credential delegation
- Examples:
  - RUCIO server delegates short-lived JWT client credential to RUCIO client that can be used for time-limited token renewal
  - VO job framework delegates short-lived JWT client credential to payload job for time-limited token renewal

oidc-agent

oidc-agent

- oidc-agent is a set of tools which allow to manage tokens using command line.
  It follows the ssh-agent design
- Source code (developed by KIT team)
- Documentation
- Installation
  - repo file available
  - supports debian- and rpm-based linux distributions
  - available for MacOS (using `brew`)
  - … and also for windows

What oidc-agent does during client registration

In order to request for tokens, you firstly have to register a Client.
With oidc-agent you have to run

```
$ eval $(oidc-agent)
$ oidc-gen -w device demo
```

The OAuth authorization flow used here is **device**. `oidc-agent` allows to specify one: code, device, password or refresh. Default to code.
The `oidc-gen -w device` command basically replaces the curl version reported in the *Device code flow* slides

What oidc-agent does during client registration

In order to request for tokens, you firstly have to register a Client.
With oidc-agent you have to run

```
$ eval $(oidc-agent)
$ oidc-gen -w device demo
[1] https://wlcg.cloud.cnaf.infn.it/
[2] https://iam-dev.cloud.cnaf.infn.it/
…
Issuer [https://iam-demo.cloud.cnaf.infn.it/]: 1
```

Select the AS where you want to register the Client.
I choose WLCG IAM here

What oidc-agent does during client registration

In order to request for tokens, you firstly have to register a Client.
With oidc-agent you have to run

```
$ eval $(oidc-agent)
$ oidc-gen -w device demo
[1] https://wlcg.cloud.cnaf.infn.it/
[2] https://iam-dev.cloud.cnaf.infn.it/
…
Issuer [https://iam-demo.cloud.cnaf.infn.it/]: 1
The following scopes are supported: openid profile email offline_access wlcg wlcg.groups storage.read:/ storage.create:/ compute.read
compute.modify compute.create compute.cancel storage.modify:/ eduperson_scoped_affiliation eduperson_entitlement eduperson_assurance
storage.stage:/
Scopes or 'max' (space separated) [openid profile offline_access]: storage.read:/ storage.create:/ compute.read compute.modify
```

List of supported scopes is taken from the `/.well-known/openid-configuration` endpoint of the AS. Insert the necessary scopes only

What oidc-agent does during client registration

In order to request for tokens, you firstly have to register a Client.
With oidc-agent you have to run

```
$ eval $(oidc-agent)
$ oidc-gen -w device demo
[1] https://wlcg.cloud.cnaf.infn.it/
[2] https://iam-dev.cloud.cnaf.infn.it/
…
Issuer [https://iam-demo.cloud.cnaf.infn.it/]: 1
The following scopes are supported: openid profile email offline_access wlcg wlcg.groups storage.read:/ storage.create:/ compute.read
compute.modify compute.create compute.cancel storage.modify:/ eduperson_scoped_affiliation eduperson_entitlement eduperson_assurance
storage.stage:/
Scopes or 'max' (space separated) [openid profile offline_access]: storage.read:/ storage.create:/ compute.read compute.modify
Registering Client ...
Generating account configuration ...
accepted
```

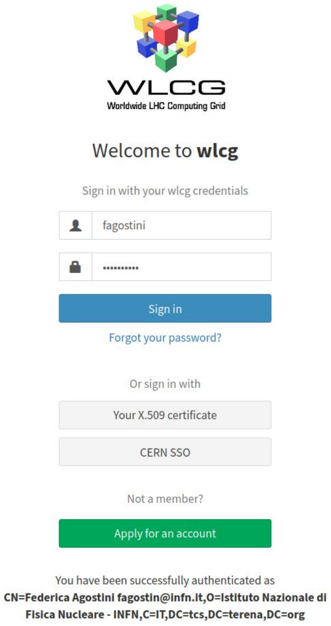
The `oidc-agent` Client has been registered in the WLCG IAM. The configuration details have been saved locally

What oidc-agent does during client registration

In order to request for tokens even when the AT is expired, you have to obtain a refresh token

```
$ eval $(oidc-agent)
$ oidc-gen -w device demo
[1] https://wlcg.cloud.cnaf.infn.it/
[2] https://iam-dev.cloud.cnaf.infn.it/
…
Issuer [https://iam-demo.cloud.cnaf.infn.it/]: 1
The following scopes are supported: openid profile email offline_access wlcg wlcg.groups storage.:
compute.modify compute.create compute.cancel storage.modify:/ eduperson_scoped_affiliation eduper:
storage.stage:/
Scopes or 'max' (space separated) [openid profile offline_access]: storage.read:/ storage.create:,
Registering Client ...
Generating account configuration ...
accepted

Using a browser on any device, visit:
https://wlcg.cloud.cnaf.infn.it/device

And enter the code: LYE0TT
```

The AS asks the user to authenticate before to insert the code



Welcome to **wlcg**

Sign in with your wlcg credentials

fagostini

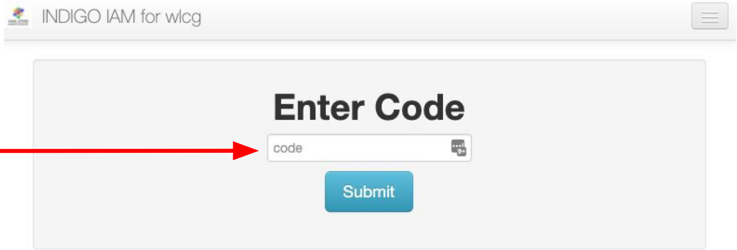
••••••••••

You have been successfully authenticated as **CN=Federica Agostini fagostin@infn.it,O=Istituto Nazionale di Fisica Nucleare - INFN,C=IT,DC=tcs,DC=terena,DC=org**

What oidc-agent does during client registration

In order to request for tokens even when the AT is expired, you have to obtain a refresh token

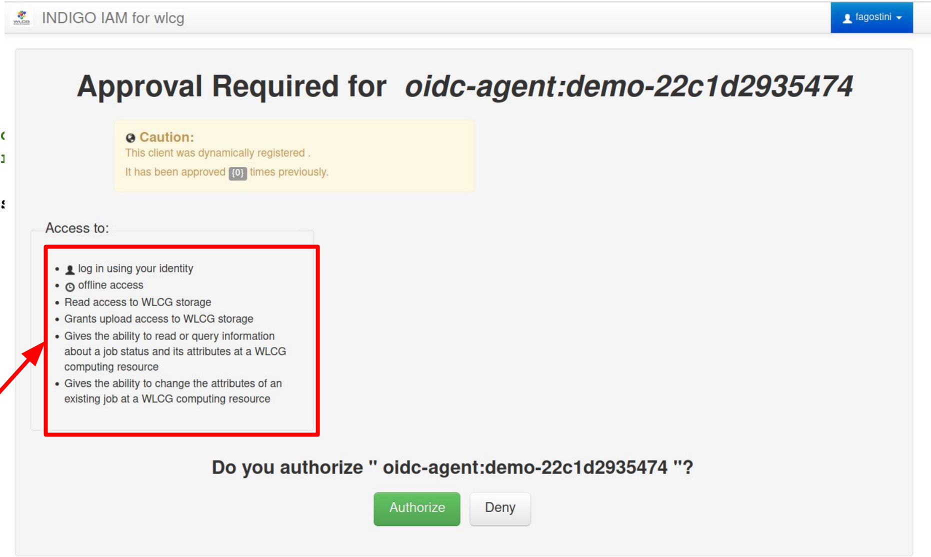
```
$ eval $(oidc-agent)
$ oidc-gen -w device demo
[1] https://wlcg.cloud.cnaf.infn.it/
[2] https://iam-dev.cloud.cnaf.infn.it/
…
Issuer [https://iam-demo.cloud.cnaf.infn.it/]: 1
The following scopes are supported: openid profile email offline_access wlcg wlcg.groups storage.read:/ storage.create:/ compute.read
compute.modify compute.create compute.cancel storage.modify:/ eduperson_scoped_affiliation eduperson_entitlement eduperson_assurance
storage.stage:/
Scopes or 'max' (space separated) [openid profile offline_access]: storage.read:/ storage.create:/ compute.read compute.modify
Registering Client ...
Generating account configuration ...
accepted

Using a browser on any device, visit:
https://wlcg.cloud.cnaf.infn.it/device

And enter the code: LYE0TT
```



What oidc-agent does during client registration

In order to request for tokens even when the AT is expired, you have to obtain a refresh token

```
$ eval $(oidc-agent)
$ oidc-gen -w device demo
[1] https://wlcg.cloud.cnaf.infn.it/
[2] https://iam-dev.cloud.cnaf.infn.it/
…
Issuer [https://iam-demo.cloud.cnaf.infn.it/]: 1
The following scopes are supported: openid profile email offline_acc
compute.modify compute.create compute.cancel storage.modify:/ eduper
storage.stage:/
Scopes or 'max' (space separated) [openid profile offline_access]: s
Registering Client ...
Generating account configuration ...
accepted

Using a browser on any device, visit:
https://wlcg.cloud.cnaf.infn.it/device

And enter the code: LYEOTT
```



INDIGO IAM for wlcg

fagostini

Approval Required for *oidc-agent:demo-22c1d2935474*

Caution:
This client was dynamically registered .
It has been approved {0} times previously.

Access to:

- log in using your identity
- offline access
- Read access to WLCG storage
- Grants upload access to WLCG storage
- Gives the ability to read or query information about a job status and its attributes at a WLCG computing resource
- Gives the ability to change the attributes of an existing job at a WLCG computing resource

Do you authorize " oidc-agent:demo-22c1d2935474 "?

Authorize Deny

The AS asks for the consent of the user to access the information linked to the requested scopes

What oidc-agent does during client registration

In order to request for tokens even when the AT is expired, you have to obtain a refresh token

```
$ eval $(oidc-agent)
$ oidc-gen -w device demo
[1] https://wlcg.cloud.cnaf.infn.it/
[2] https://iam-dev.cloud.cnaf.infn.it/
…
Issuer [https://iam-demo.cloud.cnaf.infn.it/]: 1
The following scopes are supported: openid profile email offline_access wlcg wlcg.groups storage.read:/ storage.create:/ compute.read
compute.modify compute.create compute.cancel storage.modify:/ eduperson_scoped_affiliation eduperson_entitlement eduperson_assurance
storage.stage:/
Scopes or 'max' (space separated) [openid profile offline_access]: storage.read:/ storage.create:/ compute.read compute.modify
Registering Client ...
Generating account configuration ...
accepted

Using a browser on any device, visit:
https://wlcg.cloud.cnaf.infn.it/device

And enter the code: LYE0TT

[ Polling the device code verification from the https://wlcg.cloud.cnaf.infn.it/device/approve endpoint ]

Enter encryption password for account configuration 'demo': ***
Confirm encryption Password: ***
Everything setup correctly!
```

oidc-agent Client configuration

- Local oidc-agent Client configurations are saved in `~/.config/oidc-agent` or `~/.oidc-agent`
- In this example, `demo` is an alias for the oidc-agent Client
- The Client configuration is encrypted using the password set by the user during Client registration

```
$ cat ~/.config/oidc-agent/demo
932
P0ZXJ563imIdd9xWSDjsyFJFx2RfGr51
AWS5IEnedUX13Drf7DdfGg==
24:16:16:32:1:2:67108864:2
Nag1k2Epdlyvjni47XRCJs7RUg1Q314LjLJam0bEm0WkpcBnlvddeZ6lcZXZWGwnt+YzvGeu8ij6xjzKgrmWvNjJQKGxPW7G/0HD9PHwf+8jU2A8z5WqR2Axe0fZRQkVcCaOzmL+Ws
9E+xoaP/GFeTOVXUW6FlxRGBALfix/wkvOUwNi6tbyPHGwtlXpBHie/8eR5NW+vohuwZOmoMyNpu3paQq3k4563sfgtMqU/SjflrYPupgyAVs8MwMT36Mhv6Bsdz5LgHTGALIkz6jM
/78wDPMJH0zdhqRvkP4ugaFJ2cOx8XVpMD3WbcRPFgDYgOhA+nCE/gTgwoHiCUntaCcCPORCT6QjwZ5Ulc5Uuq5MSFM1DGpllPPOuiGbdY5JMVcwgy+nHAmr1tXRvXIqqS7j8chH6O
f65B0S25eIZwnsLkPtD418dTp4wTc4eOzsx/LjwDg//R/vaG1vRsfDH3ZcvTupmak7UsVmzglHnPblK/WAqOOIdJrYC0y5SpyeSoAkKpQSVqmugC3s54JM3orjCcVRJa2LDZzNHlR4
R139CInViyaCu3BbqX246ULJe7GIG6wdkeDZS83gdMs9ItzNnhb4yOkndDPUubg0uUzdC3xkLopjkBiwKS6qkYPj0q3e8/GfCiR5PeagoCt8fYzr3XRJPfSAAXkAl2WibuJwO2yNYV
P9WvNI++DNfQtNYzCJNbPJXYVyIOzWuvTol8bMpuRcN5cP8YqEqYS+RCk7OXRurdTeL3e3KPbRrT07b8DfnVs61phwLZ2rOHYY0A1PrOO5ojjHyY1rFbqRXQU/M5WMt4ggGZ3rrbMI
uVvAyp0KqjS7ocjuBxw+10vvbvi5kdImDTfrdaEFxFMVL+2oWL2TcU8ytEF6wSiajb5yMyi1cOXJtIwFgEnnhqOdRvGg6TGoTn2c7KYOAS3mGaZkPmUGvC1K4+fztXTP1TWBRigTsF
1j5fqwYFvYPLjak7crtMAGKejmk34K8ZmSSduQp501ZXmmwngFr8NFj3WyK89iTExcYceQeg64zQH6BcZoEUb5z4gGZwwD/Yz8cnypX8vIByAxA18fbevqDrIDfzGwrZQCbQXz6Dhk
382RIrPXIHPvVMPcgqj4X/ywTWJw38JDLCvXdNoYQaektIfHikyLN5ktAWDVEdiEtgjO/fh3IzJRXNp9hzDZYreoFQXtbu9baOyKEoID7SZIVw9l8DT0vRuuyDma+dqa3/HAxe/xen
c=
H7tVFJiY/or8PH4wdvSiVG1gFtadO+POA77mDJjpbmA=
Generated using version: 4.3.2
```

oidc-agent client configuration

- The decrypted oidc-agent configuration can be checked with `oidc-add -p` command
- The encryption pwd is required. It can be saved in an env variable or a file and passed to the agent using one of the `pw-cmd`/`pw-env`/`pw-file` option (useful for automated environment)

```
$ oidc-add -p demo
Enter decryption password for account config 'demo': ***
{
    "name":    "demo",
    "client_name":    "oidc-agent:demo-22c1d2935474",
    "issuer_url":    "https://wlcg.cloud.cnaf.infn.it/",
    "config_endpoint":    "https://wlcg.cloud.cnaf.infn.it/.well-known/openid-configuration",
    "device_authorization_endpoint":    "https://wlcg.cloud.cnaf.infn.it/devicecode",
    "daeSetByUser":    0,
    "client_id":    "eb9e1cc2-f5e1-4a4b-b967-bccea266f09b",
    "client_secret":    "xxx",
    "refresh_token":    "eyJhbGciOiJub25lIn0.eyJqdGkiOiIyMTAwOTcy…",
    "cert_path":    "/etc/pki/tls/certs/ca-bundle.crt",
    "scope":    "storage.read:/ storage.create:/ compute.read compute.modify openid offline_access",
    "audience":    "",
    "oauth":    0,
    "redirect_uris":    ["edu.kit.data.oidc-agent:/redirect", "http://localhost:43708", "http://localhost:8080",
"http://localhost:4242"],
    "username":    "",
    "password":    ""
}
```

oidc-agent client configuration

```
$ oidc-add -p demo
Enter decryption password for account config 'demo': ***
{
    "name":    "demo",
    "client_name":    "oidc-agent:demo-22c1d2935474",
    "issuer_url":    "https://wlcg.cloud.cnaf.infn.it/",
    "config_endpoint":    "https://wlcg.cloud.cnaf.infn.it/.well-known/openid-configuration",
    "device_authorization_endpoint":    "https://wlcg.cloud.cnaf.infn.it/devicecode",
    "daeSetByUser":    0,
    "client_id":    "eb9e1cc2-f5e1-4a4b-b967-bccea266f09b",
    "client_secret":    "xxx",
    "refresh_token":    "eyJhbGciOiJub25lIn0.eyJqdGkiOiIyMTAwOTcy…",
    "cert_path":    "/etc/pki/tls/certs/ca-bundle.crt",
    "scope":    "storage.read:/ storage.create:/ compute.read compute.modify openid offline_access",
    "audience":    "",
    "oauth":    0,
    "redirect_uris":    ["edu.kit.data.oidc-agent:/redirect", "http://localhost:43708", "http://localhost:8080",
"http://localhost:4242"],
    "username":    "",
    "password":    ""
}
```

Even if not requested by the user, oidc-agent adds the `openid` and `offline_access` scopes during the token request. This triggers the AS to issue an ID and refresh token; the latter is stored by oidc-agent

What oidc-agent does when requesting a token

- When a user wants to obtain an AT with the `oidc-token` command, the RT stored during the Client registration is used
  - oidc-agent triggers an OAuth **refresh token flow**
- No need to re-run `oidc-gen` before. Just start the agent (`eval $(oidc-agent)`) and load the Client configuration (`oidc-add <client-alias>`) in case a new session is started
- Limit the scopes requested for your token as much as possible. The `oidc-token` command without arguments will request all the scopes allowed by your Client

```
$ oidc-token -s storage.read:/myPath demo | cut -d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "0fd76b3c-c3f1-4280-be3c-5ebead81c6d6",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1669042833,
  "scope": "storage.read:/myPath",
  "iss": "https://wlcg.cloud.cnaf.infn.it/",
  "exp": 1669046433,
  "iat": 1669042833,
  "jti": "e5bef508-1d95-4530-ba8f-d1c98563b479",
  "client_id": "eb9e1cc2-f5e1-4a4b-b967-bccea266f09b"
}
```

My uuid on the WLCG IAM instance

client_id of the `demo` client

What oidc-agent does when requesting a token

The same token request can be performed via `curl` using the command just learnt:

```
$ curl -s -L -u ${CLIENT_ID}:${CLIENT_SECRET} -d grant_type=refresh_token -d scope=storage.read:/myPath -d
refresh_token=${REFRESH_TOKEN} https://wlcg.cloud.cnaf.infn.it/token | jq .access_token | tr -d '"' | cut
-d. -f2 | base64 -d 2>/dev/null | jq
{
  "wlcg.ver": "1.0",
  "sub": "0fd76b3c-c3f1-4280-be3c-5ebead81c6d6",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1669044151,
  "scope": "storage.read:/myPath",
  "iss": "https://wlcg.cloud.cnaf.infn.it/",
  "exp": 1669047751,
  "iat": 1669044151,
  "jti": "2db33e00-616a-4156-ac06-a091430bbe33",
  "client_id": "eb9e1cc2-f5e1-4a4b-b967-bccea266f09b"
}
```

My uuid on the WLCG IAM instance

client_id of the `demo` client

Many other options can be used together with the `oidc-gen`/`oidc-add`/`oidc-token` commands to handle the Client configuration and token requests.

To know more about it, read the documentation !

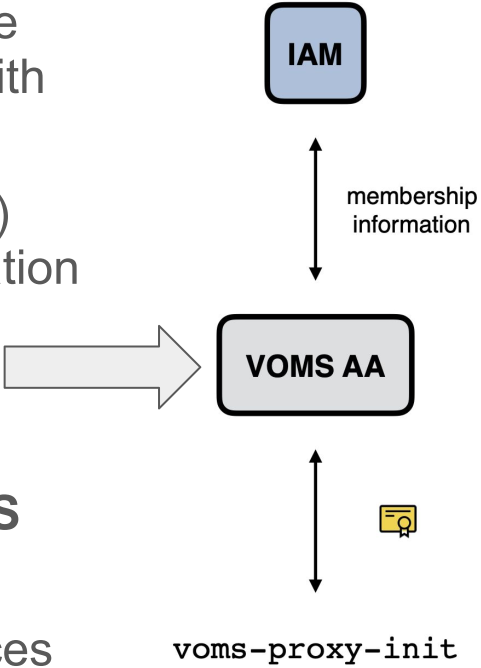
Usage in WLCG

Evolution of the WLCG AAI beyond X.509

- To access computing and storage resources in the WLCG community, users use a **VOMS proxy**, which provides information about
  - who the user is
  - for which VO it is acting
  - what it can do on the infrastructure (*i.e.*, VOMS groups and roles)
- In the near future we will use **tokens**, which will provide similar information
- Tokens are obtained from a VO token issuer (*e.g.*, INDIGO IAM) using **OIDC**
- Tokens are sent to services/resources following **OAuth** recommendations (*e.g.*, embedded in the header or an HTTP request)
- Tokens are self-contained, *i.e.* their integrity and validity can be verified locally with no callback to the token issuer

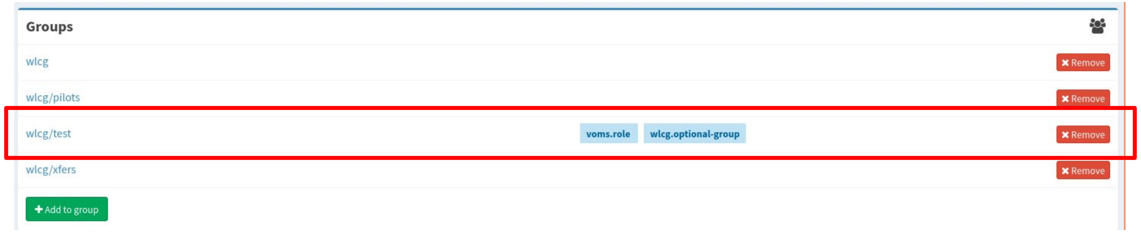
VOMS → IAM

- Knowing that the transition from X.509 to tokens will take time, IAM was designed to be **backward-compatible** with our existing infrastructure
- IAM **provides a VOMS Attribute Authority** (VOMS-AA) micro-service that can encode IAM membership information in a standard VOMS Attribute Certificate → can issue VOMS credentials (`voms-proxy-init`) understood by existing clients
- At some point **IAM will be the only authoritative VOMS server** for the infrastructure
- Proven compatibility with existing clients and Grid services



VOMS → IAM

- A voms-importer migration script has been developed to import users from the legacy VOMS to IAM
  - documentation here
  - **users will NOT have to re-register in mass** to IAM, and their IAM account will be automatically linked to their x509 certificate
- The VOMS information that is synchronized includes
  - VOMS Groups
  - VOMS Roles
  - VOMS Users: Personal information, X.509 certificates, Group and role membership, Generic attributes
- Both IAM and VOMS support the concepts of *group* and *role*. As an example,
  - group: `/wlcg/xfer` (VOMS) → `wlcg/xfer` (IAM)
  - role: `/wlcg/Role=test` (VOMS) → `wlcg/test` (IAM)



Groups

wlcg

wlcg/pilots

wlcg/test voms.role wlcg.optional-group

wlcg/xfers

In IAM it differs from a default group because it is represented with the `voms.role` and `wlcg.optional-group` labels

VOMS vs. IAM: what’s in common?

- **Attribute handling**
  - VOMS users can have assignable attributes → IAM has support for generic attributes as well
- **Group managers**
  - In VOMS, VO managers may delegate the approval of some group/role to other VO members → IAM supports for group managers, currently only to approve/reject group membership requests
- **AUP expiration**
  - Within VOMS, an expired AUP prevents to issue new VOMS X.509 proxies → an expired AUP signature forces the user to sign the AUP when the user tries to login into IAM and prevents the issuing of new tokens, the refresh of tokens or the issuing of VOMS attribute certificates

VOMS vs. IAM: what’s in common?

- **Roles**
  - VOMS roles → are replaced by “labelled” groups in IAM
- **Primary group**
  - Within VOMS, exists the concept of a primary group → the content of the `wlcg.groups` claim (*i.e.* list of user’s group memberships) in WLCG JWT tokens issued by IAM is an ordered list of groups. A WLCG JWT profile defines how a particular group ordering can be requested.
- **Additional certificates**
  - Within VOMS users can add additional certificates → IAM allows to link multiple certificates to an account (in the same way VOMS does)

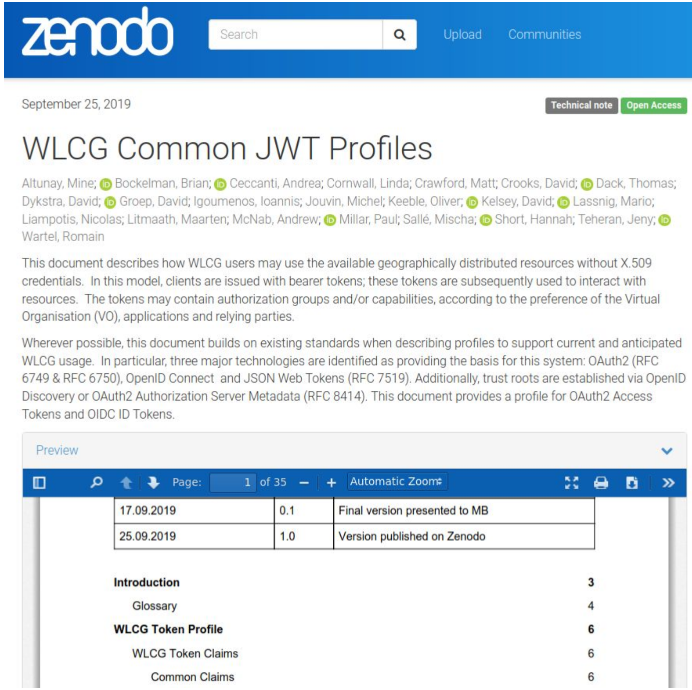
WLCG JWT profile

https://doi.org/10.5281/zenodo.3460258

“This document describes how WLCG users may use the available geographically distributed resources **without X.509 credentials**.”

“In this model, **clients are issued with bearer tokens**; these tokens are subsequently used to interact with resources. **The tokens may contain authorization groups and/or capabilities, according to the preference of the Virtual Organisation (VO), applications and relying parties.**”

“Three major technologies are identified as providing the basis for this system: **OAuth2**, **OpenID Connect** and **JSON Web Tokens**.”



September 25, 2019

WLCG Common JWT Profiles

Altunay, Mine; Bockelman, Brian; Ceccanti, Andrea; Cornwall, Linda; Crawford, Matt; Crooks, David; Dack, Thomas; Dykstra, David; Groep, David; Igoumenos, Ioannis; Jouvin, Michel; Keeble, Oliver; Kelsey, David; Lassnig, Mario; Liampotis, Nicolas; Litmaath, Maarten; McNab, Andrew; Millar, Paul; Sallé, Mischa; Short, Hannah; Teheran, Jeny; Wartel, Romain

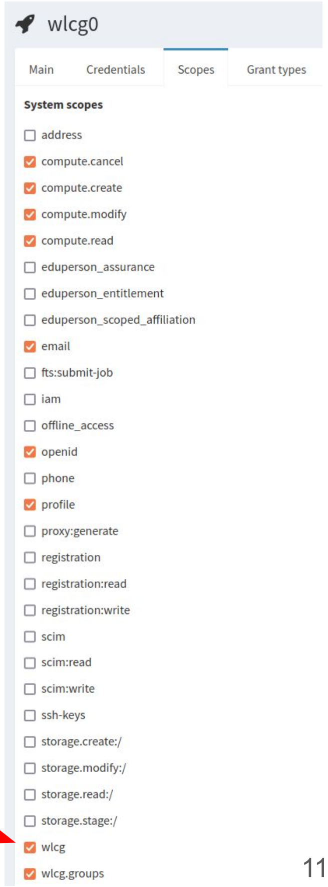
This document describes how WLCG users may use the available geographically distributed resources without X.509 credentials. In this model, clients are issued with bearer tokens; these tokens are subsequently used to interact with resources. The tokens may contain authorization groups and/or capabilities, according to the preference of the Virtual Organisation (VO), applications and relying parties.

Wherever possible, this document builds on existing standards when describing profiles to support current and anticipated WLCG usage. In particular, three major technologies are identified as providing the basis for this system: OAuth2 (RFC 6749 & RFC 6750), OpenID Connect and JSON Web Tokens (RFC 7519). Additionally, trust roots are established via OpenID Discovery or OAuth2 Authorization Server Metadata (RFC 8414). This document provides a profile for OAuth2 Access Tokens and OIDC ID Tokens.

17.09.2019	0.1	Final version presented to MB
25.09.2019	1.0	Version published on Zenodo

Supporting multiple profiles with IAM

- A profile is a set of rules that defines which information is included in
  - access tokens
  - id tokens
  - userinfo endpoint responses
  - introspection endpoint responses
- IAM allows to define a default profile (from configuration) that is used for all Clients, BUT
- it can be overridden per Client, **requesting a scope equal to the name of the profile**
  - example: a Client requesting a WLCG token with the compute.read scope should request `scope="wlcg compute.read"`
  - same logic used with the `openid` scope
- IAM currently supports three profiles: `iam`, `wlcg` and `aarc`
- The **`wlcg` profile** has been implemented in IAM following the WLCG JWT profile guidelines, in particular
  - the scope claim is always included in access tokens
  - groups are not included by default in access and ID tokens
  - groups can be requested with the `wlcg.groups` scope



System scopes

☐ address
☑ compute.cancel
☑ compute.create
☑ compute.modify
☑ compute.read
☐ eduperson_assurance
☐ eduperson_entitlement
☐ eduperson_scoped_affiliation
☑ email
☐ fts:submit-job
☐ iam
☐ offline_access
☑ openid
☐ phone
☑ profile
☐ proxy:generate
☐ registration
☐ registration:read
☐ registration:write
☐ scim
☐ scim:read
☐ scim:write
☐ ssh-keys
☐ storage.create:/
☐ storage.modify:/
☐ storage.read:/
☐ storage.stage:/
☑ wlcg
☑ wlcg.groups

WLCG specific token claims

- **wlcg.ver** version of the WLCG token profile the Relying Parties must understand to validate the token
  - it corresponds to the version of the WLCG JWT profile document
  - example: **`"wlcg.ver": "1.0"`**
- **wlcg.groups** group information about an authenticated End-User, following a UNIX-like path syntax
  - example: **`"wlcg.groups": ["/atlas", "/atlas/pilots", "/atlas/xfers"]`**
- **aud** represents the recipient the JWT is intended for
  - it is actually defined in the JWT and OpenID Connect core standard, BUT
  - the WLCG JWT profile specifies that the **`"https://wlcg.cern.ch/jwt/v1/any"`** audience must be accepted by all WLCG Relying Parties

Authorization models in WLCG

Capability-based authorization: *scope*

- When a capability is asserted, it has to be honoured by RS. It is **the VO** (*i.e.* the Authorization Server), NOT the RS, who **manages authorization within its area**
- The WLCG authorization model follows the recommendation of Section 3.3 of RFC 6749:
  - each desired capability should be requested in the scope request
  - if an entity is not entitled to a capability, the scope requested may be ignored by the server and the corresponding token may not have the corresponding claims
  - in this case, the AS must inform the Client
- The scopes limit what are the operations that can be authorized by Clients presenting an access token to a RS
- The interpretation of such authorizations would result in a list of operations the bearer is allowed to perform
- Building on the SciTokens experience, define scopes that would match our computing use-cases

Authorization models in WLCG

Identity-based authorization: *wlcg.groups*

- When groups are asserted, the bearer has the access privileges corresponding to the VO's listed groups. It is up to the **RS to determine the mapping of the group names to the access privileges**
- Requests the `wlcg.group` scope to implement a group selection mechanism for groups equivalent to the one provided by VOMS, following the approach outlined in the OpenID Connect standard
  - "scopes can be used to request that specific sets of information be made available as Claim Value"
  - in WLCG, scopes are defined and mapped to claims that are returned in access tokens, ID tokens and results for userinfo endpoint and token introspection requests
- It results in a *wlcg.group* claim whose value is an ordered JSON array reflecting the VO groups of which the token subject is a member

Capability-based authorization for storage access

- **storage.read** Read data. Only applies to *online* resources such as disk (as opposed to *nearline* such as tape where the `storage.stage` authorization should be used in addition)
- **storage.create** Upload data. This includes renaming files if the destination file does not already exist. This authorization DOES NOT permit overwriting or deletion of stored data
- **storage.modify** Change data. This includes renaming files and writing data. This permission includes overwriting or replacing stored data in addition to deleting or truncating data
- **storage.stage** Cause data to be staged from a nearline resource to an online resource. This is a superset of `storage.read`

Capability-based authorization for storage access

Storage scopes additionally provide a resource path, which further limits the authorization

- The resource path follows the format **`$AUTHZ:$PATH`**
  - Example: `storage.read:/foo` provides a read authorization for the resource at `/foo` but not `/bar`
- The resource path may be `/` to authorize the entire resource associated with the issuer
  - Example: a token issued by the Atlas IAM and containing the `storage.modify:/` scope allows to write data in the entire Atlas namespace
- Following the Scitokens model, permissions granted on a path apply transitively to subpaths
  - Example: `storage.read:/cms` grants read access to the `/cms` directory and to all its content, but does not grant read access to the `/atlas` directory

Capability-based authorization for storage access

- This approach is **not equivalent** with POSIX semantics, but matches well with our experiments data access authorization models
    - For example, if a token contains the `storage.read:/home` scope, an implementation must override normal POSIX access control and leave the bearer to access all user's home directories
- Implementing this authorization is up to Client applications (*i.e.* StoRM WebDAV, dCache, *etc.*)

The token just provides a (signed) string!

Capability-based authorization for job submission

- **compute.read** “Read” or query information about a job status and attributes
- **compute.modify** Modify or change the attributes of an existing job
- **compute.create** Create or submit a new job at the computing resource
- **compute.cancel** Delete a job from the computing resource, potentially terminating a running job

Currently, they refer to all jobs owned by the issuer (*i.e.* a finer-grained path authorization is not foreseen).

For instance, a token with `compute.read` scope issued by https://cms-auth.web.cern.ch would be able to query the status of any CMS job at the resource

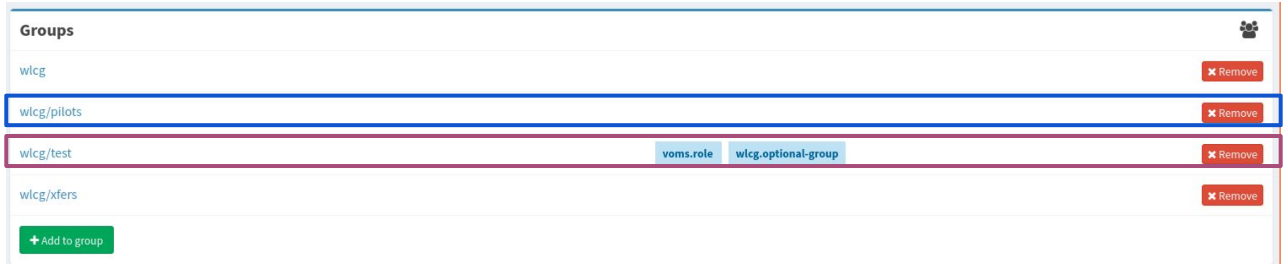
Identity-based authorization using groups

The `wlcg.group` scope is used to implement an attribute selection mechanism

In the WLCG JWT profile two types of groups have been defined

- **Default groups**, whose membership is always asserted (similar to *VOMS groups*)
- **Optional groups**, whose membership is asserted only when explicitly requested by the Client application (similar to *VOMS roles*)

Those groups appears in the access token when a user (*i.e.* the *sub* of an AT) delegates access to a Client application based on its attributes membership



Groups

wlcg

wlcg/pilots

wlcg/test voms.role wlcg.optional-group

wlcg/xfers

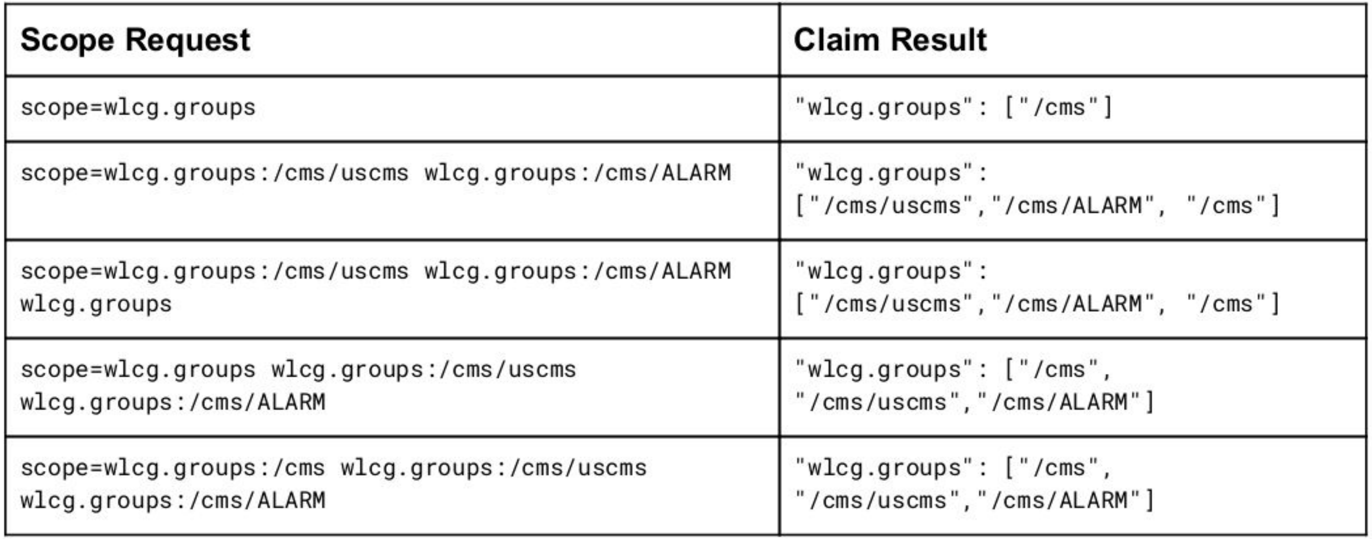
Identity-based authorization using groups

- A parametric `wlcg.groups` scope is introduced with the following form:
  `wlcg.groups[:<group-name>]`
- and the the following rules:
  - if the scope does not have the parametric part, *i.e.* its value is `wlcg.groups`, the authorization server will return the list of default groups for the user being authenticated as a value in the *wlcg.groups* claim
  - if the scope is parametric, (*i.e.* it has the form `wlcg.groups:<group-name>`), in addition to the default groups the authorization server will also return the requested group if the user is member of such group
  - the order of the groups in the returned *wlcg.groups* claim complies with the order in which the groups were requested
  - to request multiple groups, multiple `wlcg.groups:<group-name>` scopes are included in the authorization request
- This seems complex, but it's the attribute selection mechanism we use everyday with VOMS

Implementing this authorization is (mostly) up to the WLCG AuthZ server (*i.e.*, IAM)

Identity-based authorization using groups: example

In the following examples /cms is the only default group

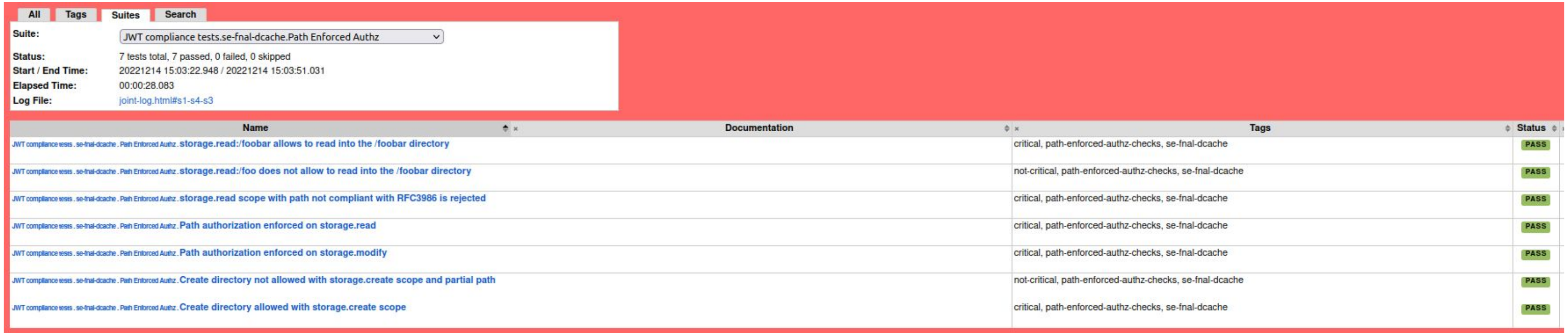


Scope Request	Claim Result
scope=wlcg.groups	"wlcg.groups": ["/cms"]
scope=wlcg.groups:/cms/uscms wlcg.groups:/cms/ALARM	"wlcg.groups": ["/cms/uscms","/cms/ALARM", "/cms"]
scope=wlcg.groups:/cms/uscms wlcg.groups:/cms/ALARM wlcg.groups	"wlcg.groups": ["/cms/uscms","/cms/ALARM", "/cms"]
scope=wlcg.groups wlcg.groups:/cms/uscms wlcg.groups:/cms/ALARM	"wlcg.groups": ["/cms", "/cms/uscms","/cms/ALARM"]
scope=wlcg.groups:/cms wlcg.groups:/cms/uscms wlcg.groups:/cms/ALARM	"wlcg.groups": ["/cms", "/cms/uscms","/cms/ALARM"]

WLCG JWT compliance testsuite

- A WLCG JWT compliance testsuite runs daily in order to check that implementation on the storage sites satisfies the WLCG JWT profile requirements
- **Capability-based** authorization with `storage.*:/[<path>]` scopes

Latest report here



Suite: JWT compliance tests.se-fnal-dcache.Path Enforced Authz

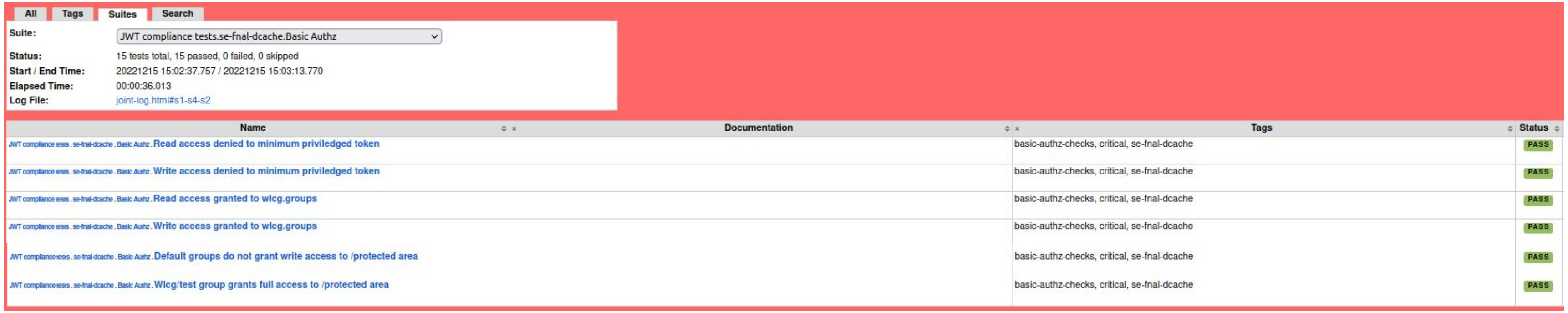
Status: 7 tests total, 7 passed, 0 failed, 0 skipped
Start / End Time: 20221214 15:03:22.948 / 20221214 15:03:51.031
Elapsed Time: 00:00:28.083
Log File: joint-log.html#s1-s4-s3

Name	Documentation	Tags	Status
JWT compliance tests . se-fnal-dcache . Path Enforced Authz . storage.read:/foobar allows to read into the /foobar directory		critical, path-enforced-authz-checks, se-fnal-dcache	PASS
JWT compliance tests . se-fnal-dcache . Path Enforced Authz . storage.read:/foo does not allow to read into the /foobar directory		not-critical, path-enforced-authz-checks, se-fnal-dcache	PASS
JWT compliance tests . se-fnal-dcache . Path Enforced Authz . storage.read scope with path not compliant with RFC3986 is rejected		critical, path-enforced-authz-checks, se-fnal-dcache	PASS
JWT compliance tests . se-fnal-dcache . Path Enforced Authz . Path authorization enforced on storage.read		critical, path-enforced-authz-checks, se-fnal-dcache	PASS
JWT compliance tests . se-fnal-dcache . Path Enforced Authz . Path authorization enforced on storage.modify		critical, path-enforced-authz-checks, se-fnal-dcache	PASS
JWT compliance tests . se-fnal-dcache . Path Enforced Authz . Create directory not allowed with storage.create scope and partial path		not-critical, path-enforced-authz-checks, se-fnal-dcache	PASS
JWT compliance tests . se-fnal-dcache . Path Enforced Authz . Create directory allowed with storage.create scope		critical, path-enforced-authz-checks, se-fnal-dcache	PASS

WLCG JWT compliance testsuite

- A WLCG JWT compliance testsuite runs daily in order to check that implementation on the storage sites satisfies the WLCG JWT profile requirements
- **Identity-based** authorization with `wlcg.groups[:<group-name>]` scopes

Latest report here



Suite: JWT compliance tests.se-fnal-dcache.Basic Authz

Status: 15 tests total, 15 passed, 0 failed, 0 skipped

Start / End Time: 20221215 15:02:37.757 / 20221215 15:03:13.770

Elapsed Time: 00:00:36.013

Log File: joint-log.html#s1-s4-s2

Name	Documentation	Tags	Status
JWT compliance tests . se-fnal-dcache . Basic Authz . Read access denied to minimum priviledged token		basic-authz-checks, critical, se-fnal-dcache	PASS
JWT compliance tests . se-fnal-dcache . Basic Authz . Write access denied to minimum priviledged token		basic-authz-checks, critical, se-fnal-dcache	PASS
JWT compliance tests . se-fnal-dcache . Basic Authz . Read access granted to wlcg.groups		basic-authz-checks, critical, se-fnal-dcache	PASS
JWT compliance tests . se-fnal-dcache . Basic Authz . Write access granted to wlcg.groups		basic-authz-checks, critical, se-fnal-dcache	PASS
JWT compliance tests . se-fnal-dcache . Basic Authz . Default groups do not grant write access to /protected area		basic-authz-checks, critical, se-fnal-dcache	PASS
JWT compliance tests . se-fnal-dcache . Basic Authz . Wlcg/test group grants full access to /protected area		basic-authz-checks, critical, se-fnal-dcache	PASS

WLCG JWT profile v1.1

- There is a draft for the next version of the WLCG JWT profile
- In particular:
  - definition of `wlcg.capability` scope/claim (PR 6, PR 10 & PR 14)
  - specify the hierarchical authorization based on sub-groups (PR 15)
  - clarify the authorization model when the capability and identity is asserted in the AT (PR 23)
  - improve authorization based on `storage.*` scopes (Issue 21)

Summary

- To access computing and storage resources in the WLCG today you use a VOMS proxy, which provides information about who you are, for which VO you're acting and what you can do on the infrastructure (*i.e.*, VOMS groups and roles)
- In the near future we will use **tokens**, which will provide more or less the same information
- Tokens are obtained from a VO token issuer (*e.g.*, IAM) using OpenID Connect
- Tokens are sent to services/resources following OAuth recommendations (*e.g.*, embedded in the header or an HTTP request)
- Tokens are self-contained, *i.e.* their integrity and validity can be verified locally with no callback to the token issuer

Useful references

RFC

- The OAuth 2.0 Authorization Framework (6749)
- JWT (7519)
- Bearer token usage (6750)
- OAuth 2.0 Device Authorization Grant (8628)
- Token exchange (8693)
- Proof Key for Code Exchange (7636)
- JWT for client authentication (7523)

Draft

- The OAuth 2.1 Authorization Framework
- OpenID Connect federation

IAM

- Source code (GitHub)
- IAM documentation
- Video in action

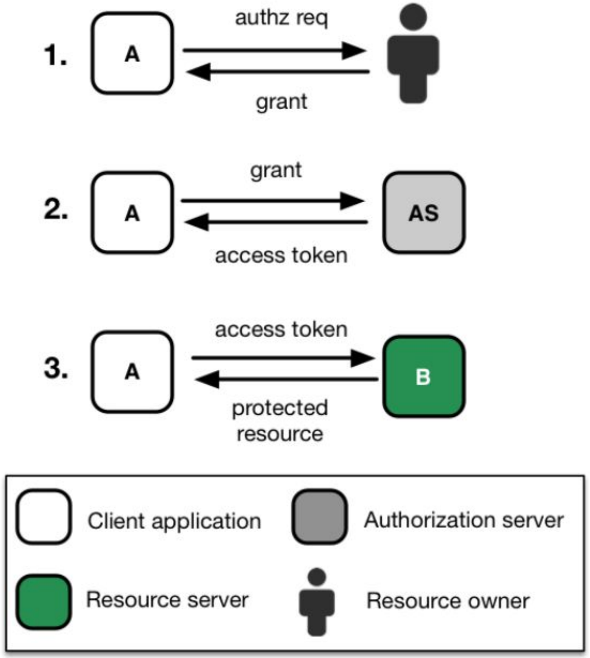
Other

- OpenID Connect 1.0
- OAuth 2.0 and OpenID Connect video (OktaDev)
- Apache integration demo
- INDIGO AAI tutorial (useful scripts to showcase the OAuth grant types)
- SAML
- oidc-agent documentation
- WLCG common JWT profiles

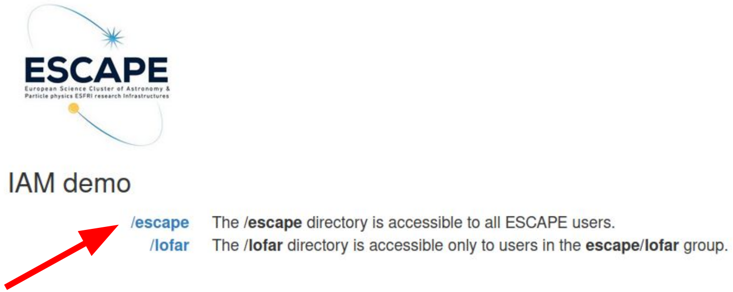
Bkp

Authorization flow in theory

1. Authorization request to the resource owner
    - The client (**A**) requests authorization from the resource owner to access a resource within a defined **scope**. The authorization request can be performed directly to the resource owner (as shown), or preferably indirectly via the authorization server (**AS**) as an intermediary
    - The client receives an authorization grant, which is a credential representing the resource owner's authorization, expressed using one of the authorization flows, or *grant types*. The authorization grant type depends on the method used by the client to request authorization from the authorization server
2. Authorization request to the AS token endpoint
    - The client requests an access token by authenticating with the authorization server and presenting the authorization grant
    - In this phase the client can obtain additional tokens (*e.g.* ID token, refresh token)



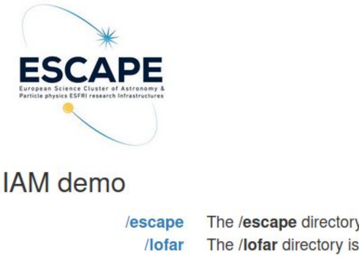
Demo application in action



IAM demo

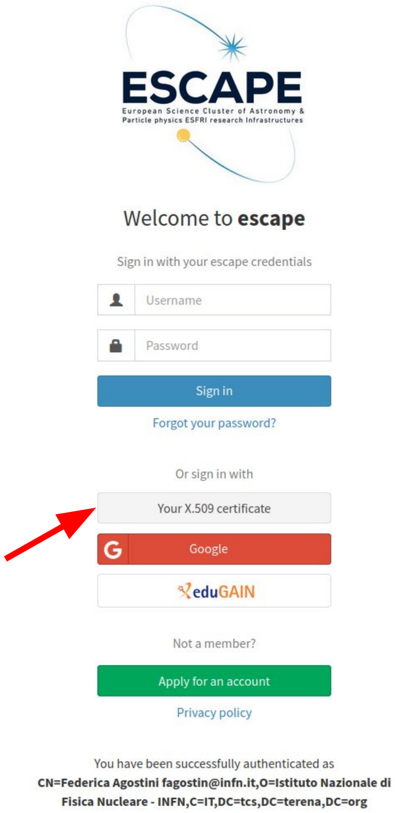
/escape	The /**escape** directory is accessible to all ESCAPE users.
/lofar	The /**lofar** directory is accessible only to users in the **escape/lofar** group.

Demo application in action



IAM demo

/escape	The /**escape** directory
/lofar	The /**lofar** directory is



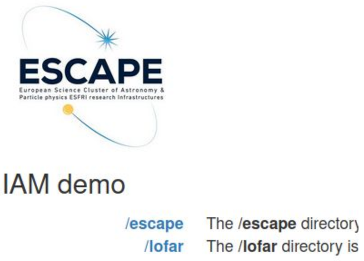
ESCAPE
European Science Cluster of Astronomy & Particle physics ESFRI research Infrastructures

Welcome to **escape**

Sign in with your escape credentials

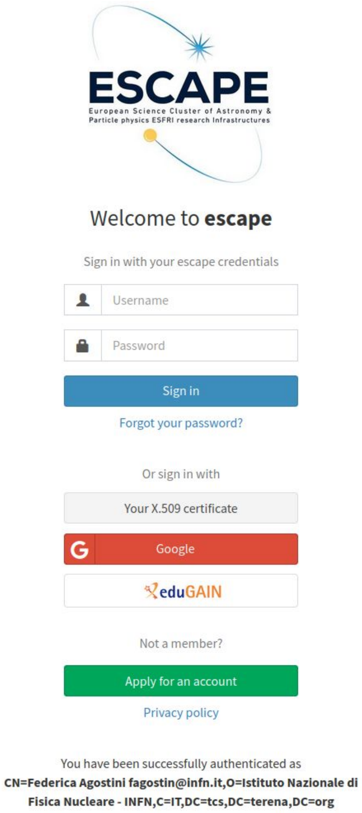
You have been successfully authenticated as **CN=Federica Agostini fagostin@infn.it,O=Istituto Nazionale di Fisica Nucleare - INFN,C=IT,DC=tcs,DC=terena,DC=org**

Demo application in action



IAM demo

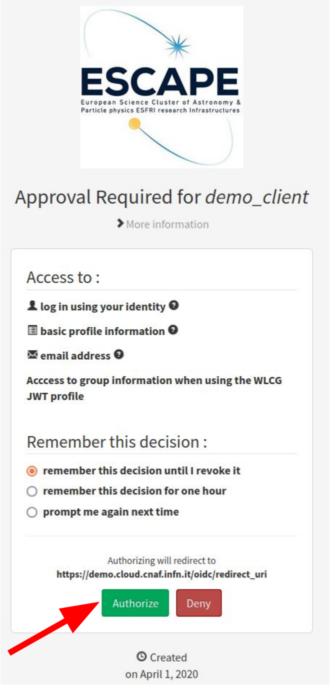
/escape	The /**escape** directory
/lofar	The /**lofar** directory is



Welcome to **escape**

Sign in with your escape credentials

You have been successfully authenticated as
CN=Federica Agostini fagostin@infn.it,O=Istituto Nazionale di Fisica Nucleare - INFN,C=IT,DC=tcs,DC=terena,DC=org



Approval Required for *demo_client*

Access to :

- **log in using your identity**
- **basic profile information**
- **email address**
- **Acccess to group information when using the WLCG JWT profile**

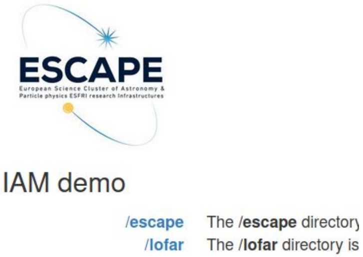
Remember this decision :

- ● **remember this decision until I revoke it**
- ○ **remember this decision for one hour**
- ○ **prompt me again next time**

Authorizing will redirect to
https://demo.cloud.cnaf.infn.it/oidc/redirect_uri

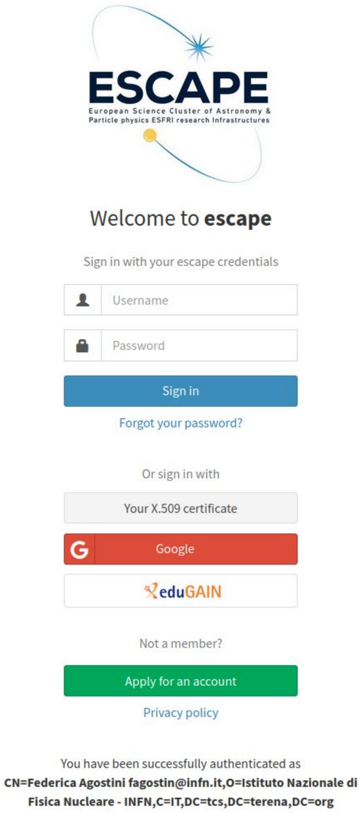
Created
on April 1, 2020

Demo application in action



IAM demo

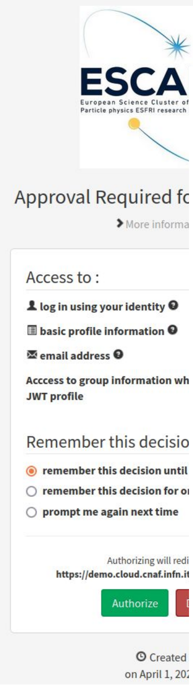
/escape	The /**escape** directory
/lofar	The /**lofar** directory is



Welcome to **escape**

Sign in with your escape credentials

You have been successfully authenticated as
CN=Federica Agostini fagostin@infn.it,O=Istituto Nazionale di Fisica Nucleare - INFN,C=IT,DC=tcs,DC=terena,DC=org



Approval Required fc

› More informa

Access to :

- **log in using your identity**
- **basic profile information**
- **email address**

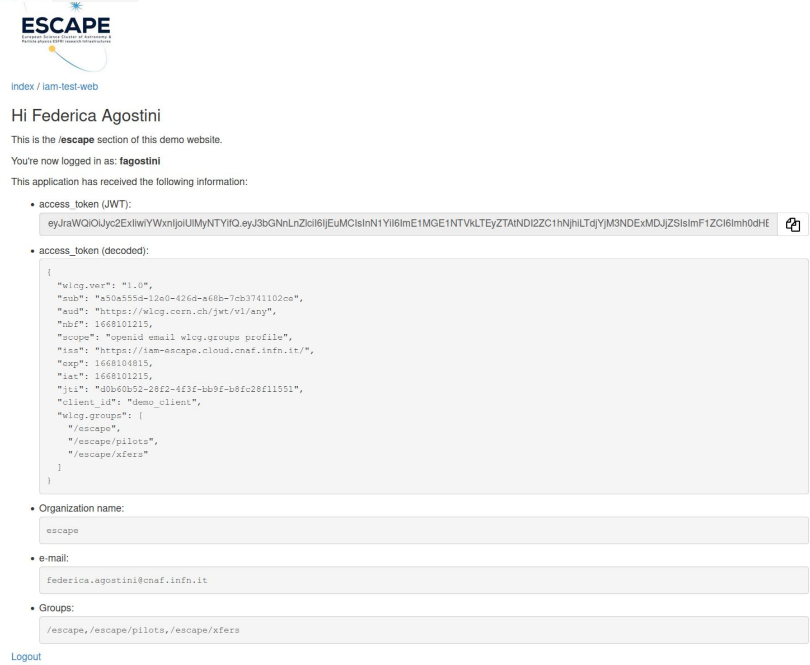
Acccess to group information wh
JWT profile

Remember this decisio

● **remember this decision until**
○ **remember this decision for o**
○ **prompt me again next time**

Authorizing will redi
https://demo.cloud.cnaf.infn.it

Created
on April 1, 202



index / iam-test-web

Hi Federica Agostini

This is the /**escape** section of this demo website.

You're now logged in as: **fagostini**

This application has received the following information:

- access_token (JWT):

```
eyJraWQiOiJyc2ExliwiYWxnIjoiUlMyNTYifQ.eyJ3bGNnLnZlcil6IjEuMClsInN1Yil6ImE1MGE1NTVkLTEyZTAtNDI2ZC1hNjhiLTdjYjM3NDExMDJjZSIsImF1ZCI6Imh0dHE
```

- access_token (decoded):

```
{
  "wlcg.ver": "1.0",
  "sub": "a50a555d-12e0-426d-a68b-7cb3741102ce",
  "aud": "https://wlcg.cern.ch/jwt/v1/any",
  "nbf": 1668101215,
  "scope": "openid email wlcg.groups profile",
  "iss": "https://iam-escape.cloud.cnaf.infn.it/",
  "exp": 1668104815,
  "iat": 1668101215,
  "jti": "d0b60b52-28f2-4f3f-bb9f-b8fc28f11551",
  "client_id": "demo_client",
  "wlcg.groups": [
    "/escape",
    "/escape/pilots",
    "/escape/xfers"
  ]
}
```

- Organization name:

```
escape
```

- e-mail:

```
federica.agostini@cnaf.infn.it
```

- Groups:

```
/escape,/escape/pilots,/escape/xfers
```

Logout