# An authentication solution for iRODS based on the OpenID Connect protocol

**Claudio Cacciari**
CINECA - Interuniversity Consortium
Casalecchio di Reno (BO) - Italy
c.cacciari@cineca.it

**Giuseppa Muscianisi**
CINECA - Interuniversity Consortium
Casalecchio di Reno (BO) - Italy
g.muscianisi@cineca.it

**Michele Carpené**
CINECA - Interuniversity Consortium
Casalecchio di Reno (BO) - Italy
m.carpen@cineca.it

**Mattia D'Antonio**
CINECA - Interuniversity Consortium
Casalecchio di Reno (BO) - Italy
m.dantonio@cineca.it

**Giuseppe Fiameni**
CINECA - Interuniversity Consortium
Casalecchio di Reno (BO) - Italy
g.fiameni@cineca.it

#### ABSTRACT

We are going to describe an authentication solution for iRODS based on the OpenID Connect (OIDC) protocol. In the context of European data infrastructures, like EUDAT, and projects, like EOSC-hub, iRODS must interoperate with other services, which support OIDC and OAuth2 protocols. The typical usage workflows encompass both direct user interaction via icommand and other clients and service-to-service interaction on behalf of the user. While in the first case we can rely on the already existing iRODS OpenID plugin, in the second one it is not possible because of two main reasons. The first is that the service-to-service process implies that the user is not requested to generate a token for iRODS, but that iRODS is able to re-use an existing token from another service. The second is that the other service needs to get access to iRODS using multiple authentication protocols in a dynamic way, not fixing one of them in the configuration. For example we have instances of DavRODS that allow to log-in via plain username and password or via OIDC token. In order to achieve those results we implemented a Pluggable Authentication Module (PAM), which allows iRODS to accept an OIDC token, re-using the password parameter of the PAM based authentication, validate it against an Authentication Service and map the user to a local account relying on the attributes provided back by the Authentication Service, once validated the token. Given the flexibility of the PAM approach, in this way it is possible to stack multiple PAM modules together, enabling a single iRODS instance to support multiple OIDC providers and even to create dynamically the local accounts, without any pre-configured mapping.

#### Keywords

OpenID Connect, B2ACCESS, B2SAFE, PAM, iRODS, authentication.

#### INTRODUCTION

IRODS was adopted years ago in the EUDAT CDI, Collaborative Data Infrastructure [1], as main component of the service for the long term data preservation and the policy driven data management, called B2SAFE [2]. Within the EUDAT CDI ecosystem the authentication is managed through an OpenID Connect [3] Server, called B2ACCESS. In order to support such protocol, we have developed a Pluggable Authentication Module (PAM) that enables iRODS to authenticate a user with an OIDC token and to map it to an iRODS's local account. The implemented mechanism relies on the PAM support of iRODS and assumes that the user has already obtained a valid token before logging in to iRODS (Figure 1). While no assumption is made about the corresponding local (to the iRODS system) account. It

*iRODS UGM 2019* June 25-28, 2019, Utrecht, Netherlands
[Copyright 2019 ©CINECA.]

can be created in advance by the administrator and explicitly mapped to the global user account (the one associated to the OIDC token). Or it can be dynamically created at the first login attempt through a shell script triggered at the end of the PAM modules' stack.

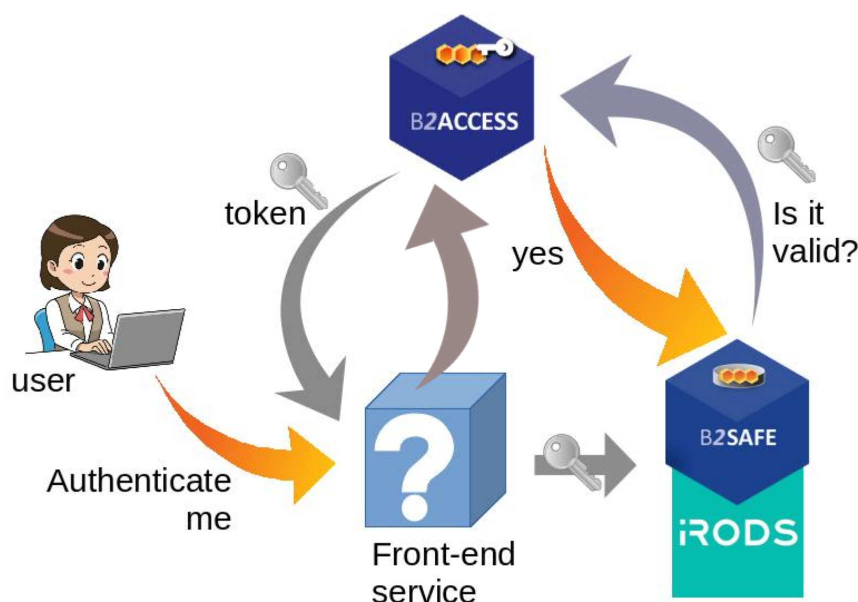


**Figure 1. Overview of the main authentication scenario.**

B2SAFE is implemented as a set of iRODS rules, python scripts and microservices and it is distributed as an optional package to be deployed on top of an existing iRODS instance.

In the next chapters we will explain which is the problem we have addressed, the details of the solution we have implemented, its benefits and limits and the perspective.

## THE AUTHENTICATION ISSUE

### The authentication flow

Our system, the EUDAT CDI, is composed by multiple distributed services, most of which belonging to different administrative domains. Each service works standalone, offering interfaces as entry points for the users, and, at the same time, can be the back-end of other services of the same system. Naturally we want to offer a user-friendly environment, therefore we adopted a federated identity approach, as defined by Gaedke, Johannes and Nussbaumer [4] and Chadwick [5], to support a single identity for the user across the whole infrastructure. There is a central authentication service (B2ACCESS), which acts both as OpenID Provider (OP) and as bridge for multiple Identity Providers (IdP). The typical authentication flow to get access to a service is listed below:

1. The user login to the wanted service.
2. The service redirect the user to the OP via OIDC protocol.
3. The user authenticates himself against the OP using the credentials and the identity associated to one of his IdPs.
4. The service receives a response from the OP and based on that, it allows the user to perform the required action and/or get access to the desired resource.

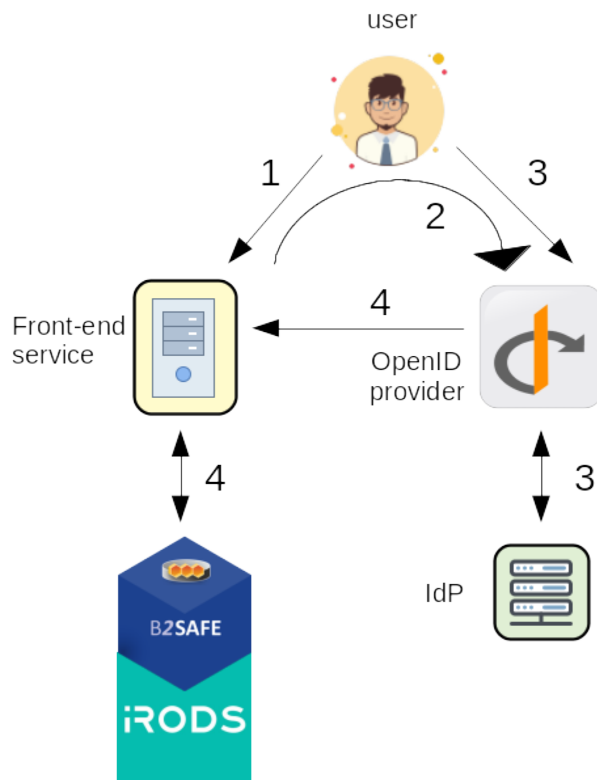


**Figure 2. Authentication flow to get access to B2SAFE service.**

In Figure 2 is shown the authentication flow to get access to the B2SAFE service.

We are assuming here to rely on the OIDC authorization code flow [6].

### The OIDC authorization code flow

We will now describe more in details the authorization code flow [7] ("OpenID Connect explained | Connect2id", n.d.):

1. The service initiates user authentication by redirecting the browser to the OAuth 2.0 authorization endpoint of OP.
2. At the OP, the user will typically be authenticated prompting the user to login. After that the user will be asked whether they agree to sign into the service.
3. The OP will then call the service's callback URI with an authorization code (on success) or an error code (if access was denied, or some other error occurred, such a malformed request was detected).
4. The service must use the code to proceed to the next step, exchanging the code for the ID token. The authorization code is an intermediate credential, which encodes the authorization. It is therefore opaque to the service and only has meaning to the OP server. To retrieve the ID token the service must submit the code to the OP. The code-for-token exchange happens at the token endpoint of the OP.
5. The service has been previously registered with the OP and has got a client ID and a secret. Both are passed via the Authorization header in the request to obtain the ID token. On success the OP will return a JSON object with the ID token, an access token and an optional refresh token.

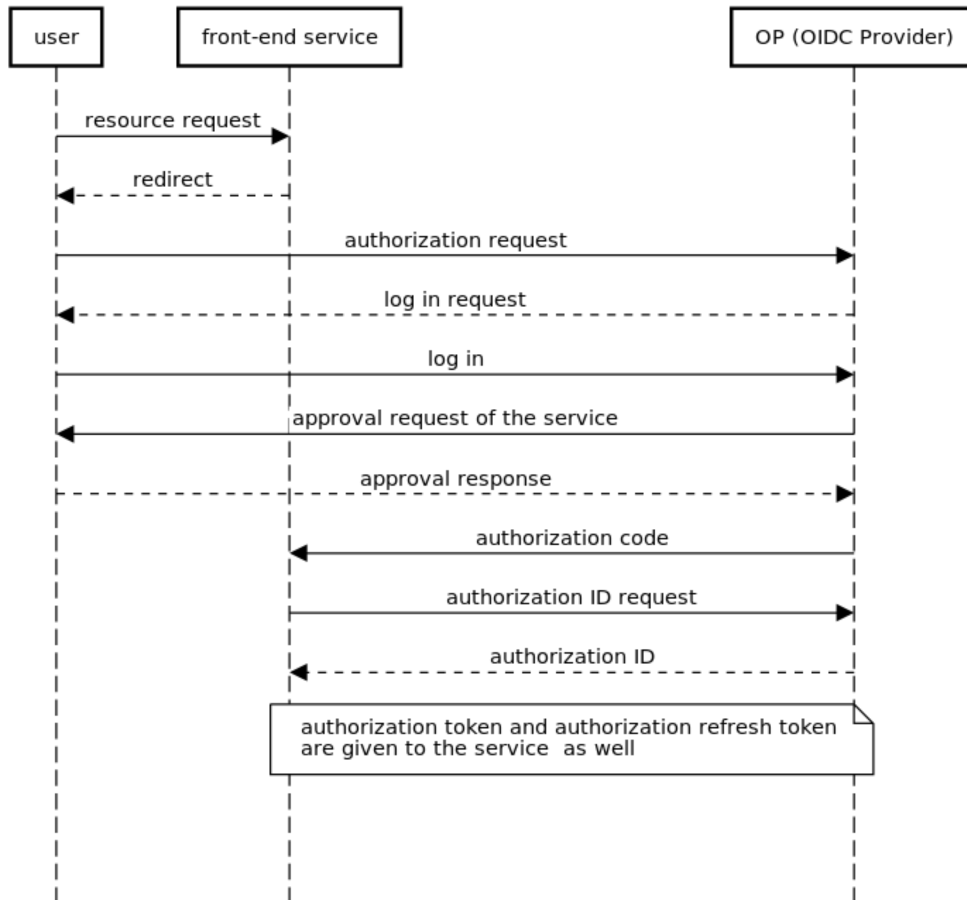


**Figure 3. Authorization code flow sequence diagram.**

### The issue

The aforementioned flow is supported by the services of our system. IRODS itself can be configured to support it, using the OpenID plugin [8]. However when we want to implement a workflow chaining together a first service, that we will call front-end, and B2SAFE as back-end, we have troubles. Usually the user wants to have access to B2SAFE, and hence to iRODS, because she wants to download data from or upload data to it. We assume that the user logins into the front-end service using the OIDC protocol and her federated identity. At this point we need that the user gets access to the B2SAFE service with the same identity, which has to be mapped to an iRODS local account. How the front-end service can pass the user identity credentials to B2SAFE so that it can authenticate the user? We could ask the user to login explicitly to B2SAFE, but this is exactly what we want to avoid. It would be possible to use a third software component, like a Web portal or a workflow manager, which would impersonate the user in front of the services, but they are not available in our system. In fact the services are connected directly to each other through their APIs. In the next chapter we will describe the proposed solution.

## THE PROPOSED SOLUTION

After that the user logs in successfully into the front-end service, this last one receives the ID token, the access token and the refresh token. Therefore our proposal is to rely on those tokens to authenticate the user against B2SAFE and in particular on the access token. In order to authenticate the user, B2SAFE relies on iRODS's authentication

mechanisms. But they do not support OIDC tokens as authentication credentials, thus we need to extend them. In particular we need to satisfy two requirements:

1. The user identity credentials must be validated
2. The user federated identity must be mapped to an iRODS's local account.

The first point can only be achieved presenting the access token to the OIDC provider, B2ACCESS in our case, which has issued the token. OpenID Connect specifies a set of standard claims, or user attributes. They are intended to supply the client with consented user details such as email, name and picture, upon request. The OIDC provider's UserInfo endpoint returns them. Submitting a request containing the access token to the UserInfo endpoint, it is possible to validate the token and to get some user attributes. In our case, B2ACCESS can provide us the email of the user. Assuming the email address is unique for each user, we can use it to map the federated identity of the user to an iRODS local account and satisfy in this way the second requirement.

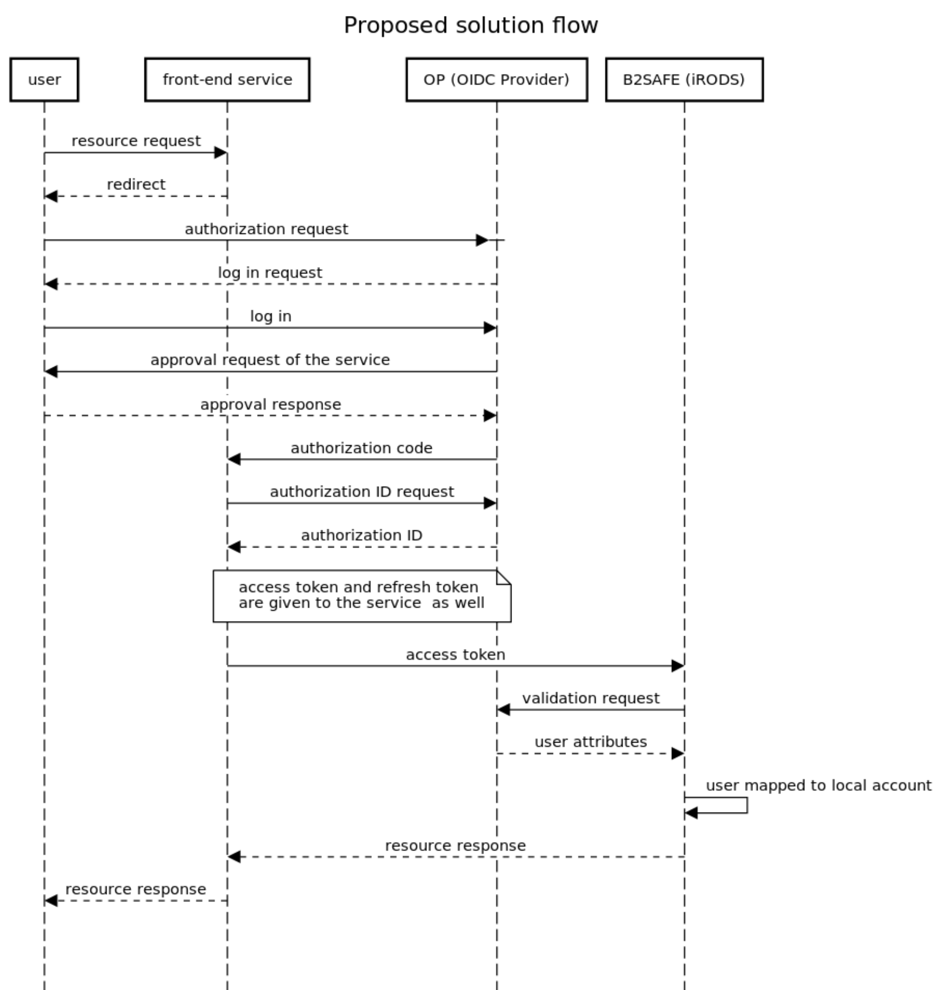


**Figure 4. New proposed authorization code flow sequence diagram.**

### Implementation

We have implemented the proposed solution as a PAM module [9], written in C, which needs to be compiled and configured. The choice of PAM as framework permits to have more flexibility compared to other solutions, like

implementing a new iRODS authentication plugin, as we will show in the next paragraphs. To enable the new module, the iRODS PAM configuration file (/etc/pam.d/irods) has to include the following line:

```
auth sufficient pam_oauth2.so <path to configuration file> key1=value2 key2=value2
```

Where pam_oauth2.so is the name of the compiled new PAM module and key1 and key2 optional parameters. Assuming the path to the configuration file is /etc/irods/pam.conf, its content looks like this:

```
#OAuth2 url for token validation
token_validation_ep = "https://b2access.eudat.eu/oauth2/userinfo"
#OIDC attribute key to identify the attribute in the response used to match the login username
login_field = "email"
# path to user map file
user_map_file = "/etc/irods/user_map.json"
```

Where:
**token_validation_ep**: it is the OIDC UserInfo endpoint.
**login_field**: it is the attribute that needs to be checked for the user mapping.
**user_map_file**: it is the path to the file defining the mapping rules.
The user_map_file is formatted as a json document and looks like this:

```
{
  "roberto": ["roberto@email.it", "r.mucci@email.it"],
  "claudio": ["c.cacc@email.com", "claudio@email.it", "c.cacciari@email.it"],
  "paolo": ["paolo@email.com"]
}
```

A list of email addresses, on the right, can be mapped to a single iRODS account, on the left.

### How it works

Since we have decided to rely on the iRODS PAM mechanism, the user, or better, in our case, the front-end service has to login with the PAM authentication method, but instead of the password of the iRODS local account, it uses the access token. The PAM module pam_oauth2.so receives the token and issues a request to the B2ACCESS's token_validation_ep. If the token is valid, the response looks like this:

```
{
  "email":"roberto@email.com",
  "token_type":"Bearer",
  "exp":1520001942,
  "iat":1519998342,
  [ ... ]
}
```

And the front-end service will get access as the local user "roberto". Enabling the debug log of the PAM framework,

the following lines will appear in the syslog:

```
Token validation EP: https://b2access.eudat.eu/oauth2/userinfo
Jun  1 09:26:34 localhost irodsPamAuthCheck: username_attribute: email
Jun  1 09:26:34 localhost irodsPamAuthCheck: user_map_path: /etc/irods/user_map.json
Jun  1 09:26:34 localhost irodsPamAuthCheck: Searching for user: roberto
Jun  1 09:26:34 localhost irodsPamAuthCheck: Found local mapping item: roberto@email.it
Jun  1 09:26:34 localhost irodsPamAuthCheck: Found local mapping item: r.mucci@email.it
Jun  1 09:26:34 localhost irodsPamAuthCheck: pam_oauth2: Found user mapping array for user
                                                         roberto
Jun  1 09:26:34 localhost irodsPamAuthCheck: pam_oauth2: user mapping item returned by B2ACCESS:
                                                         roberto@email.com
Jun  1 09:26:34 localhost irodsPamAuthCheck: pam_oauth2: succesfully mapped item to local
                                                         iRODS user
Jun  1 09:26:34 localhost irodsPamAuthCheck: pam_oauth2: successfully authenticated by B2ACCESS
```

## BENEFITS AND LIMITS

The proposed solution solves the aforementioned authentication issue because it allows to reuse the OIDC tokens without requiring the user to login again when the B2SAFE service is involved as back-end in a workflow. The front-end service can access B2SAFE using the same federated identity of the user and iRODS is able to validate the OIDC access token associated to that identity and to map it to a local account.

Another benefit is the flexibility in the account mapping. Thanks to the pam_exec, a PAM module that can be used to run an external command, it is possible, after a successful authentication, to execute a script which calls the iRODS command to create a user on the fly. Therefore it is possible to create users with the same username of the federated identity or just implement a pool of accounts which are allocated dynamically and so on.
Moreover this approach allows to support multiple OIDC providers at the same time, stacking multiple modules pam_oauth2.so in the same file /etc/pam.d/irods, each one pointing to a different configuration file. Indeed this is true for multiple PAM modules in general. For example it is possible to stack together pam_oauth2.so and the LDAP module, so that if one authentication method fails, the credentials are passed to the next one.
One limit of the described solution is the need to know the local iRODS username at the login time because this is required by the iRODS PAM authentication mechanism. This reduces the flexibility of the user mapping procedure. It is still possible to create dynamically a user account, but the username must be predictable.
More important is the fact that the access token has a limited lifetime, which is, in our case, set to one hour. The OIDC protocol includes a refresh token to allow the client, the front-end service in our case, to extend the lifetime when needed. However iRODS cannot do it because the lifetime can be extended only by the same client that has requested it the first time. Therefore the front-end service must support the refreshing of the tokens, possibly in a transparent way for the user.

## USE CASES

The solution has been tested with two front-end services in the context of the EOSC-hub project [10]. The first one is an HTTP interface [11], which exposes some functions to upload/download data using the iRODS python library [12]. The HTTP API serves users that have a federated identity and users that have just a local identity, from the same instance. This is possible thanks to the flexibility of the PAM module.
The second one is a data management tool called DataHub [13], which is able to mount external storage if this storage is exposed through a WebDAV interface. In this case the workflow is composed by three services because DataHub is connected to the WebDAV interface [14], which is deployed in front of iRODS. DataHub takes care of the initial user authentication and hence of the token refreshing.

## CONCLUSION

We have described the implementation of a solution to add the support of the OpenID Connect protocol to iRODS relying on its PAM authentication mechanism. The solution is flexible enough to enable different use cases and satisfies the requirements of the single identity of the user and of the validation of the OIDC credentials. Future developments can be envisioned about the user mapping mechanism. Currently it is just based on a static map represented as a json document. However that approach does not scale very well. It would be better to map the user in dynamic way based on the attributes. For example supporting the use of regular expressions to define matching rules, we could map all the users belonging to a certain organization to a single iRODS local user or group. In this way, even if new members join the organization, it would not be necessary to add them manually to the user map, like we do now.

## ACKNOWLEDGMENTS

The implementation of the PAM module for the OIDC protocol started forking the code of the OAuth2 PAM module by Alexander Kukushkin [15].

## REFERENCES

[1] EUDAT CDI, Collaborative Data Infrastructure, https://www.eudat.eu

[2] B2SAFE service, https://www.eudat.eu/b2safe

[3] OpenID connect, https://openid.net/connect

[4] Gaedke, M., Meinecke, J., Nussbaumer, M.: A modeling approach to federated identity and access management. In: Special interest tracks and posters of the 14th international conference on World Wide Web - WWW '05. ACM Press. https://doi.org/10.1145/1062745.1062916 (2005)

[5] Chadwick, D. W.: Federated Identity Management. In: Foundations of Security Analysis and Design V, pp. 96-120. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-03829-7_3 (2009)

[6] OpenID authorization code flow, https://openid.net/specs/openid-connect-core-1_0.html

[7] OpenID connect, https://connect2id.com/learn/openid-connect

[8] OpenID iRODS plugin, https://github.com/irods-contrib/irods_auth_plugin_openid

[9] PAM module developed, https://github.com/EUDAT-B2SAFE/pam-oauth2

[10] EOSC-hub project, https://www.eosc-hub.eu

[11] HTTP interface, https://github.com/EUDAT-B2STAGE/http-api

[12] iRODS python library, https://github.com/irods/python-irodsclient

[13] DataHub, https://www.eosc-hub.eu/services/EGI%20DataHub

[14] DavRODS - WebDAV interface, https://github.com/UtrechtUniversity/davrods

[15] OAuth2 PAM module, https://github.com/CyberDem0n/pam-oauth2