# INTRODUCTION

## INFORMATION FROM THE TECHNICAL EVALUATION REPORT

|  |  |
| --- | --- |
| **Reference component** | IDSA Reference Testbed |
| **Version** | 1.0 |
| **Author or authors** | Fraunhofer AISEC, Fraunhofer ISST and Fraunhofer IAIS |
| **Approved by** |  |
| **Date** |  |
| **File code** |  |
| **Type of evaluation** |  |
| **Taxonomy of the product** |  |

## DEVELOPER AND TOE INFORMATION

|  |  |
| --- | --- |
| **Developer Data (Name and Address)** |  |
| **Developer Contact Details (Name and e-mail)** |  |
| **Name of the TOE** |  |
| **Version of the TOE** |  |

# DECRIPTION OF THE TOE

The reference testbed is currently comprised of the three base components required to make a minimum viable interoperability infrastructure. These components are a Connector, a Dynamic Attribute Provisioning Service (DAPS) and a Broker. The Connector is the central technical component of the International Data Spaces and it is capable of fulfilling both data provider and data consumer roles.

## FUNCTIONAL DESCRIPTION OF THE TOE

## INVENTORY OF SECURITY FUNCTIONS IDENTIFIED IN THE SECURITY STATEMENT

Integrity checks, encryption, PKI… etc

# EXECUTION ENVIRONMENT

## DESCRIPTION OF THE EXECUTION ENVIRONMENT

The reference testbed is being tested in Linux, Ubuntu 20.04. To launch the different components included we will need different kinds of software.

The Dataspace Connector makes use of Git, Maven (tested on 3.6.3) and requires at least Java 11.

The Metadata Broker makes use of Git, Docker, Docker compose and OpenSSL.

The Omejdn DAPS makes use of Git, Docker and Ruby.

To be able to successfully install the testbed the user requires a system with:

Linux OS: Ubuntu 20.04  
Java 11 (at least)  
Maven  
Git  
Docker  
Docker Compose  
Ruby  
OpenSSL

## HIPÓTESIS SOBRE EL ENTORNO DE EJECUCIÓN

# EXECUTIVE SUMMARY OF THE EVALUATION

The content expected in this section of the validation report is a summary of the evaluation that is going to be carried out. In this part it is highlighted the most important aspects and the aim is to provide an overview of how the assessment is going to be developed.

# EVALUATION VERDICT

In this section the evaluator assigns in the Evaluation Technical Report a final verdict for the evaluation. The possible verdict results of every test case executed are the following:

a) **OK**: The security functionality of the TOE complies with the Security Statement and the test case resolution is satisfactory and matches the expected result of the test performed.

b) **FAIL**: The security functionality of the TOE does not comply with the provisions of the Security Declaration and / or the test case resolution is not satisfactory, not matching the expected result of the test performed.

# COMPONENT INSTALLATION

The Testbed installation is broken down into two steps:  
1. The components are individually installed (Dataspace Connector, Metadata Broker, Omejdn DAPS)  
2. The components are configured to interact with each other in a closed environment

**Dataspace Connector**

At first, clone the repository:

git clone https://github.com/International-Data-Spaces-Association/DataspaceConnector.git

The resource folder resources/conf provides three important files that are loaded at application start:

* keystore-localhost.p12: The provided keystore, on the one hand, is used as IDS certificate that is loaded by the IDS Messaging Services for requesting a valid [Dynamic Attribute Token](https://github.com/International-Data-Spaces-Association/IDS-G/blob/master/core/DAPS/README.md#dynamic-attribute-token-content) (DAT) from the [Dynamic Attribute Provisioning Service](https://github.com/International-Data-Spaces-Association/IDS-G/blob/master/core/DAPS/README.md) (DAPS). Each message to an IDS participant needs to be signed with a valid [DAT](https://github.com/International-Data-Spaces-Association/IDS-G/blob/master/core/DAPS/README.md#dynamic-attribute-token-content). On the other hand, it can be used as SSL certificate for TLS encryption.
* truststore.p12: The truststore is used by the IDS Messaging Services for any HTTP/S communication. It ensures the connection to trusted addresses.
* config.json: The configuration is used to set important properties for IDS message handling.

## **Step 1: Connector Properties**

When starting the application, the config.json will be scanned for important connector information, e.g. its ID, address, contact information, or proxy settings. Please keep this file up to date to your custom settings. In case you want to use the demo cert, you don’t need to change anything except the proxy settings.

For outgoing requests, the connector needs information about an existing system proxy that needs to be set in the resources/conf/config.json.

"ids:connectorProxy" : [ {

"@type" : "ids:Proxy",

"@id" : "https://w3id.org/idsa/autogen/proxy/548dc73a-ccfb-4039-9569-4b8e219b90bc",

"ids:proxyAuthentication" : {

"@type" : "ids:BasicAuthentication",

"@id" : "https://w3id.org/idsa/autogen/basicAuthentication/47e3cd59-d351-4f5b-99fc-561c94bad5e1"

},

"ids:proxyURI" : {

"@id" : "http://host:port"

},

"ids:noProxy" : [ {

"@id" : "https://localhost:8080/"

}, {

"@id" : "http://localhost:8080/"

} ]

} ]

Check if your system is running behind a proxy. If this is the case, specify the ids:proxyURI and change ids:noProxy if necessary. Otherwise, delete the key ids:connectorProxy and its values.

A full configuration example may look like this:

{

"@context" : {

"ids" : "https://w3id.org/idsa/core/",

"idsc" : "https://w3id.org/idsa/code/"

},

"@type" : "ids:ConfigurationModel",

"@id" : "https://w3id.org/idsa/autogen/configurationModel/7672b568-7878-4f62-8032-5c73de969414",

"ids:configurationModelLogLevel" : {

"@id" : "idsc:MINIMAL\_LOGGING"

},

"ids:connectorDeployMode" : {

"@id" : "idsc:TEST\_DEPLOYMENT"

},

"ids:connectorDescription" : {

"@type" : "ids:BaseConnector",

"@id" : "https://w3id.org/idsa/autogen/baseConnector/7b934432-a85e-41c5-9f65-669219dde4ea",

"ids:publicKey" : {

"@type" : "ids:PublicKey",

"@id" : "https://w3id.org/idsa/autogen/publicKey/78eb73a3-3a2a-4626-a0ff-631ab50a00f9",

"ids:keyType" : {

"@id" : "idsc:RSA"

},

"ids:keyValue" : "[...]"

},

"ids:description" : [ {

"@value" : "IDS Connector with static example resources hosted by the Fraunhofer ISST",

"@type" : "http://www.w3.org/2001/XMLSchema#string"

} ],

"ids:version" : "1.0",

"ids:hasDefaultEndpoint" : {

"@type" : "ids:ConnectorEndpoint",

"@id" : "https://w3id.org/idsa/autogen/connectorEndpoint/e5e2ab04-633a-44b9-87d9-a097ae6da3cf",

"ids:accessURL" : {

"@id" : "https://localhost:8080/api/ids/data"

}

},

"ids:outboundModelVersion" : "4.0.4",

"ids:inboundModelVersion" : [ "4.0.0", "4.0.4" ],

"ids:title" : [ {

"@value" : "Dataspace Connector",

"@type" : "http://www.w3.org/2001/XMLSchema#string"

} ],

"ids:securityProfile" : {

"@id" : "idsc:BASE\_SECURITY\_PROFILE"

},

"ids:curator" : {

"@id" : "https://www.isst.fraunhofer.de/"

},

"ids:maintainer" : {

"@id" : "https://www.isst.fraunhofer.de/"

}

},

"ids:trustStore" : {

"@id" : "file:///conf/truststore.p12"

},

"ids:connectorStatus" : {

"@id" : "idsc:CONNECTOR\_ONLINE"

},

"ids:keyStore" : {

"@id" : "file:///conf/keystore.p12"

},

"ids:connectorProxy" : [ {

"@type" : "ids:Proxy",

"@id" : "https://w3id.org/idsa/autogen/proxy/548dc73a-ccfb-4039-9569-4b8e219b90bc",

"ids:proxyAuthentication" : {

"@type" : "ids:BasicAuthentication",

"@id" : "https://w3id.org/idsa/autogen/basicAuthentication/47e3cd59-d351-4f5b-99fc-561c94bad5e1"

},

"ids:proxyURI" : {

"@id" : "http://proxy.dortmund.isst.fraunhofer.de:3128"

},

"ids:noProxy" : [ {

"@id" : "https://localhost:8080/"

}, {

"@id" : "http://localhost:8080/"

}, {

"@id" : "https://localhost:8081/"

}, {

"@id" : "http://localhost:8081/"

} ]

} ]

}

**Note**: If you are not familiar with the IDS Information Model, the API provides an endpoint GET /api/examples/configuration to print a filled in Java object as JSON-LD. Adapt this to your needs, take the received string and place it in the config.json.

If you want to connect to a running connector or any other system running at https://, keep in mind that you need to add the keystore to your truststore. Otherwise, the communication will fail. With the provided truststore, the Dataspace Connector accepts its own localhost certificate, public certificates, and any IDS keystore that was provided by the Fraunhofer AISEC.

## **Step 2: IDS Certificate**

In the provided config.json, the ids:connectorDeployMode is set to idsc:TEST\_DEPLOYMENT. This allows to use the keystore-localhost.p12 as an IDS certificate. For testing purpose, the existing cert can be used, as on application start, the IDS Messaging Services will not get a valid [DAT](https://github.com/International-Data-Spaces-Association/IDS-G/blob/master/core/DAPS/README.md#dynamic-attribute-token-content) from the [DAPS](https://github.com/International-Data-Spaces-Association/IDS-G/blob/master/core/DAPS/README.md) and for received messages, the sent [DAT](https://github.com/International-Data-Spaces-Association/IDS-G/blob/master/core/DAPS/README.md#dynamic-attribute-token-content) will not be checked.

To turn on the [DAT](https://github.com/International-Data-Spaces-Association/IDS-G/blob/master/core/DAPS/README.md#dynamic-attribute-token-content) checking, you need to set the ids:connectorDeployMode to idsc:PRODUCTIVE\_DEPLOYMENT. For getting a trusted certificate, contact [Gerd Brost](mailto:gerd.brost@aisec.fraunhofer.de). Add the keystore with the IDS certificate inside to the resources/conf and change the filename at ids:keyStore accordingly. In addition, set your connector id to uniquely identify your connector towards e.g. the IDS Metadata Broker:

"ids:connectorDescription" : {

"@type" : "ids:BaseConnector",

"@id" : "CONNECTOR\_URL",

**Note**: The TEST\_DEPLOYMENT mode and accepting a demo cert is for testing purposes only! This mode is a **security risk** and cannot ensure that the connector is talking to a verified IDS participant. Furthermore, messages from the Dataspace Connector without a valid IDS certificate may not be accepted by other Connector implementations and will not be accepted by the IDS Metadata Broker running in the IDS lab.

## **Step 3: General Settings (optional)**

The application.properties specifies several Spring Boot and IDS configurations.

### **Tomcat**

To define on which port the connector should be running, change server.port={PORT}.

### **OpenApi**

You can change Swagger properties by changing the following settings:

springdoc.swagger-ui.path=/api/docs

springdoc.swagger-ui.operationsSorter=alpha

springdoc.swagger-ui.disable-swagger-default-url=true

### **SSL**

If you want to add your own SSL certificate, check the corresponding path. As the provided certificate only supports the application running at localhost, you may replace this with your IDS keystore, if you want to host the connector in a productive environment.

server.ssl.enabled

server.ssl.key-store-type

server.ssl.key-store

server.ssl.key-store-password

server.ssl.key-alias

configuration.path

configuration.keyStorePassword

configuration.keyAlias

configuration.trustStorePassword

### **Http Connections**

For customizing timeout settings for incoming and outgoing requests, you may customize the following lines:

http.timeout.connect=10000

http.timeout.read=10000

http.timeout.write=10000

http.timeout.call=10000

Not that either the call timeout is used, or the other three values.

### **Authentication**

The application uses Spring Security. Each endpoint behind /\*\*, needs a user authentication, except the open IDS endpoint at /api/ids/data.

Have a look at the blocked endpoints in the ConfigurationAdapter class to add or change endpoints yourself. In case you don’t want to provide authentication for your backend maintenance, feel free to remove the corresponding lines.

For changing the default credentials, the properties are located at spring.security.user.name and spring.security.user.password.

### **Database**

The Dataspace Connector uses Spring Data JPA to set up the database and manage interactions with it. Spring Data JPA supports many well-known relational databases out of the box. Thus, the internal H2 can be replaced by e.g. MySQL, PostgreSQL, or Oracle databases with minimal effort.

To use another database for the Dataspace Connector, follow [these](https://international-data-spaces-association.github.io/DataspaceConnector/Deployment/Database) steps.

Settings are provided within the application.properties at:

spring.datasource.url

spring.datasource.driverClassName

spring.datasource.username

spring.datasource.password

spring.h2.console.enabled=false

spring.h2.console.path=/database

spring.h2.console.settings.web-allow-others=true

### **Logging**

The Dataspace Connector provides multiple ways for logging and accessing information. Please find a detailed description on how to set up static and runtime configurations [here](https://international-data-spaces-association.github.io/DataspaceConnector/Deployment/Logging).

Settings are provided within the application.properties at:

management.endpoints.enabled-by-default=false

management.endpoints.web.exposure.include=logfile, loggers

management.endpoint.loggers.enabled=true

management.endpoint.logfile.enabled=true

management.endpoint.logfile.external-file=./log/dataspaceconnector.log

Http tracing is disabled by default: httptrace.enabled=false.

### **Jaeger**

If your want to access open telemetry, have a look at [this guide](https://international-data-spaces-association.github.io/DataspaceConnector/Deployment/Build#docker). You can customize the deployment with these lines:

opentracing.jaeger.udp-sender.host=localhost

opentracing.jaeger.udp-sender.port=6831

opentracing.jaeger.log-spans=true

### **Bootstrapping**

If you want to change the base path, which will be used to find properties and catalogs for bootstrapping, you can customize the following line:

bootstrap.path=.

### **IDS Settings**

URLs of the [DAPS](https://github.com/International-Data-Spaces-Association/IDS-G/blob/master/core/DAPS/README.md) for IDS identity management and the Clearing House for contract agreement and data usage logging can be changed within the following lines:

daps.token.url=https://daps.aisec.fraunhofer.de

daps.key.url=https://daps.aisec.fraunhofer.de/v2/.well-known/jwks.json

clearing.house.url=https://ch-ids.aisec.fraunhofer.de/logs/messages/

If you leave the Clearing House address blank, the connector will ignore sending IDS messages to it.

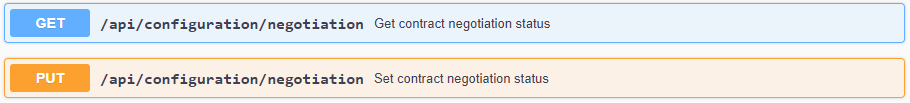
Also, for usage control, some settings are provided:

policy.negotiation=true

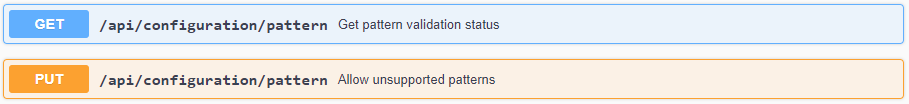
policy.allow-unsupported-patterns=false

policy.framework=INTERNAL

Contract negotiation is enabled by default. This forces other Connectors to refer to a valid contract agreement when requesting data access via an ArtifactRequestMessage. If you want to deactivate the policy negotiation, as data provider or data consumer, use the following endpoints or the corresponding line within the application.properties.



Note that the Dataspace Connector is able to received resources with usage policies that follow the IDS policy language but not one of the supported patterns. As, by default, the policy check on the data consumer side would not allow accessing data whose policies cannot be enforced, you are able to ignore unsupported patterns with setting the boolean at the endpoint /api/configuration/pattern or the property policy.allow-unsupported-patterns in the application.properties to true. As a data consumer, you are bound to concluded contract agreements that are technically mapped to IDS usage policies. Therefore, you have to ensure, that your backend applications technically enforce the usage policies instead.



**Metadata Broker**

### **Installation Guide**

The evaluator is in charge of verifying that the documentation is correct in terms of the installation and implementation of the component. Including the configuration required of the TOE so that for both Linux and Windows system it is specified in a complete way the steps to be performed by the user.

This document aims to aid IT administrators or developers in the installation of the IDS Metadata Broker.

Note: this guide works with provided images and is not targeted for development purposes. Thus, instructions for building and editing the docker image file will not be included here.

### **1 Prerequisites**

In this section, it is provided some guidance as to recommendations for the number of resources that should be available to smoothly operate the IDS Metadata Broker. The actual number of resources required heavily depends on the load. In case of very little traffic, fewer resources than listed below might be required.

#### **1.1 Hardware**

It is required 2GB of disk space for operating the IDS Metadata Broker, though it is recommended providing at least 20GB of free disk storage to avoid running out of disk space with increasing number of registered items.

To provide enough processing power for all Docker containers, it is recommended using a 64bit quad core processor or better.

#### **1.2 Software**

* **OS**: it is recommended using a Linux based operating system. However, any operating system with a Docker installation can be used (tested on Ubuntu 20.04 and Windows 10). More strict hardware requirements than listed above might apply if a non-Linux operating system is used.
* **Git**: it is used git version 2.25.1 for the installation of this component. However, any version of git higher than version 2.0.0 is sufficient to perform this installation.
* **Docker: it is used Docker version 19.03.8 for the installation of this component. It is recommended to have a docker version equal to or higher than the one mentioned for this installation.**
* **Docker compose: it is used Docker version 1.25.0 for the installation of this component. It is recommended to have a docker-compose version equal to or higher than the one mentioned for this installation.**
* **OpenSSL**: A valid X.509 certificate, signed by a trusted certification authority, it is strongly recommended to avoid warnings about insecure HTTPS connections.

Docker must be installed on the target machine. For the SSL certificate you need to have these two files:

-- **server.crt:** an x509 certificate, either self-signed or from an official CA.

-- **server.key:** the private key for the certificate. The certificate needs to be of .crt format and must have the name server.crt and the file for private key should have the name server.key.

In case your certificate is of .pem format, it can be converted with the following commands, which require OpenSSL to be installed:

openssl x509 -in mycert.pem -out server.crt

openssl rsa -in mykey.pem -out server.key

mkdir cert

mv server.crt cert/

mv server.key cert/

### **2 Installation Steps**

#### **2.1 Cloning the repository**

Use the following command to clone the repository that contains the Metadata Broker in the current path of your terminal:

git clone https://github.com/International-Data-Spaces-Association/metadata-broker-open-core.git

#### **2.2 Prepare the SSL certificate**

On your host system, create the following directory and put the files server.crt and server.key into this directory:

For Linux users:

Create the following directory path → /etc/idscert/localhost

For Windows users:

Create the following directory path → C:\etc\idscert\localhost

#### **2.3 Configuring the Docker-Compose File**

Once the repository is cloned, the docker-compose file will be found in this path:

<metadata-broker-open-coredocker/composefiles/Meta-Data-Broker/broker-localhost/docker-compose.yml >

The most crucial part of adapting the configuration is to provide the correct location of the X.509 certificate created in **Section 1.2** in the broker-reverseproxy service.

**For Linux users:** if the location of the certificate is “/home/ids/cert”, the corresponding configuration in the yml file is:

services: broker-reverseproxy:

image: registry.gitlab.cc-asp.fraunhofer.de:4567/eis-ids/broker/reverseproxy

volumes:

- /home/ids/cert: /etc/cert/

[…]

**For Windows users:** if the location of the certificate is “c:/etc/ids/cert”, the corresponding configuration in the yml file is:

services: broker-reverseproxy:

image: registry.gitlab.cc-asp.fraunhofer.de:4567/eis-ids/broker/reverseproxy

volumes:

- c:/etc/ids/cert:/etc/cert/

[…]

#### **2.4 Downloading the Docker Images**

All the IDS Metadata Broker Docker images are hosted at the GitLab of Fraunhofer IAIS and no credentials are needed to download the images. The following command is for pulling all docker images:

docker-compose pull

Note that this command should be executed in the same path of docker-compose.yml file.

#### **2.5 Starting up the IDS Metadata Broker**

To start up the IDS Metadata Broker, run the following command inside the directory of the docker-compose.yml file:

docker-compose up –d

This process can take several minutes to complete. You can test whether the IDS Metadata Broker has successfully started by opening [https://localhost](https://localhost/). The result should be a JSON document, providing some general metadata about the IDS Metadata Broker.

Furthermore, the docker-compose logs command can be used to access the logs for a docker-compose.yml file.

#### **2.6 Interacting with the IDS Metadata Broker**

The IDS Metadata Broker accepts and sends messages according to the IDS information model. This model uses the Resource Description Framework (RDF) to leverage the power of linked data. Many examples about representations of IDS concepts can be found at:

<https://github.com/International-Data-Spaces-Association/InformationModel/tree/develop/examples>

The multipart endpoint of IDS Metadata Broker is “/infrastructure”. If the IDS Metadata Broker is running using docker-compose as mentioned earlier, an HTTP POST request can be sent to interact with it. We provide some example messages, illustrating all core functions of the IDS Metadata Broker in this [postman collection](https://www.getpostman.com/collections/1cecd0def2941a993e80).

In addition to the multipart endpoint, the IDS Metadata Broker also serves a prototypical [IDS-REST](https://www.getpostman.com/collections/01d6bf596f67303c08ce) endpoint at “/catalog”. This endpoint will reach a non-prototype state soon after the final specification of the IDS-REST protocol.

### **3 Updating and Stopping the IDS Metadata Broker**

#### **3.1 Updating the IDS Metadata Broker**

To update an existing installation of the IDS Metadata Broker, first repeat the steps explained in section “Downloading the Docker Images”. Containers can be either hot updated or restarted to apply the changes.

To hot update a container, run the following command:

docker-compose up -d --no-deps --build <container name>

Alternatively, one can restart the entire service by running:

docker-compose down

docker-compose up –d

#### **3.2 Stopping the IDS Metadata Broker**

To stop the Broker run the following in the terminal in the same path as the docker-compose.yml file:

docker-compose down

**Omejdn DAPS**

**IMPORTANT**: Omejdn is meant to be a research sandbox in which we can (re)implement standard protocols and potentially extend and modify functionality under the hood to support research projects. It is **NOT** a production grade solution and should not be used as such.

## Running an Omejdn server

### Run with docker

By default, omejdn uses the following directory structure for configurations and keys:

/opt

| config/

| \\_ omejdn.yml

| \\_ user\_backend.yml

| \\_ clients.yml

| \\_ users.yml

| \\_ webfinger.yml

| \\_ oauth\_providers.yml

| \\_ scope\_description.yml

| \\_ scope\_mapping.yml

\ keys/

\\_ signing\_key.pem (The OAuth2 server RSA private key)

\\_ clientID1.cert (The public key certificate for clientID1)

\\_ clientID2>.cert

\\_ ...

It is recommended, that you create the directories config/ and keys/ locally and mount them using docker:

$ mkdir config

$ mkdir keys

$ docker run -d --name=omejdn -p 4567:4567 -v $PWD/config:/opt/config -v $PWD/keys:/opt/keys <dockerimage>

You should provide the config files in config/ and can use the templates in this repo.

**NOTE**: The server private key will be generated if it does not exist, but you can also provide your own.

## Configuring the server

### Environment variables

- APP\_ENV: May be set to 'production' to prevent debug output

- HOST: May be set to modify the host config variable (useful for docker-compose deployments)

- OMEJDN\_JWT\_AUD\_OVERRIDE: May be set to modify the expected 'aud' claim in a JWT assertion in a client\_credentials flow. The standard usually expects the claim to contain the host, hence use this only if necessary.

### Adding a client

First, you need to create a public/private RSA key pair and a X.509 certificate. Then, add the public key certificate to omejdn:

$ cp path/to/your/public/x509.cert keys/<Base64(clientID)>.cert

Now you need to add your client ***clientID*** to the config file config/clients.yml:

- clientID:

name: My Clienti

redirect\_uri: <uri> (optional, required for OIDC)

allowed\_scopes:

- <scope1>

- <scope2>

- ...

attributes:

- key: Attribute1-name

value: Attribute1-value (single value or array)

- key: Attribute2-name

value: Attribute2-value (single value or array)

certfile: <optional, the certificate file to use under keys/>

### Adding a user

Users are added by editing config/users.yml. Passwords are stored in the bcrypt format. Scopes which can be granted by the user must be explicitly defined. If you define an attribute for a scope in config/scope\_mapping.yml, the resulting access token (and ID token) will also include this attribute.

### Scopes

A client can request any subset of scopes in the scopes list when requesting a token. If you define an attribute for a scope in config/scope\_mapping.yml, the resulting access token (and ID token) will also include this attribute. In config/scope\_description.yml you can configure a short description string which is displayed to the user in an OpenID Connect flow upon requesting authorization.

There are some special scopes you may want to use:

* openid: This scope is required if the client shall be able to initia an openid flow.
* omejdn:api: This scope is required if you need to access the omejdn API.
* omejdn:admin: This scope is required if you need to access the omejdn API admin functions.

You can define any allowed client scopes directly in the client configuration. Please note that the user also needs to have the scopes configured in order to request them.

## Requesting an access token (RFC7523)

To request an access token, you need to generate a JWT Bearer token as per [RFC7523](https://tools.ietf.org/html/rfc7523" \l "section-2.2). You may use the script create\_test\_token.rb to generate a JWT Bearer token with your private key. **Note**: You need to generate the respective private key yourself. It is not part of this repo.

An example of a request using the preconfigured client testClient against omejdn to receive an access token looks like this:

$ curl localhost:4567/token --data "grant\_type=client\_credentials

&client\_assertion\_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer

&client\_assertion=eyJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJkZW1vY29ubmVjdG9yMSIsInN1YiI6ImRlbW9jb25uZWN0b3IxIiwiZXh

wIjoxNTQ4Nzg1Mzg2LCJuYmYiOjE1NDg3ODE3ODYsImlhdCI6MTU0ODc4MTc4NiwiYXVkIjoiaHR0cHM6Ly9hcGk

ubG9jYWxob3N0In0.JSQuMf-9Fd7DNna\_-s-sR7eXgcSYNCau5WgurrGJTuCSLKqhZe3odXfunN2vRFgUhU21ADF

lEq96mlbQDueBlMtaXrcHFPSpIUtvuIMIVqQcGYkDdSJr\_VmDuAykCYpyTCkLa7a8DTV-N3sECp-AxUgmEzYIfh8

jW0WS6ehgUzrnpH6t\_h\_GWXKkNSAg3ERakDc4NY02pBGmiN7bmtLZNt5b4LWALiiFiduC7JbIpx4awOU6skMApmz

gLnZmmTG20JlJRg6hAqyHEz5Cd4rUgrt0twmpC0Us\_CG23KdUF5fWI55dcO2qAVvhNQXpqz7IiPcF7-jgkrx4ouk

YNY6eHA

&scope=ids\_connector security\_level"

A response looks like this:

{"access\_token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzZWN1cml0eV9sZXZlb

CI6Nywic2NvcGVzIjpbImlkc19jb25uZWN0b3IiLCJzZWN1cml0eV9sZXZ

lbCJdLCJhdWQiOiJJRFNfQ29ubmVjdG9yIiwic3ViIjoibXlDbGllbnQxI

iwibmJmIjoxNTQ0MTM0NzMxLCJleHAiOjE1NDQxMzgzMzF9.RXvBfka9\_o

Nn7Pgu8royJY25l0ua9jj9REVZPftmggEZreb0oKfhr1bLk9KxWrcT5r2i

svb3GXRONI5zg7S2KZehZK8PQltFQqcbdOOp1Yx0BbJd1ctRiQhCW9kpAo

xHylEahniZXblQ3Z2koFxY82cyVZ48YxUo\_8Tda98CeiFufj7ZW8msGfnT

ac-lwk2yX8hRHoPVSX72GGQWgZGZd9ATubTypLYaqpLuF9hQ5JYk5WKsDq

cFoqk7j\_RxkdM0Uw7njnLWhi7GU7FZZ0UFQi-R8IAhUpIpSofcFsoVPynU

HrjYWB0ANiL-W1kBqXSNCRS9r7SF3ny3LEOVKbuN5g",

"expires\_in":3600,

"token\_type":"bearer"}

The access token will include all requested scopes as well as respective attributes.

## Token verification

All tokens are signed by the server. To retrieve the respecting public key you can use the JSON Web Key Set (<https://tools.ietf.org/html/rfc7517>, <https://auth0.com/docs/jwks>)

$ curl <omejdn URL>/.well-known/jwks.json

There exist libraries for most frameworks that allow validation of JWTs access tokens using JWKS.

## OpenID Connect

By default, OpenID Connect is disabled. In order to enable it, you need to edit omejdn.yml and set openid to true.

### Discovery

You may retrieve the server configuration under

$ curl <omejdn URL>/.well-known/openid-configuration

Please do not forget to configure your external hostname in omejdn.yml under host.

## Transport Layer Security

This service does not include TLS. Omejdn must be served/proxied through a TLS-enabled webserver, such as nginx.

TESTBED CONFIGURATION

Now that the components are individually installed, their configuration must be checked. To do this, we will check for individual performance, two component’s interoperability and finally all together.

When the components are properly configured as the installation guide suggested, we check for their basic functionality.

1/3 interoperability

**Dataspace Connector**

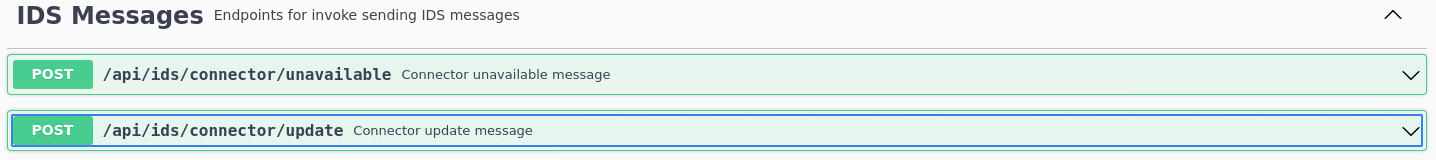
Provider: Create offer/catalogue/representation/artifact/contract

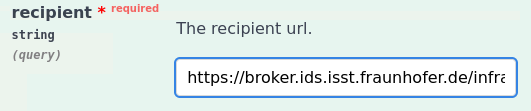
https://international-data-spaces-association.github.io/DataspaceConnector/CommunicationGuide/Provider

Consumer: Access the created offer/catalogue/presentation/artifact/contract

https://international-data-spaces-association.github.io/DataspaceConnector/CommunicationGuide/Consumer

Connect the Dataspace Connector to the public Broker (not the one in the scope of this tesbed) and make sure the connector is successfully registered to the public Broker. This action shows the new keyStore has been successfully installed and the Connector is ready to communicate with the outside.





The recipient url is https://broker.ids.isst.fraunhofer.de/infrastructure

**Metadata Broker**

Check it is working as expected through the API calls. This has been performed through Postman in our laboratory

**OmejdnDAPS**

Once they public key is in the /keys folder, the private key is in the /scripts folder and the client’s information is filled in the config/clients.yml, make use of the /scripts/create\_test\_token.rb script. This provides the user with a dynamic attribute token, which will be the same response the rest of the components will get for the same request token request.

**Important**: According to the standard, the “aud” field of the request token must include the OAuth2-server. The implementation of IDS connectors uses “idsc:IDS\_CONNECTORS\_ALL” in the aud field. To overcome this, the following changes must be performed:

config/omejdn.yml: “host: idsc:IDS\_CONNECTORS\_ALL”  
config/omejdn.yml: “audience: idsc:IDS\_CONNECTORS\_ALL”  
config/clients.yml’s “client\_id” must be the same id used in scripts/create\_test\_token.rb “iss” and “sub”

2/3 interoperability

Connect the Dataspace Connector to the Metadata Broker through the public DAPS (not the one in the scope of this testbed)

Create resources with the Dataspace Connector and make them available at the local Metadata Broker. Perform some resource updates and deletion, and ensure the methods are properly being executed. Test the calls from the Dataspace Connector are performed successfully. Ensure that if any of the security features is missing or misconfigured, the calls are rejected.

Connect the Dataspace Connector to the Omejdn DAPS.

Assuming the earlier steps for the Omejdn DAPS have been correctly configured, the user is now able to obtain a dynamic attribute token with “aud” set to “idsc:IDS\_CONNECTORS\_ALL”. Otherwise, go back to OmejdnDAPS’s important message.

Now the Dataspace Connector’s application properties must be changed to use the newly configured Omejdn DAPS. In the Dataspace Connector’s folder, src/main/resources/application.properties. Look at the “DAPS” section and add/change the lines:

3/3 interoperability

Connect the Dataspace Connector to the Metadata Broker through the Omejdn DAPS without relying on any component out of the testbed.

# CONFORMITY ANALYSIS

## ANALYSIS OF THE SECURITY STATEMENT

## ANALYSIS OF DOCUMENTATION

## PROVEN FUNCTIONALITIES

|  |  |  |  |
| --- | --- | --- | --- |
| **Test code** | | (Ej.- TEST\_0xx) | |
| **Proven functionality:** Identificación de ataque | | Evaluator: | |
| Objetive of the test: Un detector de intrusiones debe de ser capaz de detectar ataques conocidos y debería disponer de las firmas/reglas necesarias para ello. | |
| **Test scenario:** máquina recién instalada | | | |
| **Procedure** | **Expected results** | | **Results** |
| Se realizará un ataque al servicio Apache 1.3.20 aprovechando la vulnerabilidad conocida. Para ello se habilitará una  máquina externa que lanzará el ataque de | El IDS debe de detectar cuando el ataque se ha producido con marcas de tiempo similares a las de la máquina atacante. | | Se puede observar en la información proporcionada por el IDS que se han detectado todos los ataques  lanzados, pero, en algunos casos, con retardos |
| **Conclusion and verdict** | | | |
|  | | | |

# VULNERABILITY SCANNING

# REFERENCES

# ACRONYMS