FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Testing and Implementation of Components in the International Data Spaces

Ana Carolina Marques Chaves

Mestrado em Engenharia de Software

Supervisor: Prof. Rui Filipe Lima Maranhão de Abreu

July 12, 2022

Testing and Implementation of Components in the International Data Spaces

Ana Carolina Marques Chaves

Mestrado em Engenharia de Software

July 12, 2022

Resumo

Hoje em dia os dados são um bem altamente valorizado. No entanto, com a existente dificuldade em acompanhar os dados partilhados, o valor para o proprietário pode diminuir com o número de vezes que eles são transmitidos sem a devida autorização ou compensação do mesmo. Por isso, algumas empresas preferem manter os seus dados privados e intocados para evitar a sua desvalorização, mesmo quando houve um esforço para contruir os seus conjuntos de dados. Consequentemente, a IDSA foi criada para propor e promover um standard internacional de partilha de dados para combater a sua desvalorização e ajudar a partilha dados de forma segura. O seu objetivo é ajudar as empresas a manter a soberania dos dados, o que significa que o controlo dos dados deve estar sempre nas mãos do seu proprietário.

Alguns dos componentes do IDS foram implementados para criar uma infraestrutura autónoma baseada no International Data Spaces. Estes componentes são desenvolvidos considerando os requisitos do IDS, que são, a confiança, segurança e soberania dos dados, um ecossistema de dados, interoperabilidade, aplicações de valor adicionado e mercados de dados. Cada componente foi implementado e testado seguindo as guidelines fornecidas pela IDSA. Adicionalmente, foram realizadas experiências para testar a sua maturidade, interação e as funcionalidades implementadas, e os seus resultados documentados. Cada problema que surgiu foi documentado e as soluções encontradas foram implementadas e explicadas.

Os componentes considerados prontos para implementar num ambiente de produção são, em ordem, a Clearing House, o DAPS e o Metadata Broker. O único componente não considerado pronto é a App Store. Além disso, o não foi possível testar o ParIS porque o IDS Connector utilizado para testar todos estes componentes ainda não oferece interação com ele.

Abstract

Nowadays, data is a highly valued asset. Nevertheless, with the current difficulty in tracking shared data, the value for the owner may decrease with the number of times it is transmitted without proper authorization or compensation from himself. Some companies keep their data private and untouched to avoid devaluation, even when an effort was made to build their data sets. Considering this, the IDSA was created to propose and promote an international standard for data sharing to combat data devaluation and help share data securely. Their goal is to help companies keep data sovereignty, meaning the owner should always have control over its data.

Some IDS components were implemented to achieve a self-hosted infrastructure based on the International Data Spaces. These components are developed taking into account the IDS requirements, i.e. trust, security and data sovereignty, an ecosystem of data, standardized interoperability, value-adding apps, and data markets.

Each component was implemented, deployed and tested following the guidelines the IDSA provides. Furthermore, experiments to test its maturity, interaction and implemented functionalities were developed, and the result documented. Each problem that arose while experimenting and testing was documented, and the found solutions were applied and explained.

The components considered to be mature to implement in a production environment are, by order of maturity, the Clearing House, DAPS and Metadata Broker. The only one not considered ready is the App Store. Moreover, the ParIS was impossible to test, given that the IDS Connector used to test these components does not yet offer interaction with it.

Acknowledgements

My sincere acknowledgement goes to my supervisor at the university. Without him, this work would not have been possible.

I would also like to thank César Toscano. His feedback, knowledge and willingness to help have been vital in bringing this project to where it is now.

Thank you to my family and friends for your interest and willingness to help in any way possible and for providing the necessary distractions.

Many other people also contributed to this dissertation. I want to also thank them for their contributions and time.

Ana Carolina Marques Chaves

Contents

List of Figures

List of Tables

Listings

Abreviaturas e Símbolos

App4EDI	DataspaceApp4EDI
CH	Clearing House
DAPS	Dynamic Attribute Provisioning Service
DC	Data Consumer
DP	Data Provider
DSC	Dataspace Connector
DTM	Dynamic Trust Management
IdP	Identity Provider
IDS	International Data Spaces
IDSA	International Data Spaces Association
MB	Metadata Broker
MPC	Machinery Production Company
PAP	Policy Administration Point
ParIS	Participant Information System
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PMP	Policy Management Point
PXP	Policy Execution Point
SOTA	State of the Art
UC	Usage Control
VM	Virtual Machine

Chapter 1

Introduction

1.1 Context

Nowadays, data is a highly valued asset [13], and its access and exchange are quickly becoming paramount success factors for both companies and entire economies [12]. However, given the effort to create, store and maintain data, the owner or creator is rarely appropriately compensated. He either loses track of shared data with the number of times it is transmitted without proper compensation or authorization from himself, decreasing its value for the owner since he no longer is adequately compensated for its use. Alternatively, he uses a data platform provider that turns vast revenues by trading data and leveraging its value without benefitting the owner. Furthermore, even when there is an effort to generate their data sets, companies tend to keep their data private and sometimes untouched, pay for using a service, or give their data away for free [13].

To combat data devaluation [9] and to help safely share data, the International Data Spaces Association (IDSA) was created. It is a non-profit organization with the primary goal of helping companies keep data sovereignty, balancing the need for shared data usage and the need to protect it in the business ecosystem [9, 10]. More specifically, it means that the data owner should always have control over its data, allowing companies and individuals to self-determine how, when, and at what price their data is used across the value chain [7]. The intent is to accomplish its goal by creating and promoting an international data-sharing standard, IDS Reference Architecture Model (IDS-RAM) [9].

This standard enables companies to sell data as an asset without compromising its value and advertise their data sets by sharing metadata that defines it. This innovative way of publicizing data allows reaching across domains to share data while keeping the data sovereignty. In addition, the architecture standard aids data in achieving its full potential by making it available in cross-company, cross-industry business ecosystems [1].

1.2 Goals

This dissertation aims to create a self-hosted infrastructure based on the International Data Spaces.

The IDSA and Fraunhofer provide components to everyone that requests their use. However, since they host and maintain them, the other companies can only query them for the intended use; they can not, for instance, see their logs. This behaviour generated the need for a self-hosted infrastructure, developed using the code the IDS Association and Fraunhofer make available for companies that want to implement and host their own components. The components used to create said infrastructure are the Connector, Clearing House, Metadata Broker, DAPS and App Store.

Due to this dissertation objective, a study regarding the usage control part of the IDS was also carried out since it allows data owners to keep data sovereignty, one of the IDS's central aspects.

1.3 Contributions

This dissertation aims to create a self-hosted infrastructure using the IDS components. The contributions it brings are the specific steps necessary to implement these components, as well as the experiments made using them and the components' limitations and level of maturity.

Although the components have some documentation regarding their implementation, most lack specifics, or most steps are not specified. Only when the problems arise while implementing them and the development team is warned is that they either answer the issue or the documentation is altered. It is here that this dissertation contributes since it shows the detailed actions to implementing them and the challenge generated to find said actions.

The experiments are essential to see what is implemented in each component. The ones with a guide to their implementation or interaction only show the endpoint and explain what it does. So, this dissertation tests those endpoints to guarantee that the components validate what needs to validate and guarantee that everything is working as it should. An example of this is testing the persistence when removing the components' volumes.

1.4 Dissertation Structure

This dissertation is composed of six chapters:

Chapter 1: The present chapter starts with the introduction, where the context and dissertation goals are presented, and ends with a brief summary of the dissertation contributions.

Chapter 2: Presents the State of the Art, briefly introducing the IDS and its usage control and ending with the related work. The usage control concept is introduced on the SOTA due to being one of the main reasons for using the IDS ecosystem. Moreover, the fact that there is not much information available regarding its implementation was the reason it became the literature review focus.

Chapter 3: Explores the methodology used in this dissertation.

Chapter 4: Starts with exploring the International Data Spaces, including its goals and requirements. It follows the International Data Spaces Architecture Model, where every architectural layer is described. Next, the IDS components are listed and described. The chapter ends with a summary of everything documented in the chapter.

Chapter 5: Explores the IDS components implementations. It presents each component individually, always following the same structure.

Each component starts with a brief description, followed by the steps to implement and test said component. Next, challenges found while implementing them are listed, and solutions, if found, are described. Follows showing the steps to adapt the components to implement them in different use cases from the same entity/company. Next, are described the experiments developed to check the component maturity.

Finally, it ends with a critical reflection of the results by analysing the experiments to see if the components are mature enough for a production environment.

Chapter 6: This chapter seeks to reflect on the objectives to see if they were accomplished and to reflect on the results by analysing the experiments briefly. It ends with a discussion of future works.

Chapter 2

Literature review

2.1 Introduction

The motivation for this state of the art is to better comprehend the use of usage control in the International Data Spaces. For that purpose, INESC TEC projects Great[1] and STVgoDIGITAL[2] were used to connect the theoretical concepts and implementation to practical examples. The choice to use these two projects was due to both having the problem of ensuring data sovereignty, which is the IDS's primary goal. The International Data Spaces Reference Architecture will make it safer to share information between two parties and help keep the data sovereignty of data owners.

The IDS is a virtual data space that ensures secure and standardized data exchange and linkage in a trusted business ecosystem while facilitating cross-company business processes and guaranteeing data sovereignty for data owners [9]. Its primary goal is to help companies keep data sovereignty, defined as a secure, self-determined data exchange among trusted partners [12]. To overcome the challenges created by data sovereignty, it uses usage control. Within the IDS, it is an extension of the traditional access control, and it is the specification and enforcement of restrictions regulating what must (not) happen to data [5]. Thus, usage control is concerned with requirements that pertain to data processing (obligations) rather than data access (provisions) [5].

Furthermore, before the data owner transfers the data to the data consumer, he attaches usage restriction information to it to ensure data sovereignty. The Data Consumer must then accept the data owner's usage policy to use the data [9]. This process allows the data owner to force the data consumer to adhere to the previously negotiated usage policies (e.g., the data consumer must delete the data after 14 days).

A usage policy is the main reason for enforcing usage control. It is a rule that an organization has and is composed of an obligation, permission or prohibition. An example of a policy can be that only the CNC machines can share their data with a data consumer from all the factory machines.

[1] https://www.inesctec.pt/pt/projetos/great
[2] http://www.stvgodigital.pt/

2.1.1 Document Structure

This state of the art starts with this introduction. It is followed by the literature review that focuses on the International Data Spaces and their usage control. The existing work discusses the papers and articles that show how to implement the IDS and IDS Usage Control. Then, the use cases, based on INESC TEC projects, will be described to help explain the usage control concepts. After that, an overview of the IDS usage control, including its essential components and the usage control onion, will be given. Next, usage contracts are discussed with the existing usage policies. Finally, the usage control technologies and their implementation policies are presented.

2.2 Literature review

This literature review objective is to find information about the usage control in the context of International Data Spaces. Based on that topic, the keywords are *International Data Spaces*, *IDS*, *Usage Control*, *Usage Policies* and *MYDATA*.

After extensive research regarding the International Data Spaces, the essential resources found are the Usage Control position paper, International Data Spaces GitHub page, International Data Spaces Reference Architecture and International Data Spaces Rule Book.

The IDS believes it is easier to grant data sovereignty if the control of the data belongs to the providers or owners. Hence, it helps the data owner to maintain control over its data. With that in mind, it states that sovereignty is guaranteed as long as the data owner has control over its data.

The weakness of the existing research is that it is either outdated or does not implement the usage control.

2.3 Existing work

The keywords string chosen are:

"Data Spaces" OR ids AND "usage control" OR "MYDATA" OR "usage policies"

The bibliographic database used was Scopus[3]. And, as the IDS used to be called Industrial Data Spaces, *Data Spaces* instead of *International Data Spaces* was used. After running that query, the results were limited by year (only 2021, 2020 and 2019) because anything before 2019 cannot be used since it will be highly outdated.

That query, limited by year, got nine results. From those results, only three articles were helpful in the existing work.

The first article, *An Industrial Marketplace - the Smart Factory Web Approach and Integration of the International Data Space*, was published in 2020. It implemented the Base connector and did not exchange or enforce usage policies (Used MyData). This article was the most useful.

The second article, *An Architecture for Providing Data Usage and Access Control in Data Sharing Ecosystems*, was published in 2019 and is highly outdated. It achieves usage control with

[3] https://www.scopus.com/home.uri

UCON, XACML, and the IDS reference architecture used is version 2 (at the time of this SOTA, the current version is 3). It was a proposed architecture, so it was not implemented.

The third article, *Data Usage and Access Control in Industrial Data Spaces: Implementation Using FIWARE*, was published in 2020. However, the proposed solution was implemented using the IDS Reference Architecture from 2018 (version 2, like the second article).

There are few articles regarding the implementation of the IDS Reference Architecture and its usage control, given how recent this is. In addition, it is also necessary to be a member of the IDS association to have access to information that allows the IDS implementation. It is possible to see that by the volume of articles/projects implementing the IDS. However, only three show some relevant information about its implementation.

The International Data Spaces Association has various projects developed with partners (companies) implementing the IDS in different sectors and technologies. These projects are called the IDS use cases and show how the IDS standard adopts to real-life challenges. IDSA members developed them to help companies share any data in any ecosystem, which will ultimately transform the digital economy worldwide [7]. Nevertheless, there are no details about its actual implementation, only the benefits, challenges, successes and partnerships.

2.4 Use Cases

The use cases used in this state of the art to explain the concepts are:

Great Capture a set of data acquired by sensors installed on a CNC machine and made available to the external organization responsible for the maintenance of the equipment;

STVgoDIGITAL Making production orders available from an industrial organization to its production subcontractors, as well as their actual production status.

Regarding these use cases, the main problem the IDS must solve is data sovereignty. Using the data spaces facilitates communication between all the parties involved and makes it safer for the data owner to share its data.

More specifically, in the first use case (*Great* project), it is crucial to use IDS since it can facilitate and secure data sharing between the machinery production company (MPC) and the company using the CNC machines. The IDS environment allows the data owner to control how, when and which data to share.

In the second use case (*STVgoDIGITAL* project), using IDS will facilitate the sharing of production orders to all the subcontractors while guaranteeing data sovereignty and also that the knowledge is not being misused.

2.5 Usage Control Concepts

While traditional access control restricts the access to specific resources (for instance, a file or a service), the IDS architecture supports data-centric usage control. Generally, the IDS's primary

goal is to enforce usage restrictions for data after the access has been granted. Consequently, the usage control purpose is to implement policies to the data being exchanged and to continuously control how messages may be processed, aggregated, or forwarded to other endpoints [5]. The usage control enforcement prevents the IDS connectors from misusing the data at runtime, for example, by forwarding personal data to public endpoints. So, usage control is a system integrator's tool for ensuring that the architecture they are building does not violate the security requirements and an audit mechanism that creates evidence of a compliant data usage [5].

Below are presented some security requirements that cannot be achieved by using traditional access control:

- **Secrecy:** If a node does not have the clearance needed, classified information cannot be forwarded.
- **Integrity:** To guarantee its integrity, untrusted parties cannot modify critical data.
- **Time to live:** In order to guarantee the persistence of data, it must be deleted from storage after a predefined period.
- **Anonymization by aggregation:** To prevent de-anonymisation of individual records, personal data must be aggregated using a sufficient and distinct number of records when dealing with untrusted parties.
- **Anonymization by replacement:** Data which allows a personal identification (e.g., faces in camera images) must be replaced by an adequate substitute (e.g., blurred) to guarantee that individuals cannot be de-anonymization from the data [5].
- **Separation of duty:** Data sets from competitive entities, e.g., two knitwear companies, can never be aggregated or processed by the same service [5].
- **Usage scope:** Data cannot leave the connector to an external endpoint. It can only be used as input for internal data pipes.

The International Data Spaces Usage Control Position Paper [5] defines two main activity streams to implement usage control within the IDS:

Firstly, a policy language to express usage restrictions is developed. That language is descriptive, technology-independent, based on the Open Digital Rights Language[4] (ODRL) and further detailed in the form of IDS contracts [5]. Currently, the IDS supports 21 predefined classes that express the most used usage restrictions, as listed in section 2.7 from page 14.

Secondly, usage control technologies are developed to enforce the usage restrictions at a technical level [5].

[4] https://www.w3.org/TR/odrl-model/

2.5.1 Data-flow Components

Data flows need to be monitored and potentially intercepted by control points (PEPs) to enforce usage restrictions. These intercepted data flows are passed to the decision engine (PDP) for requesting permission or denial of the data flow [5]. Regarding the MPC use case, imagining that only the MPC management team can access the data, it is possible to intercept the data flow and check which team is accessing it.

It is possible to enforce usage control inside a connector, using a combination of components, some of them being always mandatory, such as the PEP and PDP. This section explains the usage control flow using all the components in figure 2.1 and what each one does with the help of the MPC use case:

PEP (Policy Enforcement Point): Control point to intercept data flows.

PDP (Policy Decision Point): Decision engine. It is responsible for answering the incoming requests (i.e., data flows) from a PEP with a decision. It starts the policy evaluation.

PIP (Policy Information Point): It provides missing information to make a decision. It can also obtain contextual information for or about the interception action (e.g. geolocation of requested device or data flow information). Considering the use case, a policy could be that the information accessed is about CNC machines.

PXP (Policy Execution Point): Used to take additional actions based on the policy rules (e.g. delete data in storage infrastructure outside the connector). The consumer connector handles this component. Considering the use case, some examples of policies could be:

- Send an email when MPC is using the data
- Delete the data after 14 days.

PMP (Policy Management Point): Manages the usage restrictions, which include the usage restrictions instantiation, negotiation, deployment, and revocation, as well as conflict detection and resolution [5]. When policies are stored independently from the data, it takes the responsibility to exchange the usage restrictions between different systems.

PAP (Policy Administration Point): Entry point for the usage policies specification. It usually has a user-friendly graphical interface (e.g. MyDATA interface), section 2.6.1. Considering our use case, some examples of policies could be:

- Data Consumer can only keep the data for two weeks;
- MPC cannot have access to personal information;
- Only MPC_Maintenance_Connector can have access;
- Only the MPC maintenance team can have access;
- MPC can only have access to the CNC logs.

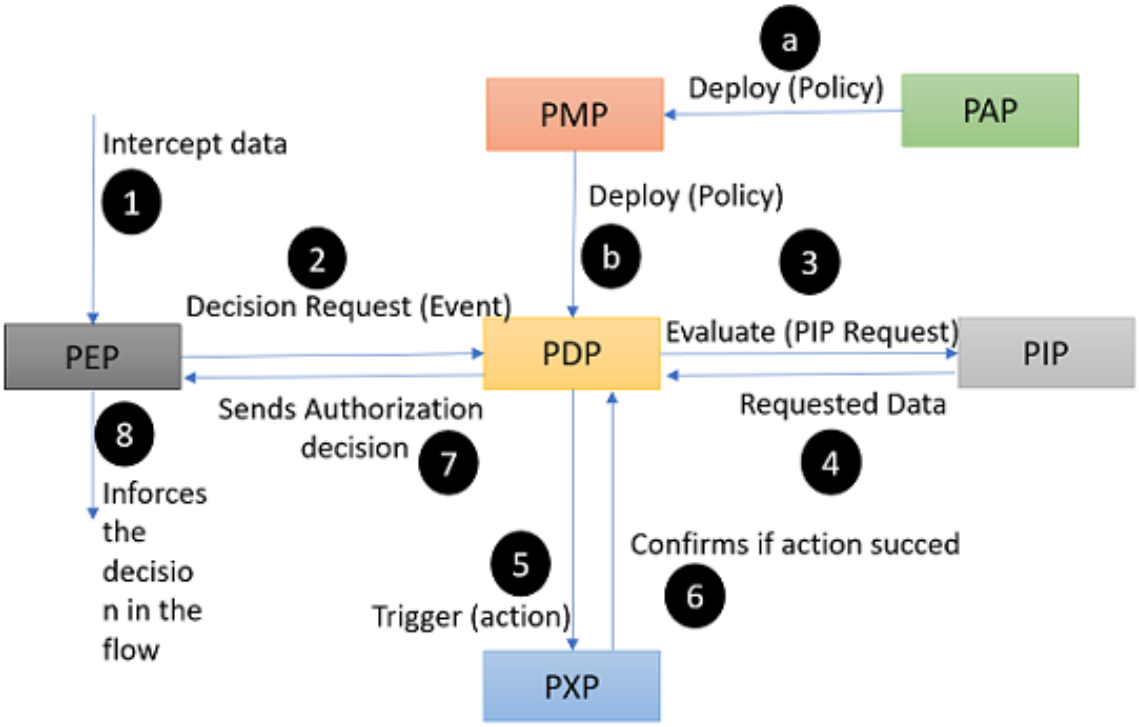


Figure 2.1: Example of a Usage Control data-flow [5]

Below is explained the Data-flow step by step, as given in the IDS Usage Control official document:

1. PEP catches the data flow;
2. PEP converts the data flow into a decision request and sends it to the PDP;
3. PDP starts the policy evaluation process and calls a PIP to retrieve the additional information;
4. PIP answers the PDP with the solicited data;
5. PDP triggers an additional action at a PXP;
6. PXP confirms the action success to the PDP;
7. PDP sends authorization decision to the PEP; and,
8. PEP enforces the decision on the intercepted data flow.

The steps a) and b) represent the PMP and PAP interaction. They show the policy deployment as an example sequence.

2.5.2 Usage Control Onion

The IDS has different stages of usage control, called Usage Control Onion, depicted in figure 2.3 taken from the Usage Control Position Paper, that starts from inside the onion and finishes in the outer layer. Each layer represents a degree of usage control to guarantee a data owner the maximum control of its data. It is divided into six layers, and the data level of control keeps minimizing from the inner layer to the outer layer.

2.5.2.1 IDS Connector at Data Provider

The *IDS Connector at Data Provider* is the inner part of the usage control onion.

The usage restrictions are usually provisions handled by a PEP on this connector. It enforces policies such as masking and filtering data (e.g., personal information must be aggregated) before leaving the company or when the DC can access data (e.g. only within business hours) [5].

2.5.2.2 IDS Connector at Data Consumer

On the data consumer connector, the usage control policies enforced are usually obligations [5] for the data consumer, such as "Data can only be kept for one week" or "Send an email to the data provider when the data is being accessed".

Depending on the usage restriction, the technical enforcement is either handled by a PEP or a PXP [5]. The PEP handles the limitation of a data flow to a specific target system, ensuring the correct usage purpose [5]. The PXP deletes the data in the storage infrastructure outside the connector [5].

2.5.2.3 Storage Infrastructure

The usage control enforcement can be implemented in the storage infrastructure. For right now, the IDS has two different ways to implement this, as shown in figure 2.2, taken from the IDS Usage Control Position Paper:

1. The storage infrastructure is used without modification. Furthermore, the encryption and decryption are handled by a PEP inside the connector;
2. A PEP is inserted into the storage infrastructure, and it controls the usage of the data.

2.5.2.4 Applications

The application controlling the data flows can have a PEP integrated. That application is developed using technologies such as Degree (Dº) to directly integrate usage control capabilities at compile time [5].

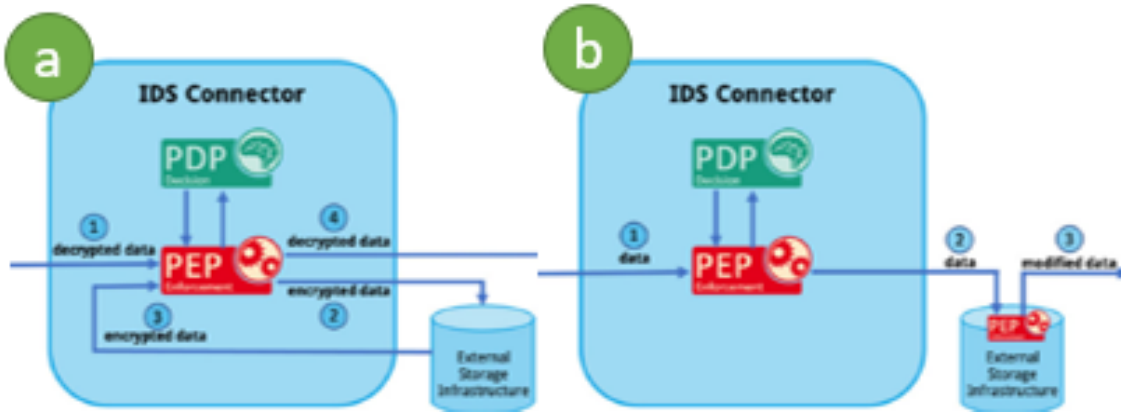


Figure 2.2: UC within Storage Infrastructure [5]

2.5.2.5 Client System

The usage control on the client system is hard to achieve since it can involve alterations to the operative system.

2.5.2.6 Operative system

The *operative system* is the last layer of usage control, and it is what we cannot control. For instance, the users could take a screen picture instead of making a screenshot.

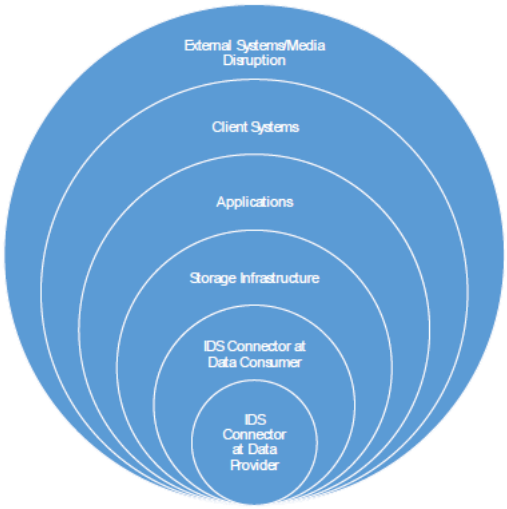


Figure 2.3: Usage Control Onion illustration [5]

2.6 Contracts

An IDS Contract is implicitly divided into two main sections, as shown in figure 2.4:

- **Contract specific metadata**: it does not affect the enforcement of the contract. Examples of information are the date when the contract has been issued or references to sensitive information about the parties involved [5];
- **IDS Usage Control Policy** of the contract: main motive to enforce a technical and organizational usage control
  - It contains several Data Usage Control statements (e.g., permissions, prohibitions and obligations) called IDS Rules.
  - It is specified in the IDS Usage Control Language, a technology-independent language.

Figure 2.4: IDS Contracts Usage Control terms [5]

The technically enforceable rules are then transformed into a technology-dependent policy (e.g., LUCON, MYDATA) to facilitate data sovereignty enforcement [5].

A contract results from the policy specification and negotiation process between two parties. A negotiation procedure is depicted in figure 2.5.

However, before that procedure, it is first necessary for the Data Provider and Data Consumer to specify their Usage Control policies (using the IDS policy editors, section 2.6.1). The outcome of the policy editor is a machine-readable contract in the IDS policy language (based on ODRL) [5]. The Data Usage Control statements of the Contracts shall them be negotiated on a semi-automated policy negotiation platform. There, the parties involved agree on a contract that benefits both. After all this process, the Data Usage Control statement of the contract agreement needs to be transformed to a technology-dependent language (e.g. MyDATA).

In figure 2.5, the negotiation process that, if succeeded, results in a contract starts. First, the Data Consumer (which in this example will be the MPC) communicates its interest in the Data

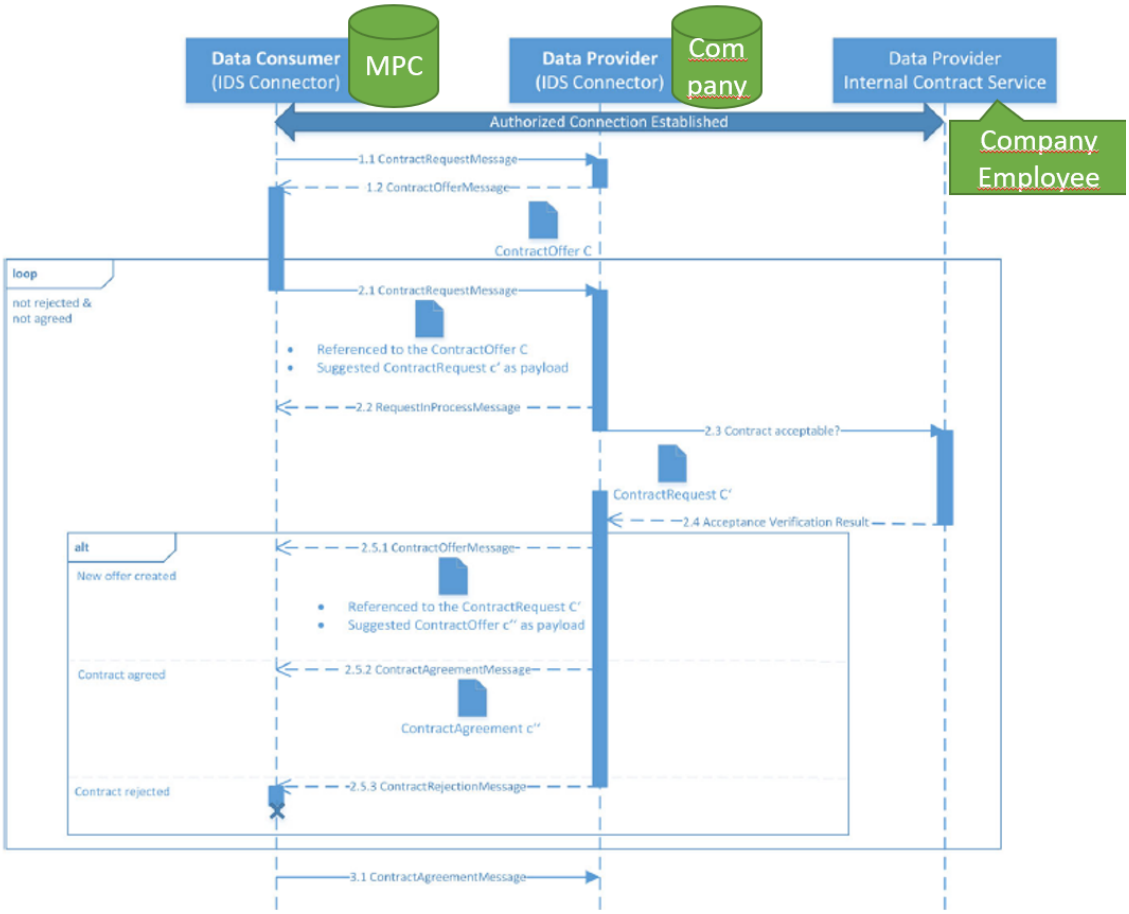


Figure 2.5: Policy negotiation between Data Provider and Data Consumer [5]

Provider data set by creating a Contract Request (1.1.) and then sends it to the Data Provider (DP) connector. The DP answers the request by sending him the contract offer with terms (e.g. MPC can only use the data for one week) (1.2.). In a second interaction, the consumer acknowledges the offer and then counter-offers with a new proposal that suits him better (e.g. he can use the data for two weeks) (2.1.).

Consequently, the provider needs to decide if he:

- responds with an adjusted contract offer (2.5.1.),
- accepts it and responds with a contact agreement (2.5.2.), or,
- rejects the request and terminates the interaction (2.5.3.).

As this process needs extensive knowledge of the context of the specific scenario, it can be made by a Data Provider company employee that reviews the offer and decides which alternative to choose (2.5.1, 2.5.2 or 2.5.3). This process can be automatic if a company has proper business logic.

2.6.1 IDS Policy Editors

Figure 2.6 (e.i. Policy Administration Point) the IDS policy editor allows data owners and providers to specify their usage control restrictions. They usually have a graphic interface and offer various guidance levels depending on the user's skill level [5]. The IDS Lab offers an IDS Policy editor to express the classes presented in section 2.7.

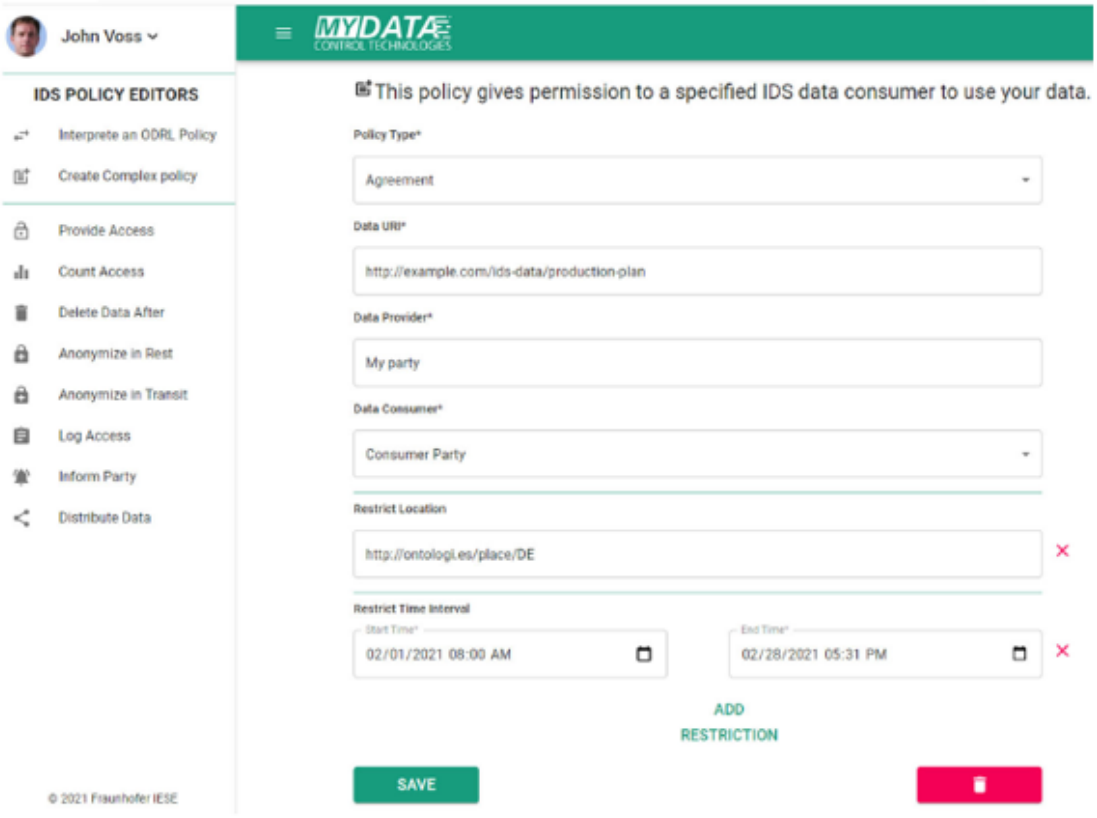


Figure 2.6: Policy editor screenshot, taken from the Usage Control Position Paper

2.7 Policy Classes

The IDS currently offers 21 predefined policy classes, as shown in table 2.1. These classes are continuously updated and growing, as seen by the change from the last usage control document, where there were only 14 classes. Below the classes are explained in more detail:

1. **Allow or inhibit data usage**: This policy class is an abstract category. It prohibits or permits a specific Data Consumer to operate an action on the data. When prohibiting an action, it is necessary to specify the action being prohibited. When the action is "use", seeing that that action has a comprehensive range of meanings, including, but not limited to, "print", "distribute", "read", and "delete", all those actions are allowed. For example, if we want the

Data Consumer only to be able to read and print the data, we need to specify a policy that only permits the actions "read" and "print".

2. **Perpetual data sale:** The IDS allows the Data Providers to sell their data. For a Data Consumer to buy that data, he must fulfil the conditions specified on the contract. An example of a condition is a one-time payment made. This policy class addresses the conditions associated with a data sale contract.

3. **Data rental restrictions:** Contrary to the last category, this class policy concerns the conditions associated with renting data. An example of this category is that a Usage Control Technology regularly has to check if a Data Consumer paid the fee specified in the rental contract.

4. **Limit the data usage to a role or group of users:** A Data Provider may want to restrict data usage to a specific role or group of users. To enforce this policy, a Usage Control Technology must verify if a user has a specific role or if he is a member of a specific organization. An example of this policy, using the MPC use case, is that the DP limits the use of data for everyone who is an MPC company member.

5. **Limit the data usage to specific connectors:** As the IDS allows the assignment of more than one connector to an IDS party, this class addresses the need to restrict the data usage to specific connectors of the specific Data Consumer. This class is useful, for instance, if MPC has a connector for each department. As the Data Provider only wants the MPC maintenance connector to have access to its data, he can restrict the access to the other MPC connectors.

6. **Limit the usage of data to applications or a group of systems:** The system and applications must be IDS certified for this policy class to be enforced. The Usage Control Technology will validate the certificate and then enforce the policy.

7. **Purpose restricted data usage:** This policy class restricts the usage of data to a specific purpose(s). The IDS is still evolving this concept. An example of this policy is "If the purpose is maintenance, allow the usage of data and else if the purpose is marketing, prohibit the data usage".

8. **Limit data usage when a specific event occurs:** This category represents the prohibition or permission to use data under specific conditions. The Data Provider can name conditions that address, for instance, "When there is a failure". Furthermore, it is possible to define a policy that says, "If there is a failure in the CNC machines, send the information to the MPC".

9. **Limit data usage to a specific time interval:** This policy allows or inhibits data usage in a specific time interval, e.g., beginning in January 2022 and finishing in February 2022.

10. **Limit data usage to a specific duration of time:** Just as the previous one, this policy also allows or inhibits data usage. However, instead of a time interval, this policy class serves to

define a time duration. An example is to define a policy that allows the use of data for two weeks.

11. **Limit data usage to specific locations:** This policy class permits or prohibits data usage based on the Data Consumer location.

12. **Restrict the data usage to N times, maximum:** This policy class restricts the number of times an action can be executed. For example, a policy can say that the data cannot be printed more than 5 times, or that data can only be read 15 times.

13. **Limit data usage to the connector's security level:** The IDS connectors have a security level. This policy restricts data usage to connectors with a specific security level.

14. **Use data and delete it after:** This policy class requires the Data Consumer to delete the data. It needs to be specified in the policy specification regarding when to delete it. It can, for instance, be right after using it or after two weeks.

15. **Modify data (in transit):** In some cases, it can be necessary to modify the data before the Data Consumer accesses it (e.g., anonymize). According to this policy, the Usage Control Technology intercepts the data and applies the modifications.

16. **Modify data (in rest):** In opposition to the last class, this policy demands that data modifications be done when the data is still on the Data Provider database.

17. **Log data usage locally:** The Data Provider can request to log the transferring information from the DP to the DC. The IDS already has logging capabilities. However, the Usage Control technologies can also apply the logging policy to the system and log the usage information locally. An example of this policy is logging the data modification.

18. **Notify a specific group of users or party when data is used:** This policy specifies when the DP wants to know when its data is being accessed, when it leaves their sites, or even when it arrives on the DC site.

19. **Attach policy when distributing the data to a third party** When distributing data to a third party, the DP can demand to have additional usage policies. This type of policy forces the DC to transmit the usage policies to the third party and have him agree with them before the latter can receive the data.

20. **Only distribute data if it's encrypted:** This policy is used when the Data Provider demands that the data can only be shared if it is encrypted. An example of this policy is that a Data Provider can demand that when sharing the data with a third party, the data needs to be encrypted.

21. **Limit data usage to a specific state:** This state refers to the state of the connectors or the contracts. An example of this restriction is refusing data access if the contract is terminated.

2.8 Usage Control Technologies

As there are different technologies to implement data usage control, the IDS offers three approaches researched and developed within Fraunhofer[5]: The MYDATA Control Technologies (MyDATA), the Logic-based Usage Control (LUCON) and Degree (Dº).

2.8.1 MYDATA Control Technologies

MyData is the oldest and, therefore, more complete approach. Not only it already has a user interface, but it also transforms the user's natural language policies (e.g., "Use my data for only seven days") into IDS Usage Policies.

MYDATA is a technical implementation of data sovereignty. It implements it by monitoring or intercepting security-relevant data flows [5].

2.8.1.1 Usage Control App

The most recent usage control update introduced the Usage Control App (UC App). MYDATA provides it to enable an easier integration within the connector [5], and it can translate IDS Usage Control Policies (based on ODRL language) into MYDATA policies [5]. In short, the UC App has the MYDATA main components in one app, enabling an easier integration in the IDS connectors.

2.8.1.2 Communication Flow

Figure 2.7 demonstrates the main flow from a Data Provider to a Data Consumer. The core container on the DP side handles the data flow, starting at the Data Provider data source, where various sources can connect with an IDS Connector. As part of the routing and before data flows to the Data Consumer, the UC App is invoked, and the Data Providers' specific usage control rules are applied to the data [5]. It is then sent to the IDS Connector on the Data Consumer side, which does the same process. Data sources or sinks can reside inside or outside the IDS connector, as shown in figure 2.2.

It is possible to implement the usage control using the router or interceptor. On the Provider side, it can use either one since the monitoring need is minor. On the Consumer side is recommended to use the Interceptor.

When using the routing, as shown in figure 2.7, the Usage Control App has to be explicitly configured as a processor in the route [5], which means that the UC App is only called where it was implemented. When using the Interceptor pattern, the Usage Control App is implicitly called between every processing step [5]. It is possible to see the difference in the pattern's behaviour in figure 2.7 below the rectangles.

It is important to note that the UC App has at least a PMP, PEP and PDP.

[5] https://www.fraunhofer.de/en/about-fraunhofer.html

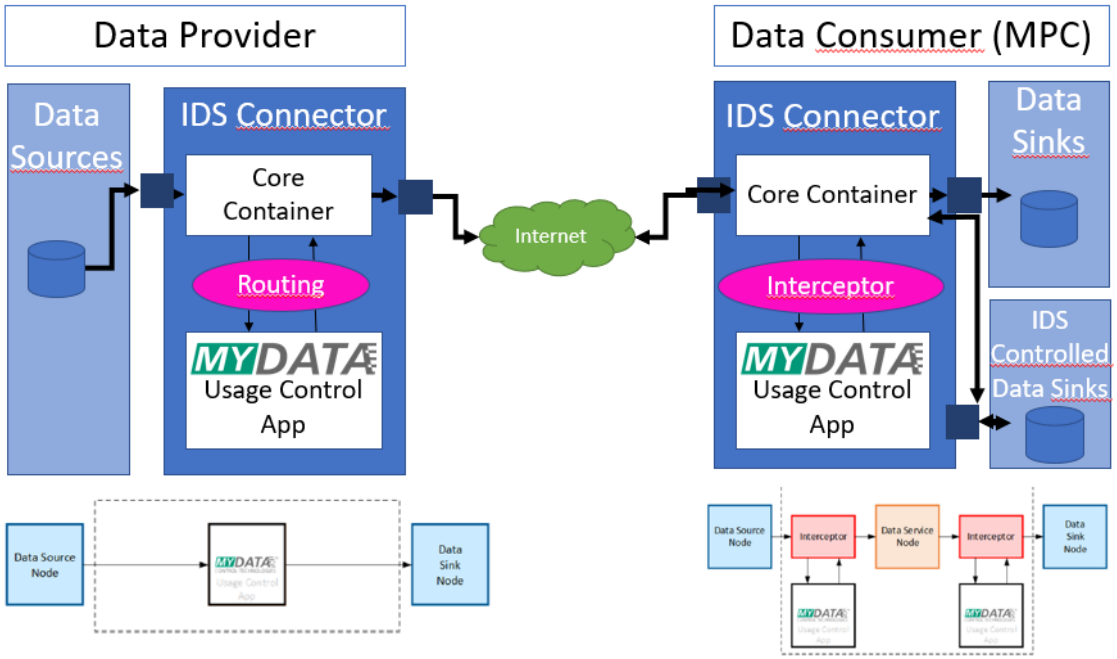


Figure 2.7: MyData Communication Flow

2.8.2 LUCON

LUCON (Logic-based Usage CONtrol) is a policy language for controlling data flows between endpoints [5]. It controls how messages may be processed and passed around services [5]. After the last update, LUCON can now fully support the IDS policy language and perform remote attestation after successful policy negotiation [5].

The security goals, as referenced in section 2.5 that LUCON can help to achieve are:

- **Secrecy:** By defining a policy that disallows data forwarding to nodes explicitly marked as trusted.
- **Integrity:** By comparing the input and output of a node
- **Time to live:** A state monitoring number of usages in a workflow can be defined to track the allowed number of usages.
- **Anonymization by aggregation:** By defining a policy that specifies the number of records that must be aggregated before they can be forwarded.
- **Anonymization by replacement:** This task needs to be wrapped inside a micro-service.
- **Separation of Duty:** By defining a policy that disallows conflicting labels.
- **Usage scope:** Defining a policy that disallows data with a specific label to leave the connector [5].

LUCON is embedded into the Trusted Connector software stack, and policies can be submitted via IDS Usage Control Policy Negotiation. IDS policies need to be processed and translated into valid LUCON policies locally. LUCON supports the full IDS Contract Negotiation interaction pattern. This way, all solutions support a similar user experience [5].

2.8.3 Degree

While MyDATA and LUCON provide usage control for existing applications, Degree (Dº) takes another approach [5]. Dº is a Domain Specific Language (DSL) for the development of Data Apps that takes the usage control into account from the beginning of the applications development. Dº uses Java as a host language.

Data Apps developed using Dº are transformed into Java apps that are then compiled into executable applications [5]

2.8.4 Policies Comparison

The IDS currently offers twenty-one main policy classes that the Information Model supports. These IDS policy classes can be refined and transformed into technology-dependent, machine-readable policies and obtain an ensured relation to legally binding formulations [11].

As these technologies are still being developed, they do not yet implement all the IDS-defined policy classes. Table 2.1 shows the classes currently supported by the technologies.

2.9 Discussion

This SOTA presented the IDS usage control, usage control technologies, IDS contracts and IDS policy classes.

Regarding the use cases, based on the INESC TEC projects *STVgoDIGITAL* and *Great*, it is possible to see the main differences between them. *Great* has a more significant focus on the usage control policies to facilitate data sharing between the data owner and the data consumer, which will then be used for maintenance purposes. *STVgoDIGITAL* is focused on the secure sharing of production orders on the supply chain, which results in more straightforward usage policies. When verifying the usage control policies each project would need to implement, it is possible to identify that project *Great* would implement policies number 1, 5, 10 and 14 from section 2.7, while project *STVgoDIGITAL* would only implement policies number 5 and 9.

Three technologies can be used when implementing those policies: LUCON, MyDATA and Degree. As the usage control is still very recent, not all technologies can implement all policies. Table 2.1 depicts all policies available, at this time, with the technologies that implement them. When connecting the policies defined above with the policies in table 2.1, it is possible to see that not all technologies implement them yet. More specific:

- (1) Allow the Usage of the Data: Is implemented by all technologies. However, when using Degree some restrictions need to be further researched.

Table 2.1: Policy Classes Support by Usage Control Technologies [5]

	MyDATA	LUCON	Degree
1: Allow the usage of data	Yes	Yes	Yes[1]
2: Perpetual Data Sale (Payment once)	Yes	No	Yes[1]
3: Data Rental (Payment frequently)	Yes	No	No
4: Role-restricted Data Usage	Yes	No	Yes[1]
5: Connector-restricted Data Usage	Yes	Yes	No
6: System-restricted Data Usage	Yes	Yes	No
7: Purpose-restricted Data Usage Policy	Yes	Yes, via purpose bound apps	Yes[1]
8: Event-restricted Usage Policy	Yes	No	Yes[1]
9: Interval-restricted Data Usage	Yes	Yes	Yes[1]
10: Duration-restricted Data Usage	Yes	Yes	No
11: Location Restricted Policy	Yes	No	No
12: Restricted Number of Usages	Yes	No	Yes[1]
13: Security Level Restricted Policy	Yes	Yes	No
14: Use data and delete it after	Yes	Yes, through containers	No
15: Modify data (in transit)	Yes	Via Apps	No
16: Modify data (in rest)	Yes	No	Yes[1]
17: Local Logging	Yes	Yes	Yes[1]
18: Remote Notifications	Yes	Yes	Yes[1]
19: Attach Policy when Distribute to 3rd Party	Yes	Yes	No
20: Distribute only if encrypted	Yes	No	Yes[1]

[1] There is not an automatic transformation available. Not all cases can be covered because of arbitrary extensions allowed in IDS Policies [5].

- (5) Connector-restricted Data Usage: Is implemented by LUCON and MyDATA. Degree does not implement this policy.
- (9) Interval-restricted Data Usage: Is implemented by all technologies. However, when using Degree some restrictions need to be further researched.
- (10) Duration-restricted Data Usage: Is only implemented by MyDATA and LUCON.
- (14) Use Data and Delete it After: Is supported by MyDATA. It is also possible to implement this policy with LUCON. However, it is necessary to use containers. Degree does not support this policy.

2.10 Conclusion

This state of the art aims to study the International Data Spaces and its usage control. A literature review focused on the current information about those two topics was developed. Extensive research was also conducted on projects, papers, and articles illustrating detailed information about the IDS implementation.

This SOTA presented the two use cases, *STVgoDIGITAL* and *Great*, to help link the theoretical concepts to a more practical example. The overall purpose of these projects is to facilitate the sharing of production orders in the supply chain and secure data sharing between the company using the data and the data owners.

This chapter also introduced the usage control components, explaining how the IDS enforces the usage control. Moreover, the degrees of UC necessary to guarantee a secure data exchange since the provider connector to the operative system was described and detailed.

Finally, usage contracts, the last step to sharing data, were described. The available policy classes were described to understand the usage control usefulness better. Three usage control technologies, LUCON, MyDATA and Degree, were introduced to implement the usage control.

Chapter 3

Activity List

This chapter explores how the work was organized to respond to the project's challenges. It was structured in several phases:

Phase I - Problem study This phase included the domain problem study, which implied a study of INESC TEC projects and the literature review focused on the concepts of:

- International Data Spaces;
- Access and usage control concepts in the IDS context.

Phase II - IDS Components Study This phase includes the detailed study of the IDS Components that are to be implemented or worked with:

- IDS Connectors;
- IDS Metadata Broker;
- IDS App Store;
- IDS Data Apps;
- IDS Clearing House;
- IDS Identity Provider.

Phase III - Test Environment Study and Test This phase includes the following:

- Study of the use case and the test environment based on it;
- Study the information exchanged;
- Study the components involved;
- Test the environment.

Phase IV - Test and Implementation This phase includes:

- IDS Components code repository study;
- IDS Components implementation;

- Challenges and solutions registration;
- Development of experiences to understand the maturity of the components;
- Components test.

Chapter 4

International Data Spaces

4.1 Introduction

The International Data Spaces[1] (IDS), formerly known as Industrial Data Spaces, is an international initiative developed by the IDSA members that proposes a Reference Architecture Model (IDS-RAM) as an international standard for data sovereignty and related aspects. It includes secure and trusted data exchange requirements in business ecosystems [9]. Additionally, the IDS aims to be applied across company borders and existing supply chains [9, 3]. For this to be applicable, the IDS needs to be easy to adopt, technology-independent, and reusable, and comply with existing technologies [6, 3]

The IDS standard solves a market obstacle: For data to unfold its value creation potential, it must be described and tradable according to a global and interoperable standard [4]. Its primary goal is to guarantee data sovereignty, which the IDS Reference Architecture document defines as "a natural person's or corporate entity's capability of being entirely self-determined with regard to its data" [9].

The IDS uses usage control to cope with the data sovereignty challenges. In chapter 2, this was extensively explained in the International Data Spaces context. It is essentially the specification and enforcement of restrictions regulating what must (not) happen to data and is concerned with data consumer obligations [5].

The strategic requirements the IDS aims to meet are trust, security and data sovereignty, an ecosystem of data, standardized interoperability, value-adding apps, and data markets.

Trust is the IDS basis, and the use of certification achieves it. Each participant is evaluated and then certificated. Only after the certification is granted is he capable of accessing the trusted business ecosystem.

Security and Data Sovereignty are fulfilled by evaluation and certification of each technical component. According to the IDS core aspect of ensuring data sovereignty, a data owner attaches the usage restriction information to their data before transferring it to the Data Consumer [9]. To use the data, the Data Consumer must accept the data owner's usage policy [9]. This process

[1] https://internationaldataspaces.org/

allows the data owner to force the data consumer to adhere to the previously negotiated usage policies (e.g., the data consumer must delete the data after fourteen days). A usage policy is a primary reason for enforcing usage control. It is an organization's rule and is composed of an obligation, permission or prohibition.

The Ecosystem of Data follows the notion of decentralized data storage. This means that data remains with the data owner until data is transferred or exchanged to a trusted party. This approach requires a comprehensive description (value and usability) of each data source. Additionally, the Metadata Brokers provide data search services.

Standardized Interoperability requirement refers to the IDS Connector, the main technical component. Considering this is the central IDS component, it can be implemented in different variants and acquired from different vendors. Nevertheless, each Connector can communicate with any other Connector or technical component [9].

Value Adding Apps are data services injected as Apps into the Connector to provide services on top of data exchange processes, e.g., services for data processing, data format alignment, and data exchange protocols [9].

Finally, Data Markets enable the creation of novel data-driven services that use data apps and promote new business models for these services by providing clearing and billing mechanisms, domain-specific broker solutions, and marketplaces.

4.2 IDS-RAM

IDS-RAM focus on a creation of a secure "network of trusted data". The general structure of the Reference Architecture Model is illustrated in figure 4.1. The model is comprised of five different layers: Business, Functional, Process, Information and System. In addition, it has three perspectives that need to be implemented across all five layers: Security, Certification, and Governance.

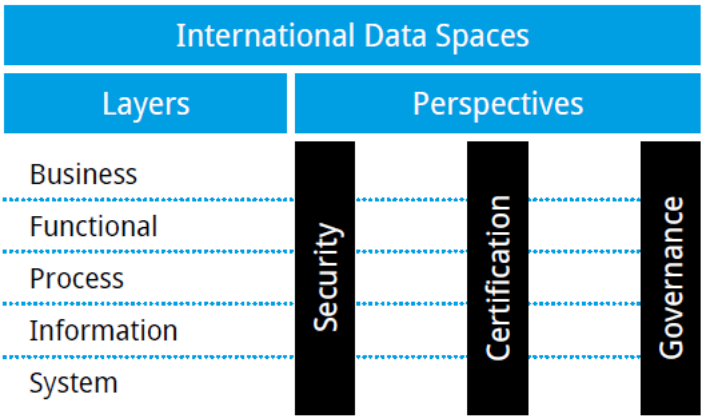


Figure 4.1: Reference Architecture Model structure, as presented by the IDS Reference Architecture document [9]

4.2.1 Business Layer

The Business Layer mainly describes the IDS participants' roles and interaction with each other.

Four categories of roles are defined: Core Participant, Intermediary, Software/Service Provider, and Governance Body.

The Core Participant is any organization that owns, wants to provide, consume or use data. It subdivides into five roles:

Data Owner who owns the data and makes it available. It can be the same entity as the Data Provider;

Data Provider is responsible for providing data and can be the same as Data Owner. Its main activity is providing a Data Consumer with data from a Data Owner. Furthermore, it can use Data Apps to enrich or transform the data to improve its quality or log information to the Clearing House to facilitate billing or resolve a conflict;

Data Consumer receives data from a Data Provider. It can establish a connection with the DP directly or by inquiring a Broker to search for datasets and using the response to connect with the DP. Additionally, it can also log information to the Clearing House, use Data Apps to enrich or transform the data received, or use a Service Provider to connect to the IDS (if it does not deploy the technical infrastructure for participation itself);

Data User is the entity that has the legal right to use the data and usually, but not necessarily, is the same as the Data Consumer; and,

App Provider is responsible for developing Data App to be used on IDS.

The Intermediary's roles belong to the Broker Service Provider, Clearing House, Identity Provider, App Store, and Vocabulary Provider, as detailed in chapter 4.3.

Software / Service Provider category role refers to companies providing software or services to IDS participants. Specifically, Software Providers supply software for implementing the functionalities required by IDS, and Service Providers host the technical infrastructure enabling organizations that do not deploy such infrastructure to participate in the IDS [9].

Lastly, Governance Body refers to Certification Body and Evaluation Facilities (responsible for the certification process and issuing certificates) and the International Data Spaces Association.

Figure 4.2 from page 27 shows how the various roles in the IDS Ecosystem interact.

The Data Owner is who owns the data and permits the Data Provider to provide the data.

Data Provider interacts with the Service Provider by transferring data to them, with the Broker Service Provider by publishing metadata that defines its data sets, with the Clearing House by logging every transaction, and with the App Store Provider by using the Data Apps they provide.

The Data Consumer interacts with the Service Provider by receiving the Data Provider's previously transferred data. It interacts with the Broker Service Provider by searching metadata that

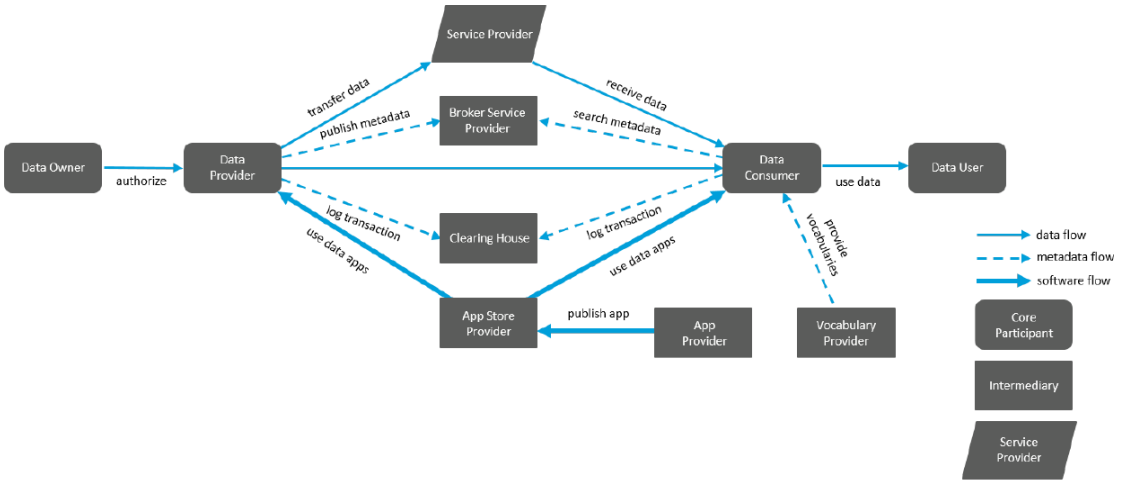


Figure 4.2: Roles and Interactions, as presented by the IDS Reference Architecture document [9]

describes the available data sets. With the Clearing House, it logs every transaction. It interacts with the App Store Provider by using their data apps. Furthermore, with the Vocabulary Provider by using the vocabularies they provide.

The App Provider publishes apps on the App Store Provider that the Data Provider and Data Consumer will use.

The Data User uses the data the Data Consumer receives.

4.2.2 Functional Layer

This layer defines the IDS functional requirements, depicted in section 4.1, and the features to be implemented resulting from them.

4.2.3 Process Layer

The Process layer specifies the interactions between different components of the International Data Spaces. Thanks to that, this layer provides a dynamic view of the Reference Architecture Model.

Some examples of these interactions are Onboarding (how to be granted access to the IDS as a Data Provider or Data User) and Exchanging data (how to search for a suitable Data Provider and how to invoke the actual data operation).

4.2.4 Information Layer

This layer specifies the Information Model, IDS's domain-agnostic, common language [9].

Its primary purpose is to enable (semi)automated exchange of digital resources within a trusted ecosystem of distributed parties while preserving the data sovereignty of Data Owners [9].

The IDS Information Model was presented by the IDS Reference Architecture Model in 2019 and consisted of three levels of representations: IDS Reference Architecture Model, IDS Ontology and IDS Information Model Library, as depicted in figure 4.3 on page 28.

The IDSA publishes and maintains each representation separately, which allows for different abstractions. The IDS Reference Architecture (Conceptual Representation) is the most generic and descriptive, with a high-level overview of the main concepts, and the IDS Information Model Library is the most specific and executable. The IDS Ontology (Declarative Representation) is the only normative specification of the Model and uses the machine-interpretable specification of concepts based on a stack of W3C Semantic Web technology standards. The less abstract level, IDS Information Model Library (Programming Representation), targets Software Providers and maps the IDS Ontology onto native structures of a target programming language.

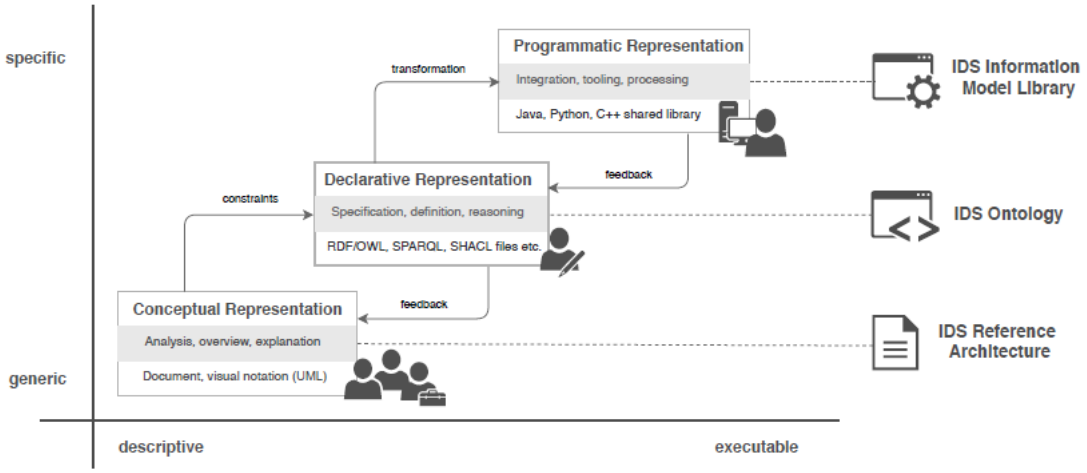


Figure 4.3: Information Model Representations [9]

4.2.5 System Layer

On the System Layer, the roles established on the Business Layer are mapped onto a concrete data and service architecture to meet the requirements specified in the Functional Layer. Three major technical components are specified: Connector, Broker, and App Store. Each component is described in detail in section 4.3.

Figure 4.4 depicts the interactions between the core components. The Data Sources and Data Sinks are always connected to a connector (the data provider or data consumer). Moreover, the connectors can communicate with each other as well as with the Broker and App Store.

They communicate with the App Store by downloading apps for data manipulation.

Communication with the Broker is made by exchanging metadata describing the available data sets.

Finally, to communicate with each other, the connectors exchange data sets and metadata (if a Metadata Broker is not used).

It is important to note that a peer-to-peer network concept transfers data between the Data Provider Connector and the Data Consumer Connector.

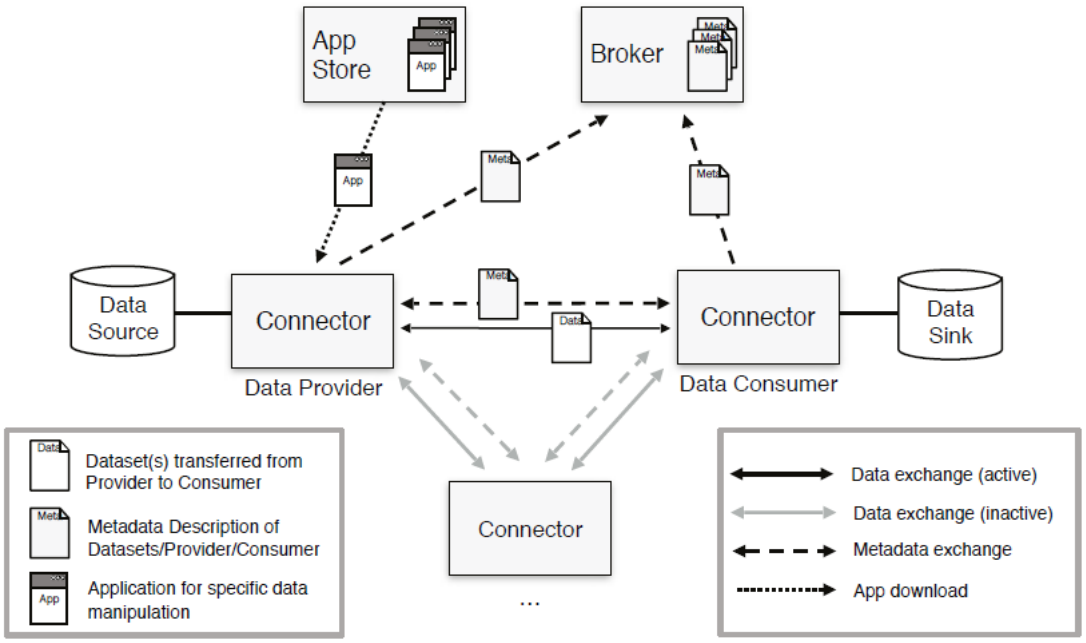


Figure 4.4: Interaction of technical components [9]

4.2.6 Security Perspective

The Security Architecture provides means to identify participants, protect communication and data exchange transactions, and control the use of data after it has been exchanged [9].

Below is described how the different ID-RAM layers address the Security aspects.

They are crucial for defining roles and basic interaction patterns in the Business Layer of International Data Spaces.

The security requirements may restrict or prevent certain transactions and operations on the Functional Layer. However, many use cases would be impossible without security (e.g., offering sensitive data to trusted business partners). Here enters the concept of data usage control which allows Data Providers to attach data usage policies to their data to define how a Data Consumer may use said data.

The security aspects are addressed on the Process layer by constantly monitoring, validating and redesigning existing processes.

The Information Layer provides the means for Participants to use a common vocabulary and standard semantics to express concepts and relationships between them. In doing so, it is possible to specify access and usage control policies understandable to all Participants.

The System Layer defines the Connector as the central technical component. Therefore, the security features are predominantly implemented on this component.

4.2.7 Certification Perspective

The certification process is divided into three phases:

Application Phase the main goal of this stage is for the IDS evaluation and certification process successful start.

Evaluation Phase This stage's primary goal is to evaluate an applicant or core component based on the defined evaluation criteria.

Certification Phase Its central goal is to examine the evaluation report by the certification body, which issues a certificate if the evaluation process results are positive.

After completing the evaluation, the Certification Body awards the applicant an International Data Spaces certificate.

4.2.8 Governance Perspective

The Governance perspective defines the requirements to be met by the business ecosystem to achieve a secure and reliable corporate interoperability [9]. Below is addressed how the governance is achieved across all five ID-RAM layers.

On the Business Layer, the governance is accomplished by considering the business point of view concerning data ownership, data provision, and data consumption and by describing core service concepts, such as data brokerage.

On the Functional Layer by guaranteeing interoperability and connectivity with the finality of supporting trust, security, and data sovereignty. The relation with governance is evident in specific IDS components, such as Clearing House and the Identity Provider. However, the functionality of specific technical core components (e.g., the App Store or the Connector) also relates to the Governance Perspective.

The Process Layer achieves governance since the processes described there define its scope regarding the technical architecture.

On the Information Layer, governance is achieved by the vocabulary developed there. It plays a vital role in this perspective because of its relevance for describing data by metadata in the International Data Spaces.

The System Layer is related to governance due to its technical implementation of different security levels for data exchange between the Data Endpoints in the IDSs.

4.3 Components

4.3.0.1 Connectors

The success of trusted data ecosystems requires the provision of IDS connectors as a central component for secure and trusted data exchange [12].

As the main technical component for data exchange, the IDS connector has different variants from different vendors [7]. Nonetheless, they all should be able to communicate with each other. Depending on its implementation, a connector can be described as Base Free, Base, Trust or Trust+ connector [9]. The IDSA gives that classification during the certification process, and the

difference between them is that each requires stricter security and trust requirements than the last one [3]. We can choose the connector that better meets our data protection requirements and usage scenarios based on those security levels.

The IDS Connector's primary function allows data owners and providers to exchange and share their data with other participants in the IDS ecosystem while ensuring data sovereignty [9]. It is also responsible to connects all kinds of data clouds, platforms, and marketplaces [7]. This component is the entry point to the IDS.

The Connector keeps the data control in the hands of the DP. It allows him to define precisely how often, which data and at what price the data can be used and if it can be stored by the data user or a third party.

To help vendors develop connectors, the IDSA offers sample code, reference implementations, and testbeds, including test services for interoperability testing, technical specifications, a starter kit and a co-creation platform in the form of a developer community [12]. The IDSA and Fraunhofer provide four open-source connectors to use or create upon them: Dataspace Connector[2], Trusted Connector[3], Eclipse Dataspace Connector[4] and TRUE Connector[5]. Other connectors from different vendors, such as the IDS Open Data Connector[6] and FIWARE TRUE Connector[7] are also open-source.

Regardless of the vendor, all connectors have the same data model based on the IDS Information Model structure, depicted in figure 4.3 on page 28. The connectors data model is shown in figure 4.5.

On the data model, it is possible to see that a Catalogue is the first structure to create, and can have various resources (requested and offered).

The Offered Resource is the central structure since data cannot be provided without it. This resource is traded and exchanged between the IDS connectors and is only completed if it has at least one contract and at least one representation with one artefact.

A Representation contains the abstract content of a Digital Resource. Furthermore, it has at least one artefact, which can be associated with multiple resources.

The Artefact is the materialization or instance of a Representation. Therefore, it must be connected to a representation and have data.

A Contract comprises a set of Rules that describe permissions or obligations. It needs to have at least one rule, which can be associated with multiple resources.

A Contract Rule defines a Usage Policy that describes permissions or obligations. It can be associated with various contracts and resources.

The Agreement has all the usage information between a data provider and a data consumer.

Some connectors are described in more detail below:

[2] https://github.com/International-Data-Spaces-Association/DataspaceConnector

[3] https://github.com/Fraunhofer-AISEC/trusted-connector

[4] https://github.com/eclipse-dataspaceconnector/DataSpaceConnector

[5] https://github.com/International-Data-Spaces-Association/true-connector

[6] https://www.dataspaces.fraunhofer.de/en/software/connector/open_data_connector.html

[7] https://github.com/Engineering-Research-and-Development/fiware-true-connector/

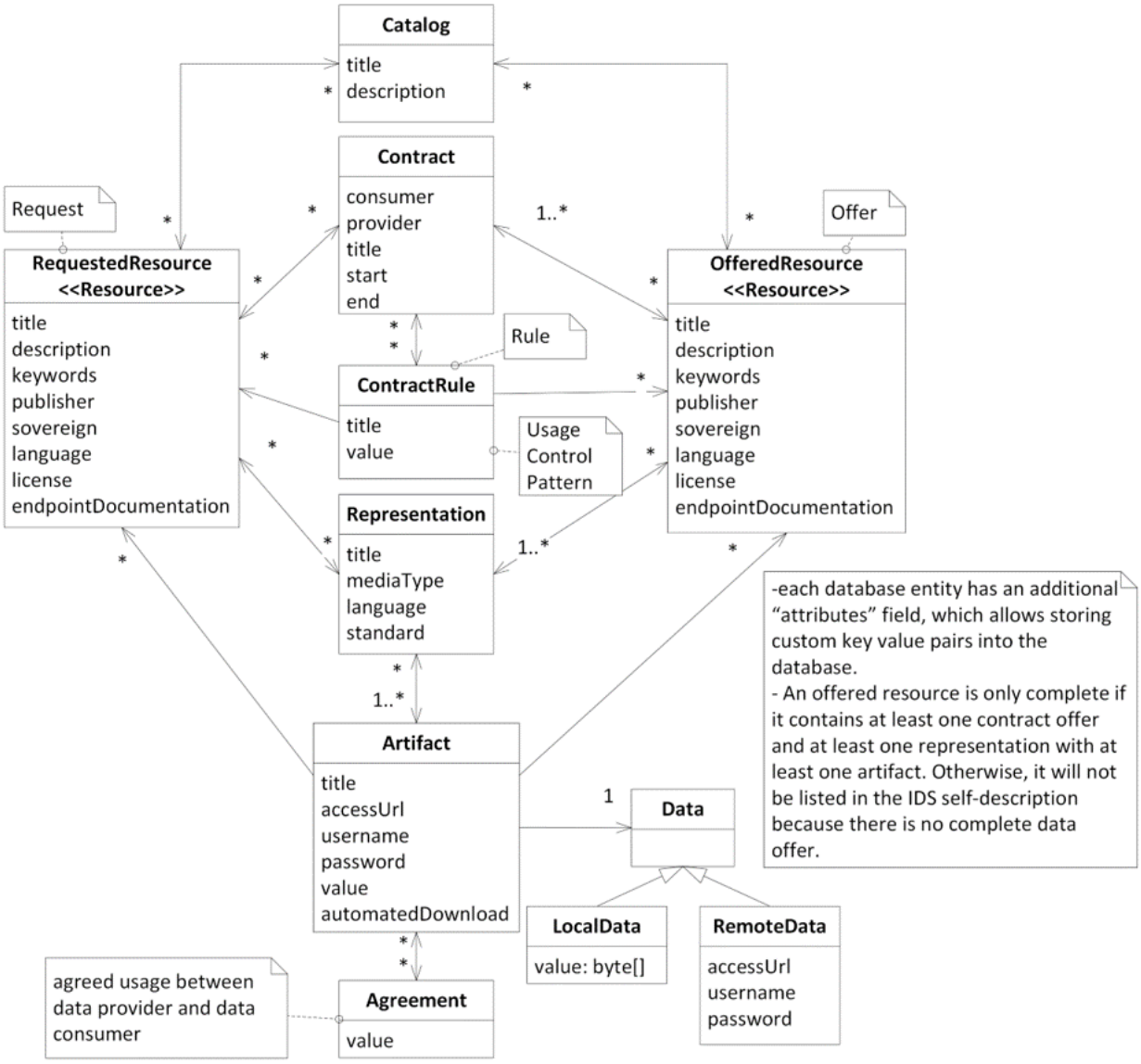


Figure 4.5: Data Model

Open Data Connector is an out-of-the-box solution that allows open data sharing, free of charge, to be used and re-used.

DSC (Dataspace Connector) is a connector developed by Fraunhofer-ISST that enables extending data with policies and conditions that can be created, processed, controlled and enforced[8]. It is open-source, and right now, it implements nine basic usage control class policies. This connector allows the integration of existing software with the International Data Spaces. Furthermore, as this is an open-source, heavily documented (with detailed information about integrating other IDS core components) and easy-to-use solution, it is one of the more used implementations.

Eclipse Dataspace Connector[9] is the newest open-source project, and it started on April 29. It wants to provide a framework for a sovereign, inter-organizational data exchange, and it will implement the IDS standard and Gaia-X relevant protocols.

TRUE (TRUsted Engineering) Connector is a connector for the IDS ecosystem, and already comes pre-configured. However, it also has documentation to help modify it. An important note is that the usage control functionalities can be configured.

FIWARE TRUE (TRUsted Engineering) Connector (FTC) enables integration between Fiware and IDS ecosystems. The connector is easily customized to fit a widespread of scenarios[10].

The Trusted Supplier Connector (TSC) is the connector used on Gaia-X[11] project [11]. It was developed by the company German Edge Cloud for the ONCITE - Sharing Data in the Supply Chain [12] project, in partnership with the IDSA. The ONCITE project focuses on the sharing of data in the supply chain.

The Trusted Connector has a security certification level of Trust. One of its main features is the IDS Communication Protocol (IDSCP) support, which binds user data to negotiated usage policies and integrity verification[13]. Another prominent feature is the usage control, implemented using LUCON. This connector can be installed using Docker, and a prototype to test the connector with some basic usage control functionalities[14] is given. This is one of the oldest and more used connector implementations.

Table 4.1 specifies if a connector has any usage control functionalities, even if it does not yet implement all classes.

4.3.0.2 Data Apps

Data Apps are data services encapsulating data processing or transformation functionality bundled as container images for simple installation by application container management [9]. They can be

[8] https://internationaldataspaces.org/usecases/fraunhofer-isst-dataspace-connector/
[9] https://github.com/eclipse-dataspaceconnector/DataSpaceConnector
[10] https://fiware-true-connector.readthedocs.io/en/latest/
[11] https://www.gaia-x.eu/
[12] https://internationaldataspaces.org/usecases/german-edge-cloud/
[13] https://www.dataspaces.fraunhofer.de/en/software/connector/trusted-connector.html
[14] https://industrial-data-space.github.io/trusted-connector-documentation/docs/getting_started/

Table 4.1: Usage Control implementation within Connectors

Connectors	Usage Control
Trusted	Yes
Dataspace	Yes
Open Data	No
Trusted Supplier	Yes
TRUE	Yes
Eclipse Dataspace	In the future
FIWARE TRUE	Yes

deployed in the IDS connectors to execute tasks like transformation, aggregation, or analytics on the data.

A Data App must comply with the International Data Spaces system architecture to be deployable. In addition, these applications may be certified by IDS-approved certification bodies in order to increase trust in them. The app developers must publish each Data App in the App Store for being accessed and used by Data Consumers and Data Providers.

The App Providers are the ones that develop Data Apps. They should describe each one using metadata (in compliance with a metadata model) concerning its semantics, functionality, and interfaces.

There are three types of Data Apps [9]:

- **Self-developed Data Apps:** are Data Apps used by the DP's own connector (they usually do not require any certification from the Certification Body);
- **Third-party Data Apps:** apps retrieved from the App Store;
- **Data Apps provided by the DC connector:** are Data Apps that allow the DP to use certain functions before data is exchanged (e.g., data aggregation or filtering)

In addition, as described in the IDS Reference Architecture document, Data Apps can be divided into three categories:

- **System Adapters:** Data Apps on the DP side. Their task is to connect all types of data sources or data sinks and make them available to the underlying connector. It includes, for instance, connections to databases, web services or Big Data systems. Furthermore, system adapters can transform data models from connected data sources into a standardized format. Additionally, this type of application can also enrich connected data with additional metadata[15].
- **Smart Data Apps (or Data Sink Connectors):** Data Apps on the Data Consumer side, with the task of executing any data processing, transformation, or storage functionality. It includes all applications that somehow manipulate the data flow within a connector. Examples include applications for data quality assurance or machine learning. The data that this

[15] https://international-data-spaces-association.github.io/IDS-AppStore/data-app

kind of data app processes and then makes again available is usually already enriched with metadata[16].

- **Other Apps:** They provide a particular use of the data on the Data Consumer or Data Provider side. Usage Policies can enforce the data processing in a trusted environment, i.e., a Trusted Connector.

An example of a Data App is the DataspaceApp4EDI described in section 5.3.

4.3.0.3 App Store

The IDS App Store is a secure platform for distributing Data Apps [9].

Data apps can process or transform data before or after the actual data exchange and can be deployed inside an IDS Connector. One of their main goals is to facilitate data processing workflows. Each Data App and corresponding metadata information must be published in the App Store to access and use. The App Store is then responsible for managing information about Data Apps. Therefore, it should provide interfaces for publishing and retrieving Data Apps and their corresponding metadata[17].

The corresponding metadata is stored in the App Store for each data app that was successfully certified. Via a search interface, suitable data apps for one's use can be identified[18]. The IDS App Store features various search options (e.g., by functional or non-functional properties, pricing model, certification status, community ratings, ...) [9], and suitable data apps can be requested.

Essentially, an IDS App Store consists of a registry for available Data Apps in the specific App Store. So it supports the registration of available Data Apps. Therefore, an App Store supports operations for registering, publishing, maintaining, and querying Data Apps and operations for making a Data App available to a connector. These basic operations can be complemented by additional services such as billing or support services [9].

App stores can be provided by IDS members and must be separately certified under IDS standards. [19]

4.3.0.4 Identity Provider

Identity Providers (IdP) offer services to create, maintain, manage, monitor and validate information from and of participants in the IDS [9]. They are crucial since they avoid unauthorized access to data [9].

In short, the Identity Provider verifies the authenticity of all the actors involved in the architecture and provides all the characteristics related to identity management [8]. These characteristics include actor registration, authentication, password management and the option of grouping actors

[16] https://international-data-spaces-association.github.io/IDS-AppStore/data-app
[17] https://international-data-spaces-association.github.io/IDS-AppStore/introduction
[18] https://international-data-spaces-association.github.io/IDS-AppStore/introduction
[19] https://internationaldataspaces.org/use/ids-components/

in organizations to manage them under identical conditions [8]. This component is responsible for issuing technical identities to parties that have been approved to become Participants in the International Data Spaces [9].

If a single Data Provider shares its data with more than one Data Consumer, it can have its own IdP since there is no need to validate the identity of other Data Providers [8].

Another example of an IdP responsibility is creating, maintaining, and revoking an IDS identity for the DP and DC connectors.

The Identity Provider consists of:

- Dynamic Attribute Provisioning Service (DAPS): to manage the participants' dynamic attributes;
- Certification Authority: to manage the digital certificates for the IDS participants; and,
- Dynamic Trust Monitoring service (DTM): for continuous monitoring of the security and behaviour of the network [9].

4.3.0.5 Metadata Broker

The IDS Metadata Broker is an informative component that provides functionalities of registry and discovery metadata of connectors and resources for the IDS participants and components [5].

Its goal is to index metadata of individual datasets and thereby enable data distribution by offering high-performance search opportunities in the IDS ecosystem.

So, a Broker is a query-able registry that allows users to register/update/delete the metadata of their datasets. It uses the SPARQL query language, a standardized query language in the semantic web. The IDS Metadata Broker's main task is to provide search functionalities for data offerings. Because of that, it is necessary to have a certain openness to the metadata provisions. However, certain Metadata Broker implementations can consider usage restrictions for the received metadata. The reason for that change is that, in some instances, the knowledge of the resource's existence can threaten the data sovereignty. An example of that behaviour is a Broker hiding the existence of a company's connector to its direct competitor, even though it provides that information to the company's own suppliers and customers [5].

Its behaviour ensures individual data privacy and sovereignty, thus leading to users' trust, a key element in the IDS.

The broker is an optional component since no broker is necessary if the connectors know each other in advance. Multiple brokers in the IDS can also exist since they can have specific purposes, e.g. supply chain broker and mobility broker[20].

It is significant to note that the data transfer, except metadata, between the DP and DC does not involve a broker. Moreover, given that the communication between a Broker and a Connector

[20] https://www.isst.fraunhofer.de/en/business-units/data-business/projects/MobiDS.html

is message-oriented, there are two categories of broker messages: *publishing messages* (delivery of metadata to the index services) and *query messages* (query of metadata from the index service).

A practical example of the use of a broker is when a subcontractor wants to exchange data with a producer. In this case, as the subcontractor already knows the name of the producer company, he can query the supply chain broker to know the producer connector ID. That ID will then be used to communicate with the intended connector.

4.3.0.6 Clearing House

This IDS component is a trusted third party and acts as an Intermediary in the IDS ecosystem. In other words, the CH mediates between a Data Provider and a Data Consumer to guarantee that both parties meet their contractual obligations, namely: The DP sharing data with the Data Consumer agrees with the Usage Contract and Data Usage Policies. The Data Consumer uses data just as the Usage Contract and Data Usage Policies defined and effecting payment to the DP as agreed.

For each data exchange transaction, the DP attaches metadata to the DC requested data, specifying some attributes, e.g., data usage restrictions, pricing information, and duration of data availability. This metadata guarantees data sovereignty by ensuring the Data Provider can specify a Data Usage Policy as deemed appropriate [2].

The CH primary purpose is to serve as an independent entity in the data exchange. As an independent and trusted entity, its use can reduce the risks and uncertainty of sharing and exchanging data by providing trust between components that did not necessarily trust each other. Companies do not have trust either because they do not know each other or have not done data exchange transactions yet. Some risks that the use of a Clearing House can mitigate are the contract unfulfillment (e.g. the data the DP is sharing does not have the quality discussed, or the DC lack of payment); and the usage contract violation (e.g., the DC sharing data with third parties when that action was not agreed on the contract). The participants on a data exchange can choose to use a Clearing House or not, and the Clearing House that each participant uses does not need to be the same.

The CH interfaces with and relies upon the Identity Provider, Dynamic Trust Management (DTM), and Policy Enforcement Points (PEP) to provide the functions for managing data-sharing and payment processes after the Data Provider and the Data Consumer have made a mutual agreement. The Identity Provider is responsible for providing services to create, maintain, manage and validate identity information, which are essential to avoid unauthorized access to data. The Dynamic Trust Management provides services for continuous network security and behaviour monitoring. Furthermore, the PEP enforces data usage restrictions since the system's actions need to be monitored and potentially intercepted by control points [2].

The IDS Clearing House provides two essential functions for all data and financial exchanges: clearing and settlement based on transaction logging [2].

More specifically, the clearing function is executed prior to data sharing. It is responsible for verifying the usage contract and usage policies (Legal function), verifying the payment conditions (Financial function) and enabling transaction execution and binding transaction to an instance of

a data-sharing agreement and Usage Contract (Technical function). An example of the latter is the non-repudiation, meaning that the subcontracted cannot deny receiving the data, nor can the producer deny sending the data.

During the data-sharing process, the monitoring and logging functions are executed. The IDS CH can be invoked for logging metadata and querying the previously stored items, e.g., for data provenance tracking or monitoring and discharging the data-sharing transaction. The settlement functions prior to and during the data-sharing process are:

- Data-sharing transaction discharging;
- Transactions metadata logging;
- Tracing data provenance;
- Data transactions monitoring and reporting;
- Data transactions auditing and tracking for determining accountability and resolving possible conflict;
- Data transaction billing and invoicing.

Finally, after exchanging data, the administrative part of the transaction must be addressed, including logging, billing, and invoicing. The CH charges a fee for conducting these functions and absorbing the risk of not fulfilling the contract between both parties. The settlement functions, after sharing data, or in case of not sharing any data, for conflict resolution, are [2]:

- Investigating claim and enforcing action upon violation of Usage Contract or Data Usage Policy;
- Escalate to a court (Legal function);
- Block a participant via Identity Provider or downgrade its degree of trust using DTM (Technical function);
- Request financial compensation (Financial function).

The metadata artefacts required from other services to execute the clearing and settlement functions required above are listed below:

- **Legal Contract:** states legal aspects required for (or to avoid) conflict resolution;
- **Commercial Contract:** States commercial conditions under which specific data will be provided, including price, invoicing, and payment conditions. An example can be the data being free and having no invoicing and payment conditions;
- **Service Levels:** States quality parameters of data provided, such as completeness, accuracy, or timeliness;

- **Usage Control Policy:** States how data may be used or distributed after granted access to individuals, roles, or systems. An example can be the possibility of the subcontracted to save the bill of materials;
- **Access Control Policy:** States which individuals, roles, and or systems are granted access to data provided. An example can be that only the subcontracted team can have access;
- **Usage Contract:** Combines **Access Control Policies** and **Usage Control Policies**; expresses DP's internal (business) data-sharing policies and external (regulatory) policies. An example can be that each bill of materials can only be shared for five days;
- **Contractual Conditions:** Combines **Service Levels, Usage Contract, Legal Contract**, and **Commercial Contracts**;
- **Data-Sharing Agreement:** Species conditions under which specific data will be shared; consists of **Contractual Conditions** and **Usage Contract**;
- **Data Transaction:** Represents a specific instance of sharing primary data, including data request, data response, and associated processes for management and administration.

An example of a Clearing House use case is given on the IDS Clearing House White Paper [2] and consists of a contract violation. Its description defined it as purchased data not being delivered or delivery violates contract (e.g., quality of data not as specified). The actors involved are Data Provider (Producer), Data Consumer (Subcontracted), Clearing House (CH), and Dynamic Trust Management (DTM). Its preconditions are that the Usage Contract and Commercial Contract exists, and the claim management system must be available to report claims.

Its basic flow is:

1. Usage Contract and Commercial Contract have been successfully agreed upon by Producer and Subcontracted
2. The transaction has been commercially cleared and settled
3. Data is not delivered as specified in the contract / Data is not delivered at all
4. Subcontracted raises a claim to CH about non-fulfilment of the contract
5. CH uses logging information (metadata) to verify the validity of the claim:

   a) If a claim is invalid, CH declares the claim unjustified.

   b) If a claim is valid, CH starts clarification with the Producer for correction of delivery or financial compensation.

   c) If the claim cannot be resolved, the issue may be escalated to DTM to reduce the Producer's trust level.

4.4 Summary

The purpose of this chapter was to present the International Data Spaces, its architecture and its components. Here, the IDS goals and requirements were discussed and given a detailed explanation of the IDS-RAM five layers, as depicted in figure 4.1 on page 25.

The IDS's main goals are to guarantee data sovereignty, to be applied across company borders and existing supply chains and to establish an international standard (IDS-RAM) for data sharing and exchange.

It aspires to meet six requirements: *Trust* is the International Data Spaces basis and is achieved by guaranteeing that every IDS participant is certified by a trusted entity before being granted access to the business ecosystem. *Security and data sovereignty* is carried out by evaluating and certificating each technical component and attaching usage restrictions to the data exchanged. An *Ecosystem of data* follows the concept of decentralized data storage, accomplished by eliminating central data storage capabilities, which makes the data stay with the data owner until it is time to transfer it to the data user. *Standardized interoperability* assures that the connector variant or vendor does not interfere with the connector's ability to communicate with other connectors or technical components. *Value-adding apps* supply services on top of data exchange processes by injecting applications into IDS connectors. *Data markets* enable the creation of novel data-driven services that use data apps and promote new business models for these services.

The International Data Spaces Architecture Model is made up of five layers: The *Business Layer*, centred on the roles the IDS participants can have and their interactions with each other (Core Participants, Intermediary, Software/Service Provider, and Governance Body). The *Functional Layer* defines the IDS functional requirements and features that result from them. The *Process Layer* establishes the interaction between the different International Data Spaces components, with some examples to facilitate the implementation of the data spaces. The *Information Layer* specifies the information model with all its different abstractions: IDS Reference Architecture or Conceptual Representation presents a high-level overview of the main concepts, with no commitment to a particular technology or domain, and as such, it is demonstrated through textual documents and visual notation, e.g. UML diagrams [9]; IDS Ontology or Declarative Representation implemented based on a stack of W3C Semantic Web technology standards and standard modelling vocabularies, e.g. DCAT and ODRL; and, IDS Information Model Library or Programming Representation which is implemented using programming languages, e.g. Java or Python. The *System Layer* is responsible for decomposing the IDS components, considering aspects such as integration, configuration, deployment, and extensibility. Furthermore, the Reference Architecture Model comprises three perspectives that must be implemented across all five layers: Security, Certification, and Governance.

This chapter also describes the IDS components: Connector, Metadata Broker, App Store, Data Apps, Clearing House, and Identity Provider. The *Connector* is the IDS central technical component and is responsible for every communication inside the IDS ecosystem; without a certified connector, it is impossible to participate in a data space. The *Metadata Broker* is an IDS core

component, and it is a central metadata catalogue that describes all the data sources in the network and the respective IDS Connectors that provide access to them. The *App Store* is the last IDS core component; it consists of a repository for available Data Apps that allows software providers to install and configure data applications for their customers. A *Data App* can be deployed inside a connector to facilitate the data process workflows; they can be used to process or transform data before or after the actual data exchange. The *Clearing House* can be considered a central auditing and logging system, responsible for keeping a copy of the data usage contracts agreed by data providers and data consumers and registering all the data flows between the respective IDS Connectors. The *Identity Provider* ensures the digital identity and authenticity of participants and connectors in the network and guarantees secure and encrypted communications between them.

Chapter 5

Implementation

The IDSA, along with the Fraunhofer-AISEC and Fraunhofer-ISST, are developing prototypes for the IDS components. Those prototypes are available for everyone to use or personalize to fit their data spaces and specific needs.

With the prototypes in mind, this chapter focuses on implementing the IDS components to create a self-hosted data space controlled by INESC TEC. It is divided into sections, with each section starting with a brief description of the IDS component, followed by the steps for its implementation, their test using an existing test architecture, the challenges that arise when testing, and the experiments made.

The problems were only registered and not solved due to the INESC TEC objective of not implementing a customized component version but using the base components on their data spaces. Their goal is to substitute the Fraunhofer and IDSA-hosted components with their own, with only the basic configuration documented on the Dataspace Connector documentation page. With that purpose in mind, every challenge or bug encountered, given that the components are still under development, was registered and created an issue on the component GitHub page (if there was not one already).

These components were implemented and tested on the same virtual machine, *vcese18.inesctec.pt*, and, when stable, swapped to *vcese19.inesctec.pt* VM. The only exception is the existing architecture connectors, already implemented on *vcese11.inesctec.pt* and *vcese12.inesctec.pt* VMs.

5.1 Test Methodology

The experiments developed and implemented were based on both the DSC and the IDS component available documentation, which means that every feature described on the DSC page was tested, as well as the features described on the implemented component documentation. Then, it was seen if all features were implemented and if there were any discrepancies between both documentations.

Also, experiments based on the use case in which the existing architecture is based were developed to see if the IDS component is ready to be implemented in a production environment alongside the environment described in section Existing Architecture.

This approach results in specific tests for each component, meaning that the type of tests made is particular to each component and its interaction with the environment described in section 5.3.

5.2 Intended Architecture

The infrastructure architecture implemented on this chapter is depicted in figure 5.1. All IDS components described are presented, as well as the IDS Connectors and App4EDI already implemented.

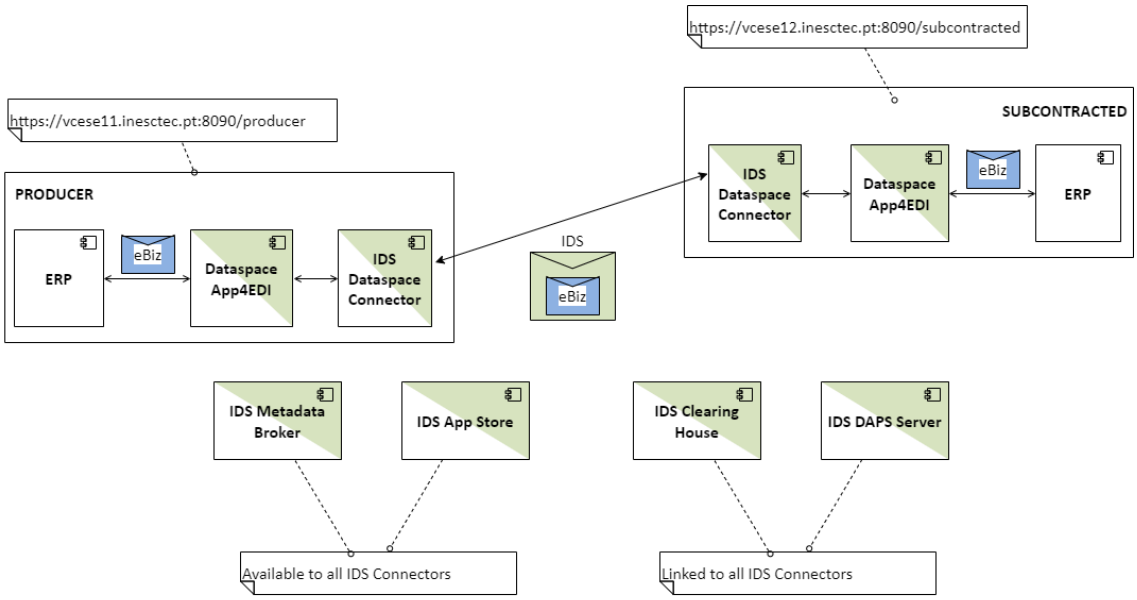


Figure 5.1: Intended Architecture

5.3 Existing Architecture

Regarding testing and implementing the IDS components studied in chapter 4, the first step was to study the existing architecture depicted in figure 5.2. This environment was based on a production environment from the *STVgoDIGITAL* project detailed in section 2.4, but with the companies' names removed. The project's scope is to make available production orders and production status from an industrial organization to its subcontractors.

On figure 5.2, it is possible to see that the producer has an ERP that uses the eBiz document type (not relevant to this study). To share its documents with a subcontracted, he needs to go to the App4EDI Swagger page, shown in figure 5.4 on page 46. The application then shares the document with the IDS Connector, which then shares it with the Subcontracted connector that shares it with the subcontracted app4EDI. This process can also happen in the opposite direction, from the subcontracted to the producer.

Regarding its implementation, this architecture consists of two separated environments, one for the producer (located on virtual machine *vcese11.inesctec.pt*) and one for the subcontracted (located on VM *vcese12.inesctec.pt*). Each environment consists of a Connector and a Data App.

The connectors used for this environment are the Dataspace Connectors, described in section 4.3.0.1.

The App IDS, called DataspaceApp4EDI, is a project created by INESC TEC, and its primary responsibility is to share documents with a connector and do a contract negotiation exchange.

This environment is constantly pushing data between the producer and the subcontractor.

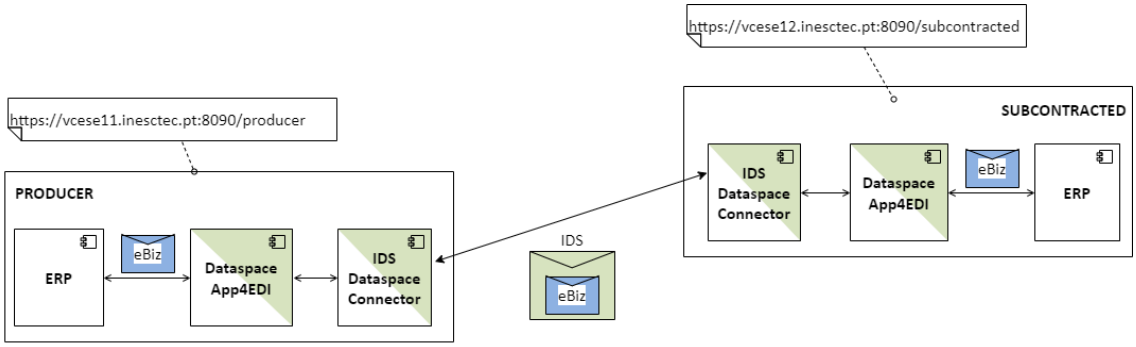


Figure 5.2: Existing Architecture

5.3.1 Connectors

The IDS connectors used in environment is the Dataspace Connector. He is responsible for the communication between the producer and the subcontracted.

Figure 5.3 on page 45 shows the Connector Swagger page, where all the available endpoints for the Connector operations are presented.

The endpoints to manage the catalogues, artefacts, representations, resource offers, contracts and rules, which are the Connector data model structure based on the IDS Info model structure, were studied and later on tested.

The page also shows the endpoints to get the Connector self-description and configurations. It also has the necessary endpoints to communicate with the other IDS components studied in chapter 4 and implemented in this chapter.

5.3.2 DataspaceApp4EDI

This IDS App was first developed to run acceptance tests on each new Dataspace Connector release. However, it was then extended to accomplish other things. Such as testing the Connectors' creation of resources (i.e. Catalogues, Artefacts, Representations, Resource Offers, Contracts and Rules) and connection of the ERP systems with the IDS ecosystem by allowing the creating and sending of documents as well as removing the already processes documents by the ERP.

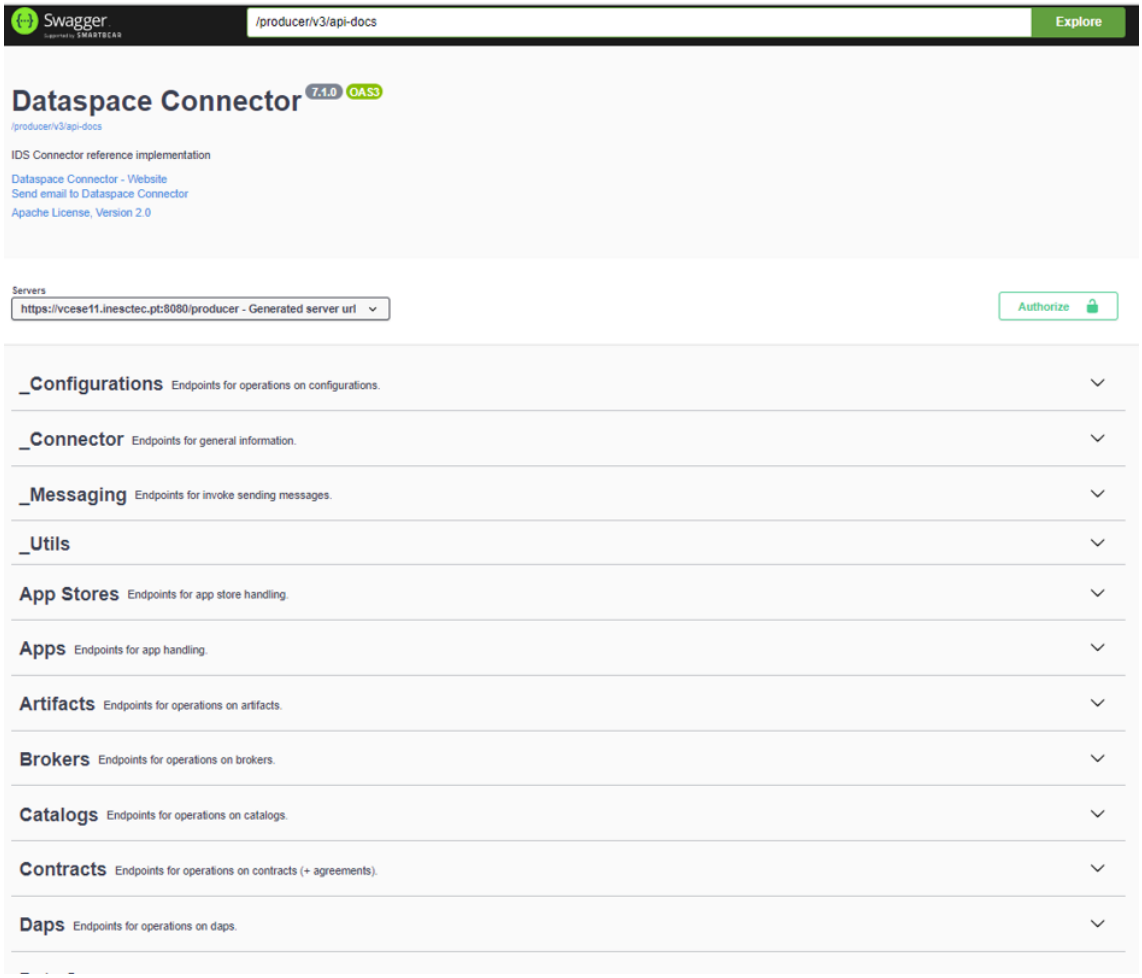


Figure 5.3: DataspaceConnector Swagger Page

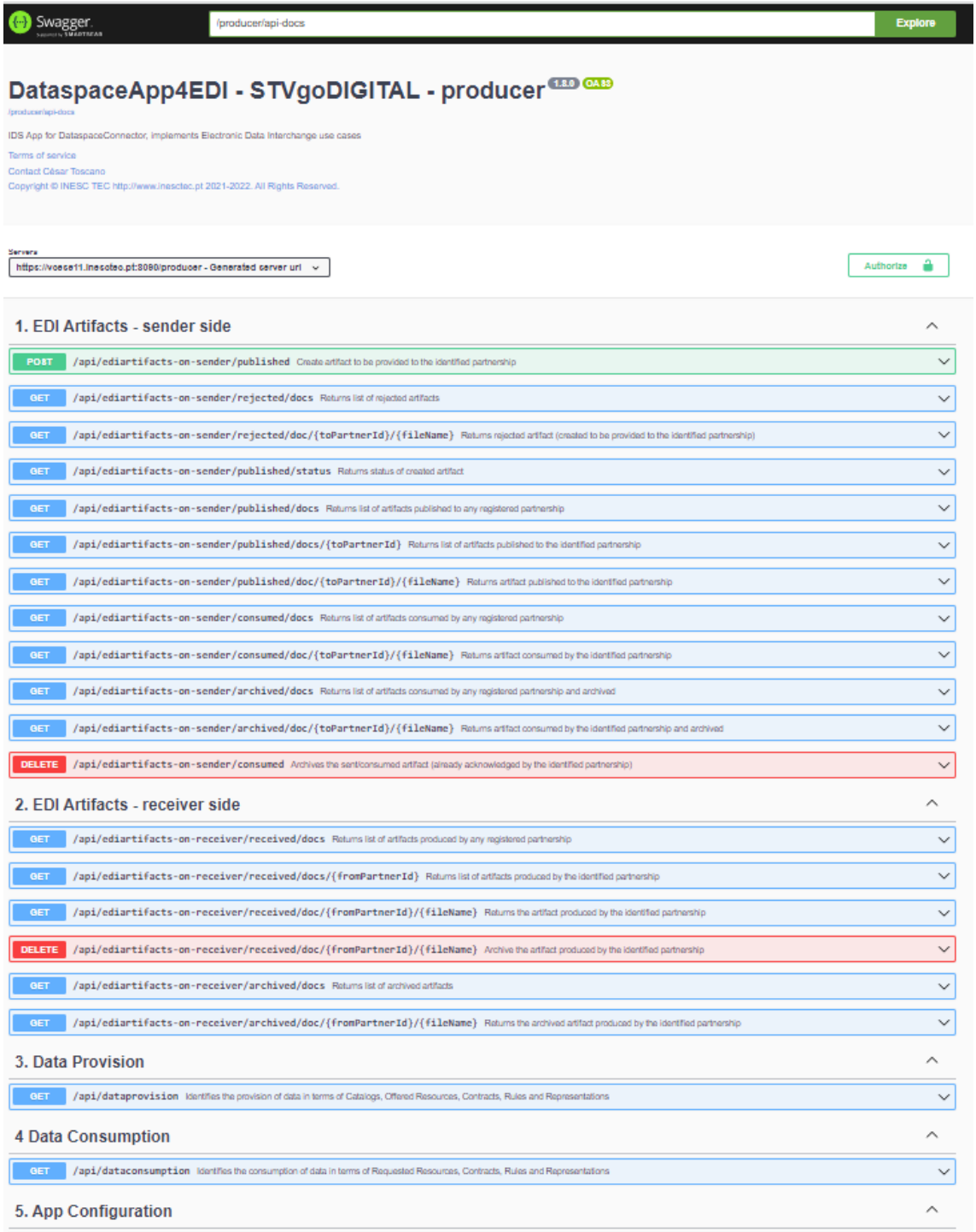


Figure 5.4: DataspaceApp4EDI Swagger Page

Therefore, the application provides a REST interface, depicted in figure 5.4, for ERP-type systems in order to allow:

The ERP issuer (e.g. producer) to:

1. create a document in the eBIZ format and request it to be sent to a specific business partner (e.g. subcontracted):
   - ERP invokes a POST function providing the partner ID, document type and title as parameters. In the message body will be the document in eBIZ format.
2. look for rejected documents (e.g. documents that do not possess the correct extension):
   - ERP invokes the GET function. It will return a list of rejected documents.
3. obtain a specific rejected document, send to a particular business partner, with a specific file name:
   - ERP invokes the GET function providing the partner ID and the file name as parameters. It will return the rejected document.
4. obtain a document status:
   - ERP invokes the GET function providing the partner ID, the file name and document type as parameters. It will return an object with the document parameters (i.e. partner ID, file name and document type) and its status (e.g. consumed by the ERP).
5. obtain a list of documents published to any business partner:
   - ERP invokes the GET function. This function will return a list of documents sent by any business partner.
6. look for documents published to a specific business partner (e.g. subcontracted):
   - ERP invokes the GET function providing the partner ID as a parameter. It will return a list of documents published to a certain business partner.
7. look for documents published to a specific business partner (e.g. subcontracted) and with a certain name (e.g. 2022-05-10T15_39_50Z-producer-subcontracted):
   - ERP invokes the GET function providing the partner ID and file name as parameters. It will return the file's contents with the name given in the parameters.
8. obtain a list of documents consumed by any business partner:
   - ERP invokes the GET function without parameters. It will return a list of artefacts consumed by any registered business partner.
9. obtain a document consumed by an identified business partnership and with a certain name:
   - ERP invokes the GET function providing the partner ID and file name as parameters. It will return the content of the file with the name given in the parameters.

10. look for a list of documents consumed and already archived by any business partner:
    - ERP invokes the GET function without parameters. It will return a list of artefacts consumed and archived by any registered business partner.
11. obtain a document consumed and archived by an identified business partnership, and with a specific file name:
    - ERP invokes the GET function providing the partner ID and file name as parameters. It will return the content of the file with the name given in the parameters.
12. delete the sent and consumed document:
    - ERP invokes the DELETE function providing the partner ID, file name ad file type as parameters.

The ERP receiver (e.g. subcontracted) to:

1. obtain a list of documents sent to him:
    - ERP invokes the GET function. This function will return a list of documents received, regardless of the business partner that sent them.
2. look for documents sent by a particular business partner (e.g. producer):
    - ERP invokes the GET function providing the partner ID as a parameter. It will return a list of documents sent by the business partner defined in the parameter.
3. look for documents sent by a specific business partner (e.g. producer) and with a certain name (e.g. 2022-05-10T15_39_50Z-producer-subcontracted):
    - ERP invokes the GET function providing the partner ID and file name as parameters. It will return the content from the file defined in the parameters.
4. archive already processed documents:
    - ERP invokes the DELETE function providing the partner ID and file name as a parameter. It archives the document to later appear on the listing of archived documents.
5. look for a list of archived documents:
    - ERP invokes the GET function without parameters. It will return a list of archived documents.
6. obtain a archived document produced by an identified business partner, and with a specific file name:
    - ERP invokes the GET function providing the partner ID and file name as parameters. It will return the content from the file with the name given in the parameters.

5.4 Clearing House

The Clearing House is a trusted third-party responsible for storing every communication between the IDS Connectors, from contract negotiation to data exchange. Plus, all these communications are stored encrypted.

The connector implementation used on this test environment, Dataspace Connector, can log to the Clearing House the finalized contract agreements, data usage (if registered in a usage policy), and incoming and outgoing messages (request and response).

The Fraunhofer-AISEC is the centre responsible for creating a Clearing House prototype[1]. It currently implements a Logging Service that consists of two parts, the *Clearing House App* and *Clearing House Processors*.

The *Clearing House App* implements the CH business logic. It comprises three services: *Keyring API service*, responsible for creating keys and for the actual encryption and decryption of stored data. *Document API* with the responsibility of storing the data. *Logging Service* responsible for responding to a logging request by signing a receipt.

The *Clearing House Processors* is used to configure the routes for the trusted Connector by integrating the *Clearing House App* into the Trusted Connector. This Connector is the entry point for all communications involving the Clearing House.

5.4.1 Implementation/Installation Guide

The Clearing House implementation starts with the creation of a docker-compose. There, all necessary containers, environment variables and bind mounts[2] are instantiated. Appendix A shows the final docker-compose.

Not only is it necessary to create a docker container for each service (*Keyring API*, *Document API*, *Logging Service* and *Trusted Connector*), but it is also required to create mongo containers for the *Clearing House App* services. After creating the database containers, creating and instantiating the docker bind mount was necessary. The information about each container bind mount is found on Appendix A.

The environment variables instantiated for the *Trusted Connector* were the *TC_DAPS_URL* (URL to DAPS server), *TC_KEYSTORE_PW* (password to bind mount Keystore file), *TC_TRUSTSTORE_PW* (password to bind mount Truststore certificate), *TC_CH_ISSUER_CONNECTOR* (ID for the message origin Connector) and *TC_CH_AGENT* (ID for the server agent, which is the agent that initiates the message).

The other containers, *Keyring API*, *Document API*, and *Logging Service*, all have the environment variables *API_LOG_LEVEL* and *RUST_BACKTRACE*.

Furthermore, the *Logging Service* also requires keys to sign the receipts. Those were created using an OpenSSL command:

openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:4096 -outform der -out private_key.der

[1] https://github.com/Fraunhofer-AISEC/ids-clearing-house-service

[2] When a file or directory on the host machine is mounted into a container.

5.4.2 Testing procedures

The Clearing House was implemented on the virtual machine *vcese19.inesctec.pt*. If the previous section was followed, the only steps left to run the component are to have the suitable files on the bind mounts and then run the docker-compose already created:

```
1 docker-compose up -d --build
```

Listing 5.1: Start docker-compose

The existing environment already described in section 5.3 was used to test this component.

In that environment, it was necessary to instantiate an environment variable (*clearing.house.url*) on the "Producer" and "Subcontractor" containers of the docker-compose. It is possible to see the CH working on the existing environment container logs (figure 5.5), on the Clearing House Trusted Connector (figure 5.6 on page 51), and on the Clearing House Logging Service (figure 5.7 on page 52).

Figure 5.5: *Producer* container logs

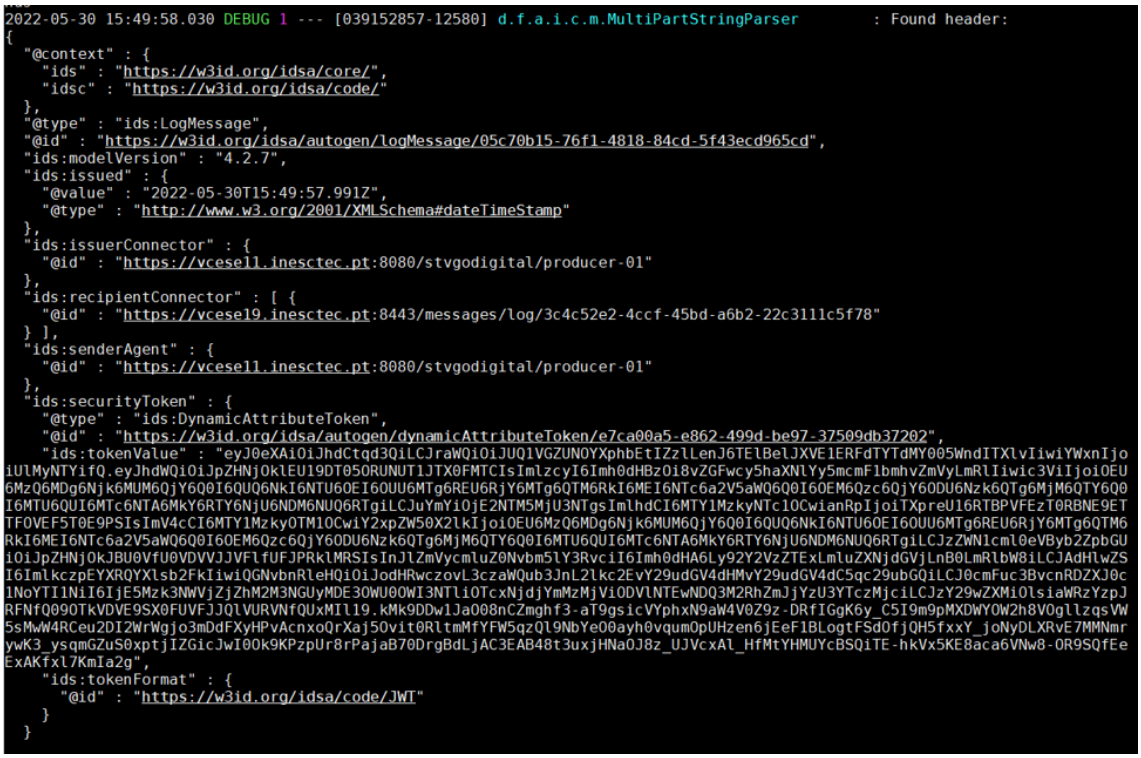


Figure 5.6: *Trusted Connector* container logs

The connector Swagger page is used to query the Clearing House. Below are described the steps to do so:

1. Go to *_Messaging* section
2. **POST** ***/api/ids/query***
   - Recipient ID: `https://<ClearingHouseURL>/messages/query/<contractagreementID>`
   - Request body: ""

The contract agreement ID can be found on the endpoint *Contracts*, on *GET /api/agreements*. Since an agreement always has two parties, a provider and a consumer, there can be two different IDs on the GET contract agreement response. Furthermore, the Dataspace Connector uses the provider-side agreement ID. If only one ID appears, specifically if the field *remoteId* does not have an URL, then the ID used to query the CH is the one found on the *self href*, which means the Connector is the provider. If the field *remoteId* has an URL (so there are two IDs on the contract agreement), the Connector queried is the consumer, and the ID used to query the CH is the one available on the *remoteId*. Figure 5.8 on page 52 represents a contract where the Connector queried is the provider for that contract. On the other hand, figure 5.9 on the page 53 represents a contract where the *remoteId* has an ID, meaning the Connector queried is the consumer for that given contract.

Figure 5.7: *Logging System* container logs

Figure 5.8: Agreement ID on the provider side

```
{
  "creationDate": "2022-05-30T14:00:40.963+0000",
  "modificationDate": "2022-05-30T14:00:40.963+0000",
  "remoteId": "https://vcese12.inesctec.pt:8080/subcontracted/api/agreements/266c365d-474c-4162-bbcf-79fc4606a538",
  "confirmed": true,
  "value": "{\n  \"@context\" : {\n    \"ids\" : \"https://w3id.org/idsa/core/\",\n    \"idsc\" : \"https://w3id.org/idsa/code/\"\n  },\n  \"@type\" : \"ids:ContractAgreement
\",\n  \"@id\" : \"https://vcese12.inesctec.pt:8080/subcontracted/api/agreements/266c365d-474c-4162-bbcf-79fc4606a538\",\n  \"ids:permission\" : [ {\n    \"@type\" : \"ids:Permission
\",\n    \"@id\" : \"https://vcese12.inesctec.pt:8080/subcontracted/api/rules/827334fae-b137-4599-9c05-964ca97d25ab\",\n    \"ids:description\" : [ {\n      \"@value\" : \"\",\n
\"@type\" : \"http://www.w3.org/2001/XMLSchema#string\"\n    } ],\n    \"ids:target\" : {\n      \"@id\" : \"https://vcese12.inesctec.pt:8080/subcontracted/api/artifacts/69fd770a-3d00
-4167-a9ab-55f9981e9367\"\n    },\n    \"ids:title\" : [ {\n      \"@value\" : \"\",\n      \"@type\" : \"http://www.w3.org/2001/XMLSchema#string\"\n    } ],\n    \"ids:assignee\" : [
{\n      \"@id\" : \"https://vcese11.inesctec.pt:8080/stvgodigital/producer-01\"\n    } ],\n    \"ids:assigner\" : [ {\n      \"@id\" : \"https://vcese12.inesctec.pt:8080/stvgodigita
l/subcontracted-01\"\n    } ],\n    \"ids:constraint\" : [ {\n      \"@type\" : \"ids:Constraint\",\n      \"@id\" : \"https://w3id.org/idsa/autogen/constraint/41406eb9-4129-46dc-aeb7
-93715248fb1f\",\n      \"ids:operator\" : {\n        \"@id\" : \"https://w3id.org/idsa/code/SAME_AS\"\n      },\n      \"ids:leftOperand\" : {\n        \"@id\" : \"https://w3id.org/i
dsa/code/SYSTEM\"\n      },\n      \"ids:rightOperand\" : {\n        \"@value\" : \"https://vcese11.inesctec.pt:8080/stvgodigital/producer-01\",\n        \"@type\" : \"http://www.w3.o
rg/2001/XMLSchema#anyURI\"\n      }\n    } ],\n    \"ids:action\" : [ {\n      \"@id\" : \"https://w3id.org/idsa/code/USE\"\n    } ]\n  } ],\n  \"ids:provider\" : {\n    \"@id\" : \"h
ttps://vcese12.inesctec.pt:8080/stvgodigital/subcontracted-01\"\n  },\n  \"ids:contractEnd\" : {\n    \"@value\" : \"2023-05-30T13:59:59.000Z\",\n    \"@type\" : \"http://www.w3.org/2
001/XMLSchema#dateTimeStamp\"\n  },\n  \"ids:consumer\" : {\n    \"@id\" : \"https://vcese11.inesctec.pt:8080/stvgodigital/producer-01\"\n  },\n  \"ids:contractStart\" : {\n    \"@val
ue\" : \"2022-05-30T14:00:38.147Z\",\n    \"@type\" : \"http://www.w3.org/2001/XMLSchema#dateTimeStamp\"\n  },\n  \"ids:contractDate\" : {\n    \"@value\" : \"2022-05-30T14:00:38.147Z
\",\n    \"@type\" : \"http://www.w3.org/2001/XMLSchema#dateTimeStamp\"\n  }\n}",
  "_links": {
    "self": {
      "href": "https://vcese11.inesctec.pt:8080/producer/api/agreements/f2fec544-60ef-44b8-9da3-da881a444dfc"
    },
```


Figure 5.9: Agreement ID on the consumer side

5.4.3 Challenges

Given that the CH is the most mature component, testing it had no significant difficulties. Nevertheless, the following section shows some observations made while testing this component.

5.4.3.1 Certificates being validated

Since the virtual machine first used to test this component was having some difficulties, the CH was moved and implemented on another machine without changing any file, including the virtual machine certificates.

Testing the CH without changing the certificates should have thrown an error of **Hostname not verified**.

This lack of validation was fixed shortly after implementing the Clearing House for the first time.

5.4.3.2 Contract negotiation logs error

When the CH was first tested, the IDS Connectors had been running beforehand, which meant the contract negotiation was already done.

Eventually, the connectors were stopped and the volumes removed, forcing them to do a contract negotiation exchange. It was then that the error on this exchange was caught. After searching the Clearing House and the Dataspace Connector GitHub pages, it was seen that there was already an issue regarding this problem.

With the next release, the developers fixed this error.

5.4.4 Experiments

To test the CH maturity were developed some experiments. They are described in detail in the following sections.

5.4.4.1 Experiment 1: Get all log entries from a contract agreements

This basic experiment retrieves all log entries stored under a given contract agreement in the Clearing House.

For this experiment, the Recipient ID used on the query endpoint was `https://vcese19.inesctec.pt:8443/messages/query/b6cc5d69-44ed-4c22-a524-f1904`

```
[
  {
    "@context": {
      "ids": "https://w3id.org/idsa/core/",
      "idsc": "https://w3id.org/idsa/code/"
    },
    "@type": "ids:LogMessage",
    "@id": "https://w3id.org/idsa/autogen/logMessage/1d688f26-1183-40f0-8b60-d14e16bfd6eb",
    "ids:modelVersion": "4.2.7",
    "ids:issued": "2022-05-30T14:00:39.009+00:00",
    "ids:issuerConnector": "https://vcese11.inesctec.pt:8080/stvgodigital/producer-01",
    "ids:senderAgent": "https://vcese11.inesctec.pt:8080/stvgodigital/producer-01",
    "payload": "{\n  \"@context\" : {\n    \"ids\" : \"https://w3id.org/idsa/core/\",\n    \"idsc\" : \"https://w3id.org/idsa/code/\"\n  }
    "payload_type": "application/ld+json"
  }
]
```


Figure 5.10: Log Entries

Figure 5.10 shows that the contract agreement has only a *Contract Agreement Log Message*. From that figure, it is possible to see that the field *payload* contains a JSON structure. Therefore, listing 5.2 displays that payload in JSON format to facilitate reading. From that listing, it is possible to see much relevant information, such as a rule associated with that contract agreement, the artefact to exchange, the consumer, the provider, the contract date, contract start and contract end, and the constraints and action.

```
1  {
2      "@context": {
3          "ids": "https://w3id.org/idsa/core/",
4          "idsc": "https://w3id.org/idsa/code/"
5      },
6      "@type": "ids:ContractAgreement",
7      "@id": "https://vcese11.inesctec.
           pt:8080/producer/api/agreements/b6cc5d69-44ed-4c22-a524-f19041d4af89",
8      "ids:permission": [
9          {
10             "@type": "ids:Permission",
11             "@id": "https://vcese11.inesctec.
                   pt:8080/producer/api/rules/2a0091a1-6e07-478d-bbbb-14b5c2dba609",
12             "ids:description": [
13                 {
14                     "@value": "",
15                     "@type": "http://www.w3.org/2001/XMLSchema#string"
16                 }
```

```
            ],
            "ids:target": {
                "@id": "https://vcese11.inesctec.
                    pt:8080/producer/api/artifacts/76460936-0ef0-4e55-880a-33b3304887b3
                    "
            },
            "ids:title": [
                {
                    "@value": "",
                    "@type": "http://www.w3.org/2001/XMLSchema#string"
                }
            ],
            "ids:assignee": [
                {
                    "@id": "https://vcese12.inesctec.
                        pt:8080/stvgodigital/subcontracted-01"
                }
            ],
            "ids:assigner": [
                {
                    "@id": "https://vcese11.inesctec.
                        pt:8080/stvgodigital/producer-01"
                }
            ],
            "ids:constraint": [
                {
                    "@type": "ids:Constraint",
                    "@id": "https://w3id.
                        org/idsa/autogen/constraint/64d81310-7334-4d2b-9f8e-691fdfa0688f
                        ",
                    "ids:operator": {
                        "@id": "https://w3id.org/idsa/code/SAME_AS"
                    },
                    "ids:leftOperand": {
                        "@id": "https://w3id.org/idsa/code/SYSTEM"
                    },
                    "ids:rightOperand": {
                        "@value": "https://vcese12.inesctec.
                            pt:8080/stvgodigital/subcontracted-01",
                        "@type": "http://www.w3.org/2001/XMLSchema#anyURI"
                    }
                }
            ],
            "ids:action": [
                {
                    "@id": "https://w3id.org/idsa/code/USE"
                }
            ]
        }
```

```
    ],
    "ids:provider": {
        "@id": "https://vcese11.inesctec.pt:8080/stvgodigital/producer-01"
    },
    "ids:contractEnd": {
        "@value": "2023-05-30T14:00:14.000Z",
        "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
    },
    "ids:consumer": {
        "@id": "https://vcese12.inesctec.pt:8080/stvgodigital/subcontracted-01"
    },
    "ids:contractStart": {
        "@value": "2022-05-30T14:00:27.213Z",
        "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
    },
    "ids:contractDate": {
        "@value": "2022-05-30T14:00:27.212Z",
        "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
    }
}
```

Listing 5.2: Contract Agreement Payload

5.4.4.2 Experiment 2: Get all log entries from a contract agreements

Given that the previous experiment only had a log entry, another experiment was made with a contract agreement that has more exchanged information.

For this experiment, the Recipient ID (Contract Agreement ID) was `https://vcese19.inesctec.pt:8443/messages/query/2665d-474c-4162-fc4606a538`

Figure 5.11 shows an excerpt of the response from the previous request. Just as the last experiment, the relevant information is found on the *payload* field.

Listing 5.3 shows the *Contract Agreement Message* from the previous payload in JSON format to be more legible. There is depicted the Agreement ID and the provider ID, *subcontracted-01*, among other information. Such as the rule applied to the artefact (with the ID *69fd770a-3d00-4167-a9ab-556*) meaning that only a consumer with the ID

https://vcese11.inesctec.pt:8080/stvgodigital/producer-01 can use said artefact. The listing also shows that the contract ends on 2023-05-30 and starts on 2022-05-30.

```
{
    "@context": {
        "ids": "https://w3id.org/idsa/core/",
        "idsc": "https://w3id.org/idsa/code/"
```

Figure 5.11: Log Entries

```
    },
    "@type": "ids:ContractAgreement",
    "@id": "https://vcese12.inesctec.
        pt:8080/subcontracted/api/agreements/266c365d-474c-4162-bbcf-79fc4606a538",
    "ids:provider": {
        "@id": "https://vcese12.inesctec.pt:8080/stvgodigital/subcontracted-01"
    },
    "ids:permission": [
        {
            "@type": "ids:Permission",
            "@id": "https://vcese12.inesctec.
                pt:8080/subcontracted/api/rules/82734fae-b137-4599-9c05-964ca97d25ab
                ",
            "ids:description": [
                {
                    "@value": "",
                    "@type": "http://www.w3.org/2001/XMLSchema#string"
                }
            ],
            "ids:target": {
                "@id": "https://vcese12.inesctec.
                    pt:8080/subcontracted/api/artifacts/69fd770a-3d00-4167-a9ab-556
                    "
            },
            "ids:title": [
                {
                    "@value": "",
                    "@type": "http://www.w3.org/2001/XMLSchema#string"
```

```
                }
            ],
            "ids:assignee": [
                {
                    "@id": "https://vcese11.inesctec.
                        pt:8080/stvgodigital/producer-01"
                }
            ],
            "ids:assigner": [
                {
                    "@id": "https://vcese12.inesctec.
                        pt:8080/stvgodigital/subcontracted-01"
                }
            ],
            "ids:constraint": [
                {
                    "@type": "ids:Constraint",
                    "@id": "https://w3id.
                        org/idsa/autogen/constraint/41406eb9-4129-46dc-aeb7-93715248fb1f
                        ",
                    "ids:operator": {
                        "@id": "https://w3id.org/idsa/code/SAME_AS"
                    },
                    "ids:leftOperand": {
                        "@id": "https://w3id.org/idsa/code/SYSTEM"
                    },
                    "ids:rightOperand": {
                        "@value": "https://vcese11.inesctec.
                            pt:8080/stvgodigital/producer-01",
                        "@type": "http://www.w3.org/2001/XMLSchema#anyURI"
                    }
                }
            ],
            "ids:action": [
                {
                    "@id": "https://w3id.org/idsa/code/USE"
                }
            ]
        }
    ],
    "ids:contractEnd": {
        "@value": "2023-05-30T13:59:59.000Z",
        "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
    },
    "ids:consumer": {
        "@id": "https://vcese11.inesctec.pt:8080/stvgodigital/producer-01"
    },
    "ids:contractStart": {
        "@value": "2022-05-30T14:00:38.147Z",
```

```
        "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
    },
    "ids:contractDate": {
        "@value": "2022-05-30T14:00:38.147Z",
        "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
    }
}
```

Listing 5.3: Contract Agreement from agreement

Listing 5.4 shows the *Artefact Request Message* from the figure 5.11 payload in JSON format. It is possible to see, among other things, the artefact to which the message refers (*b340d840-471a-4f0f-a017-daede75e203a*), the ID of the issuer Connector (*https://vcese11.inesctec.pt:8080/stvgodigital/producer-01*), the endpoint of the recipient Connector (*https://vcese12.inesctec.pt:8080/subcontracted/api/ids/data*) and the contract agreement ID (*266c365d-474c-4162-bbcf-79fc4606a538*).

```
{
    "@context": {
        "ids": "https://w3id.org/idsa/core/",
        "idsc": "https://w3id.org/idsa/code/"
    },
    "@type": "ids:ArtifactRequestMessage",
    "@id": "https://w3id.
        org/idsa/autogen/artifactRequestMessage/b340d840-471a-4f0f-a017-daede75e203a
        ",
    "ids:requestedArtifact": {
        "@id": "https://vcese12.inesctec.
            pt:8080/subcontracted/api/artifacts/69fd770a-3d00-4167-a9ab-55f9981e9367
            "
    },
    "ids:modelVersion": "4.2.7",
    "ids:issued": {
        "@value": "2022-05-30T14:00:43.773Z",
        "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
    },
    "ids:issuerConnector": {
        "@id": "https://vcese11.inesctec.pt:8080/stvgodigital/producer-01"
    },
    "ids:recipientConnector": [
        {
            "@id": "https://vcese12.inesctec.pt:8080/subcontracted/api/ids/data"
        }
    ],
    "ids:senderAgent": {
        "@id": "https://vcese11.inesctec.pt:8080/stvgodigital/producer-01"
    },
    "ids:securityToken": {
        "@type": "ids:DynamicAttributeToken",
```

```
29          "@id": "https://w3id.
                org/idsa/autogen/dynamicAttributeToken/5b8bbfae-eecd-4aad-b2df-06f1e045bbb1
                ",
30          "ids:tokenValue": "
                eyJ0eXAiOiJhdCtqd3QiLCJraWQiOiJUQ1VGZUNOYXphbEtIZzlLenJ6TElBelJXV.
                E1ERFdTYTdMY005WndITXlvIiwiYWxnIjoiUlMyNTYifQ.eyJhdWQiOiJpZHNjOklE.
                U19DT05ORUNUT1JTX0FMTCIsImlzcyI6Imh0dHBzOi8vZGFwcy5haXNlYy5mcmF1bmhv.
                ZmVyLmRlIiwic3ViIjoiOEU6MzQ6MDg6Njk6MUM6QjY6Q0I6QUQ6NkI6NTU6OEI6OU.
                U6MTg6REU6RjY6MTg6QTM6RkI6MEI6NTc6a2V5aWQ6Q0I6OEM6Qzc6QjY6ODU6N.
                zk6QTg6MjM6QTY6Q0I6MTU6QUI6MTc6NTA6MkY6RTY6NjU6NDM6NUQ6RTgiLCJuYmY.
                iOjE2NTM5MTkyMTQsImlhdCI6MTY1MzkxOTIxNCwianRpIjoiTVRNM016QTBPVFUzT.
                npZeU1UZzNPVEV5TmpZPSIsImV4cCI6MTY1MzkyMjgxNCwiY2xpZW50X2lkIjoiOEU6Mz.
                Q6MDg6Njk6MUM6QjY6Q0I6QUQ6NkI6NTU6OEI6OUU6MTg6REU6RjY6MTg6QTM6RkI6.
                MEI6NTc6a2V5aWQ6Q0I6OEM6Qzc6QjY6ODU6Nzk6QTg6MjM6QTY6Q0I6MTU6QUI6.
                MTc6NTA6MkY6RTY6NjU6NDM6NUQ6RTgiLCJzZWN1cml0eVByb2ZpbGUiOiJpZHNjO.
                kJBU0VfU0VDVVJJVFlfUFJPRklMRSIsInJlZmVycmluZ0Nvbm5lY3RvciI6Imh0dH.
                A6Ly92Y2VzZTExLmluZXNjdGVjLnB0LmRlbW8iLCJAdHlwZSI6ImlkczpEYXRQYX.
                lsb2FkIiwiQGNvbnRleHQiOiJodHRwczovL3czaWQub3JnL2lkc2EvY29udGV4dHM.
                vY29udGV4dC5qc29ubGQiLCJ0cmFuc3BvcnRDZXJ0c1NoYTI1NiI6IjE5Mzk3NWVjZj.
                ZhM2M3NGUyMDE3OWU0OWI3NTliOTcxNjdjYmMzMjViODVlNTEwNDQ3M2RhZmJjYzU3YT.
                czMjciLCJzY29wZXMiOlsiaWRzYzpJRFNfQ09OTkVDVE9SX0FUVFJJQlVURVNfQUxMIl19.
                trEoP2nJBIlQV1hqy35_qfurHR1PqrEkAmnUzXc4KQHa801DQDu3lB70EhhI8LghPOl1U.
                A3kq_4uC33L3jworatSsvdynr5GvMFEaO_booECFzmgjOJrzKurHrzLH5DnVzQPaVifZ93.
                BQpz0sqAIKnGlqhveQLxkr5GWDM6hxGr9rfufrrSBMMrFawaPBF6IO88yTWPLpajH_zZs.
                OskLWgYAoxf4tyiUxnGluigslnJXy3clKldncT2KJv6w0F_3gGBtiLpbdpE_Vg0Pxt.
                YQGofHwTOrANwtWkzE25BbNDUU01WGKDhbVg1rtc5q-gfs0caktqSB_LT50Z5TfHIg",
31          "ids:tokenFormat": {
32              "@id": "https://w3id.org/idsa/code/JWT"
33          }
34      },
35      "ids:transferContract": {
36          "@id": "https://vcese12.inesctec.
                pt:8080/subcontracted/api/agreements/266c365d-474c-4162-bbcf-79fc4606a538
                "
37      }
38  }
```

Listing 5.4: Artefact Request Message from agreement

Listing 5.5 shows the payload that contains the *Artefact Response Message* from figure 5.11. It is possible to see, among other things, the artefact response message ID (*1fdec0ac-8397-48af-9e1f-c5b2c1569ac3*), artefact request message this message responds to (*b340d840-471a-4f0f-a017-daede75e203a*), the ID of the issuer Connector

(*https://vcese12.inesctec.pt:8080/stvgodigital/subcontracted-01*), the ID of the recipient Connector (*https://vcese11.inesctec.pt:8080/stvgodigital/producer-01*) and the contract agreement ID (*266c365d-474c-4162-bbcf-79fc4606a538*). This response is very similar to the artefact request message from the previous listing, the only difference being the field that shows the ID of the

message this message responds to.

```
1
2  {
3      "@context": {
4          "ids": "https://w3id.org/idsa/core/",
5          "idsc": "https://w3id.org/idsa/code/"
6      },
7      "@type": "ids:ArtifactResponseMessage",
8      "@id": "https://w3id.org/idsa/autogen/artifactResponseMessage/1fdec0ac-8397-48
           af-9e1f-c5b2c1569ac3",
9      "ids:modelVersion": "4.2.7",
10     "ids:issued": {
11         "@value": "2022-05-30T14:00:46.131Z",
12         "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
13     },
14     "ids:correlationMessage": {
15         "@id": "https://w3id.org/idsa/autogen/artifactRequestMessage/b340d840-471a
               -4f0f-a017-daede75e203a"
16     },
17     "ids:issuerConnector": {
18         "@id": "https://vcese12.inesctec.pt:8080/stvgodigital/subcontracted-01"
19     },
20     "ids:recipientConnector": [
21         {
22             "@id": "https://vcese11.inesctec.pt:8080/stvgodigital/producer-01"
23         }
24     ],
25     "ids:senderAgent": {
26         "@id": "https://vcese12.inesctec.pt:8080/stvgodigital/subcontracted-01"
27     },
28     "ids:securityToken": {
29         "@type": "ids:DynamicAttributeToken",
30         "@id": "https://w3id.org/idsa/autogen/dynamicAttributeToken/2e434742-e8d2
               -4678-b04b-9ceebbcba44b",
31         "ids:tokenValue": "
               eyJ0eXAiOiJhdCtqd3QiLCJraWQiOiJUQ1VGZUNOYXphbEtIZzlLenJ6TElBelJXVE1ER-
               FdTYTdMY005WndITXlvIiwiYWxnIjoiUlMyNTYifQeyJhdWQiOiJpZHNjOklEU.19
               DT05ORUNUT1JTX0FMTCIsImlzcyI6Imh0dHBzOi8vZGFwcy5haXNlYy5mcm-
               F1bmhvZmVyLmRlIiwic3ViIjoiMkY6NzY6MkQ6RUM6QUE6ODQ6RUY6NTc6.
               MkM6MTQ6QTI6QzU6QTU6MDc6OTQ6NTM6NTA6RjI6QTg6REQ6a2V5aWQ6Q0I6OEM6Qzc6Qj.
               Y6ODU6Nzk6QTg6MjM6QTY6Q0I6MTU6QUI6MTc6OTQ6NTM6NTA6RjI6QTg6REQ-6
               a2V5aWQ6Q0I6OEM6Qzc6QjY6ODU6Nzk6QTg6MjM6QTY6Q0I6MTU6QUI6MTc6NTA6MkY6R-
               TYNjU6NDM6NUQ6RTgiLCJzZWN1cml0eVByb2ZpbGUiOiJpZHNjOkJBU0VfU0VDVV.
               JJVFlfUFJPRklMRSIsInJlZmVycmluZ0Nvbm5lY3RvciI6Imh0dHA6Ly92Y2VzZTEyLmlu-
               ZXNjdGVjLnB0LmRlbW8iLCJAdHlwZSI6ImlkczpEYXRQYXlsb2FkIiwiQGNvbn.
               RleHQiOiJodHRwczovL3czaWQub3JnL2lkc2EvY29udGV4dHMvY29udGV4dC5qc29u.
               bGQiLCJ0cmFuc3BvcnRDZXJ0c1NoYTI1NiI6ImRmNGY5NmNhNWZiOTM1N2NlYmM.
```

```
            zODViYzVjMjU5NTA2MjhhMTA4ZDI4YjRmNTFhY2E5ZTJjMDdlMzA5ZDEzYmQiLCJz.
            Y29wZXMiOlsiaWRzYzpJRFNfQ09OTkVDVE9SX0FUVFJJQlVURVNfQUxMIl19.19
            DT05ORUNUT1JTX0FMTCIsImlzcyI6Imh0dHBzOi8vZGFwcy5haXNlYy5mcm-
            F1bmhvZmVyLmRlIiwic3ViIjoiMkY6NzY6MkQ6RUM6QUE6ODQ6RUY6NTc6.
            MkM6MTQ6QTI6QzU6QTU6MDc6OTQ6NTM6NTA6RjI6QTg6REQ6a2V5aWQ6Q0I6OEM6Qzc6Qj.
            Y6ODU6Nzk6QTg6MjM6QTY6Q0I6MTU6QUI6MTc6OTQ6NTM6NTA6RjI6QTg6REQ-6
            a2V5aWQ6Q0I6OEM6Qzc6QjY6ODU6Nzk6QTg6MjM6QTY6Q0I6MTU6QUI6MTc6NTA6MkY6R-
            TYNjU6NDM6NUQ6RTgiLCJzZWN1cml0eVByb2ZpbGUiOiJpZHNjOkJBU0VfU0VDVV",
        "ids:tokenFormat": {
            "@id": "https://w3id.org/idsa/code/JWT"
        }
    },
    "ids:transferContract": {
        "@id": "https://vcese12.inesctec.pt:8080/subcontracted/api/agreements/266
            c365d-474c-4162-bbcf-79fc4606a538"
    }
}
```

Listing 5.5: Artefact Response Message from agreement

This experiment shows three of the four messages logged to the Clearing House: contracted agreement messages and incoming and outgoing artefact messages. To log the fourth type of message (data usage) is necessary to specify a usage policy that obligates the logging, which this existing architecture does not do.

Experiments 1 and 2 show that not all agreements have so many messages between the participants. The only mandatory message type is the *Contract Agreement Message*.

5.4.4.3 Experiment 3: Existing environment communication

After running the Clearing House for some time, it was necessary to know if turning it off would impact the DataspaceApp4EDI plus connector behaviour.

Three different tests were implemented to guarantee that it does not. All these tests start with turning the Clearing House off.

1. Submit a document on the producer App4EDI interface and see if it appears on the subcontracted application interface. This base test confirmed that the operation App4EDI plus Connector is not dependent on the CH.

2. Turn the subcontracted DataspaceApp4EDI and connector off. Follow that action by submitting a document on the producer App4EDI interface to the subcontracted application. After around 2 minutes, turn the subcontract environment on and check if the sent document appears.

3. Turn the subcontracted environment off. Submit a document on the producer environment and then only start the subcontracted Connector. After around 3 minutes, start the subcontracted DataspaceApp4EDI and check if the document appears.

These tests are supposed to pass without problems and were run every time the existing environment or the Clearing House was updated.

5.4.4.4 Experiment 4: Retrieve a single log entry from a contract agreement

The Clearing House provides a process to retrieve a log entry stored under a given contract agreement[3]. Therefore, it only returns the specified log message instead of a list.

However, the Dataspace Connector documentation only provides an example that only takes one parameter to query the Clearing House.

Experiment 4 is developed, taking into account the documentation inconsistency. It tests the process of returning a single log entry from a contract agreement. The process uses the request method *POST /messages/query/{pid}/{id}* from the CH documentation.

This experiment uses the response obtained from experiment 2, depicted in figure 5.11 on page 57. The combination of values tried is shown on the list below:

- Contract ID + Contract Agreement ID:

  https://vcese19.inesctec.pt:8443/messages/query/266c365d-474c-4162-bbcf-79fc4606a538/266c365d-474c-4162-bbcf-79fc4606a538

- Contract + Artefact Request Message ID:

  https://vcese19.inesctec.pt:8443/messages/query/266c365d-474c-4162-bbcf-79fc4606a538/a385835f-dd31-4660-a1b7-65cb1b963733

  https://vcese19.inesctec.pt:8443/messages/query/266c365d-474c-4162-bbcf-79fc4606a538/d55b69e6-bca5-4942-aa84-b54f689ef494

- Contract ID + Artefact Request Message ID from Payload:

  https://vcese19.inesctec.pt:8443/messages/query/266c365d-474c-4162-bbcf-79fc4606a538/b340d840-471a-4f0f-a017-daede75e203a

- Contract ID + Artefact Response Message ID:

  https://vcese19.inesctec.pt:8443/messages/query/266c365d-474c-4162-bbcf-79fc4606a538/4cd99171-ce14-40cb-85e3-e7b2dde2d58b

  https://vcese19.inesctec.pt:8443/messages/query/266c365d-474c-4162-bbcf-79fc4606a538/ba5f6656-7527-40c8-9d92-c966b8df1e71

- Contract ID + Artefact Response Message ID from Payload:

  https://vcese19.inesctec.pt:8443/messages/query/266c365d-474c-4162-bbcf-79fc4606a538/1fdec0ac-8397-48af-9e1f-c5b2c1569ac3

All these tests returned an error. Despite not being a significant issue, it is noted that the connector still needs to develop this functionality.

[3] https://github.com/Fraunhofer-AISEC/ids-clearing-house-service/blob/master/doc/clearing-house-api.md

5.4.4.5 Experiment 5: Test querying the CH with a request body

It was tried to query the Clearing House with a request body. For that, it was used the example value. The response returned the log entry from the request ID.

This experiment concluded that the request body is not being validated by the Cleaning House.

5.4.5 Differences between the releases

Given the two-month difference between the time the Clearing House was first implemented on VM *vcese18.inesctec.pt* (January 25, 2022) and the, at the time last available release (v0.7.1 on November 01, 2021), it was decided to use the repository code instead of the official release. Then every time the repository was updated, the changes were implemented on the virtual machine.

Comparing that first implementation with the last available release (v0.7.7 on April 27, 2022), some problems were solved. The most notable differences were the certificate validation, the contract negotiation agreements logs stored without errors, and the IDSCP2 routes on the bind mount file *clearing-house-routes.xml* do not give errors.

Despite the main differences with these releases happening on the repository code, there were changes on docker-compose and bind mounts to accommodate them. Those changes introduced some mandatory Trusted Connector environment variables and some variables to configure the Keystore and Truststore passwords (*TC_CH_ISSUER_CONNECTOR* and *TC_CH_AGENT)* and *TC_KEYSTORE_PW* and *TC_TRUSTSTORE_PW*, respectively), updated the Trusted Connector tag version to 6.3.0 (replacing the 6.2.0 that stop working) and rebuilding the Clearing House processors and putting the file on the bind mount.

On May 24, 2022, version v0.8.1 of Clearing-house Service was released. This version brought various code organization changes and, technically, fixed the *camel-spring.xsd* URL and did a cleanup on the integration tests. Regarding the code organization, the main changes, the *clearing-house-core* repository became redundant since all the code went to a directory inside the *Clearing-house-app* directory of the *clearing-house-service* repository. Finally, the *clearing-house-api* was renamed to **Logging System** which is what is currently implemented. The logs that before were all on tc-core are now divided between the tc-core and logging-service

5.4.6 Adaption to other environments

When the Clearing House was stable, there was a move to *vcese19.inesctec.pt*. This allowed a better understanding of each file that needs swapping in the bind mounts when the virtual machine changes. The files changed are:

- Keystore on the bind mounts: the file, *keystore.p12*, is the certificate for the virtual machine that IDS assigned to INESC TEC;
- Keys file: Recreate the keys to sign the receipts for the container *logging-system* using the OpenSSL command given in section 5.4.1;

- The Clearing House processors should also be rebuilt, and replaced the bind mounts jar file with the new one.

5.5 Metadata Broker

The Metadata Broker's responsibility is to register and make a catalogue of IDS participants and components available.

In the *STVgoDIGITAL* environment, the Connectors need the Broker to know the partner ID. What is currently happening is that the application App4EDI has a configuration file that says the partners and their IDs, with which the Connector can communicate. Introducing a broker in this environment solves the need to configure all partnerships and IDs, a behaviour that is not sustainable in large or medium production environments. Essentially, using an MB allows the Connector to query said Broker to know the partner ID.

The Broker (metadata-Broker-open-core[4]) is still under development by the International Data Spaces Association. Its implementation was straightforward, considering the IDSA already has the repository ready to deploy with docker-compose. However, some challenges arose given the Broker development state and the lack of documentation (when the component was implemented).

The implementation has three containers: reverse proxy, fuseki and the broker itself. The Apache Fuseki is a SPARQL server responsible for persistence storage.

5.5.1 Implementation/Installation Guide

There were two different options to implement the Metadata Broker. The first and chosen was to use a docker-compose with the Fraunhofer container image, and the other was to build the image from the repository code and then use the created image. As the repository is constantly changing, both approaches have the downfall of always being necessary to update the images. However, when using the Fraunhofer image, it is imperative to pull the images regularly since they change the image but maintain the tag name.

The first step to implement the Broker was changing the docker volumes to include a bind mount instantiated on the same directory, */cert*, of the docker-compose. There were put three files, *isstbroker-keystore.jks*, *server.crt* and *server.key*, as described in section DAPS token was not being validated correctly. There are also specified the *Broker-core* container environment variables changes. Essentially, one environment variable value was updated, and one environment variable was added.

The final docker-compose is shown on Appendix B.

[4] https://github.com/International-Data-Spaces-Association/metadata-Broker-open-core

5.5.2 Testing procedures

The Metadata Broker does not have any user interface. All the communication with it are done via the clients Connectors. However, it is possible to see the Broker is running when going to `https://vcese19.inesctec.pt:443/`, where the Broker configuration file is displayed.

To test the Broker, the "producer" and the "subcontracted" Connector Swagger page was used. The steps are as follows:

1. **Create Broker entity:** *POST /api/Brokers*

   Request Body:

```
{
  "title": "Broker",
  "description": "Metadata Broker",
  "location": "https://vcese19.inesctec.pt:443/infrastructure"
}
```

Listing 5.6: Create Broker Request Body

2. **Register Connector at Broker**, to then register resources: *POST /api/ids/connector/update*

   Recipient URL: `https://vcese11.inesctec.pt:8080/producer/api/Brokers/<Broker-ID>`

3. **Register resource at Broker**: *POST /api/ids/resource/update*

   The Recipient URL is the Broker URL, the same as the previous step.

   The Resource ID comes from the Endpoint Offered Resource, *GET /api/offers*.

   Figure 5.12 shows an example request.

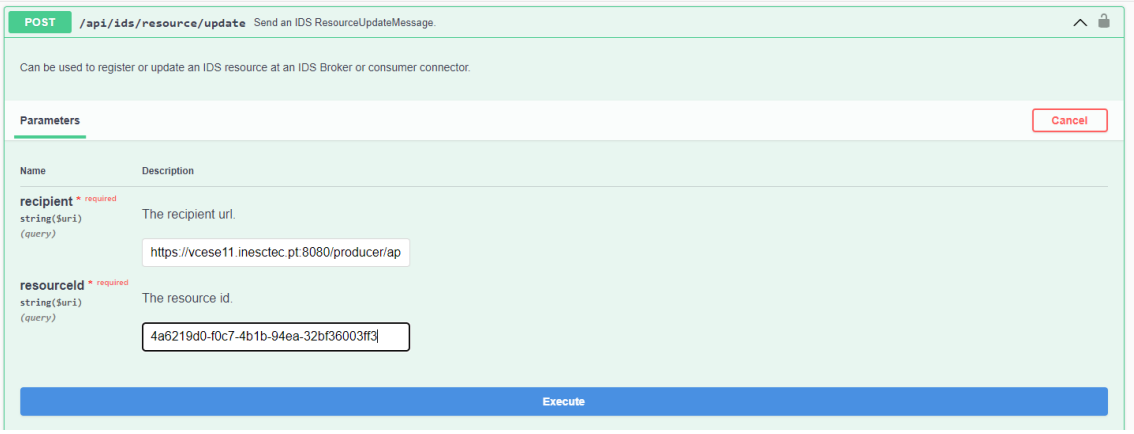


Figure 5.12: Update Resource Example

4. **To get all offers registered at a Broker:** *GET /api/brokers/id/offers*.
5. **To get all Brokers an offer has been registered to:** *GET /api/offers/id/Broker*
6. **Query Broker to search for keywords:** *POST /api/ids/search*
7. **Query Broker with SPARQL:** *POST /api/ids/query*
8. **Request more details about an element:** *POST /api/ids/description*

5.5.3 Challenges

Given that there was not much documentation regarding the Metadata Broker deployment, use and test, some challenges were encountered.

The sections below describe the challenges and solutions found.

5.5.3.1 DAPS token was not being validated correctly

This challenge arises when making step 2 from the Testing procedures section, which is the first time the Connector and the Broker communicate.

The solution found involved changing the certificates and adding an environment variable to connect with the DAPS correctly.

The first step was to exchange the *isstbroker-keystore.jks* file by the certificate the IDS assigned INESC TEC for virtual machine *vcese19.inesctec.pt* and to put it in the bind mount. This certificate was requested on a form IDS has for all types of certificates. When the certificate came, it was necessary to convert it from *p12* to *jks* file extension using OpenSSL. The certificate was then put on the bind mount, which was also altered for a better location.

The *server.crt* and *servcer.key* are the certificates created by INESC TEC to the virtual machine.

With the proper certificates on the bind mounts was then necessary to change some fields on the docker-compose, namely to add environment variable *DAPS_URL* and update environment variable *IDENTITY_JAVAKEYSTORE* (path to the previously created certificate). The following listing illustrates the new variables.

```
1 DAPS_URL=https://daps.aisec.fraunhofer.de/v2/token
2 IDENTITY_JAVAKEYSTORE=/etc/cert/isstbroker-keystore.jks
```

Listing 5.7: Broker docker-compose changes

5.5.3.2 Hostname being validated

Contrary to the Clearing House, the hostname on the certificates was being validated. Given that the *vcese18.inesctec.pt* machine was having some problems, this test was being made on the

vcese11.inesctec.pt machine. Because of that, an error **Hostname not verified** was thrown into the *broker-core* container logs.

Exchanging the certificates from challenge 5.5.3.1 with certificates from the *vcese11.inesctec.pt* virtual machine solved the issue.

Later, the Broker was changed to virtual machine *vcese19.inesctec.pt* and the certificates were changed again.

5.5.3.3 Broker does not run on test environment

The IDS Dataspace Connectors have two different deploy modes: a test and a productive mode. The difference is that when in test deployment, the Connector is trusting all certificates.

The Broker needs the Connector to be in productive deployment for it to work. As a localhost test environment was being used, the Connector was in test deployment mode. The changes needed to change modes are:

On Connector *config.json* file, the field *id* changes from "TEST_DEPLOYMENT" to "PRODUCTIVE_DEPLOYMENT", and the *Keystore id* changes to the Keystore certificate file. The applied changes on file *config.json* are demonstrated below, on listing 5.8.

```
"ids:ConnectorDeployMode" : {
    "@id" : "idsc:PRODUCTIVE_DEPLOYMENT"
},
"ids:keyStore" : {
    "@id" : "file:///tmp/conf/vcese11.inesctec.pt.p12"}
```

Listing 5.8: Deploy Mode

On the Connector docker-compose, the environment variable *configuration.keyStorePassword* value is replaced with the new file password.

5.5.3.4 Structure error when registering resources at Broker

When trying to register resources at the Broker, the message **The following mandatory field(s) of Constraint are not filled or invalid: ids:pipEndpoint.** was thrown. When checking the resource catalogue, the field was indeed empty, but the error persisted even after updating the resource.

After checking the Broker Issues page, one issue was already open about the same error, and it was necessary to wait a bit for the developers' answer.

The error was because the Broker message format was updated and the resources already created on the existing environment had the old format. However, the error message was misleading since the *pipEndpoint* was not the only field updated, nor is it mandatory.

To solve this problem, it was necessary to create new resources from scratch to see which resources format was updated (Catalogs, Contracts, Offered Resources, Rules, Representations and Artefacts). The updated resources were the artefact and the contract.

It was also compared the values from the resources that did not work with the newly created one, and from that, it was seen that the Rule from the existing resource was far more complex than the new one. After testing the simpler rule, it was seen that the Broker threw that error when a rule restricts the data usage for a specific provider or consumer.

After seeing that the new format solved the issue, it was necessary to go to the existing environment App4EDI and update the Contracts, Artefacts and Rules for them to create the revised version and then be able to test the Broker without needing to change them manually. Updating the data app is also crucial since, as this is a test environment, sometimes it is necessary to remove the docker volumes and restart the contract agreement and data exchange from scratch.

The experiments made to solve this issue are described in more detail in section 5.5.4, experiments Experiment 1: Create new resources, Experiment 2: Only update the Contract, Experiment 3: Only update artefact, Experiment 4: Update artefact and contract, and Experiment 5: Find the value that throws the error.

5.5.4 Experiments

To test the Broker maturity were developed some experiments. They are described in detail in the following sections.

5.5.4.1 Experiment 1: Create new resources

When first registering resources on the Broker, an error message was returned, as described in the section 5.5.3.4 named Structure error when registering resources at Broker.

The first step to solve the problem was to research which model should contain the *pipEndpoint* and fill it. As this did not solve the issue, the next step was to search for this error on the Metadata Broker GitHub page.

At the time, the issue was already registered. However, the only answer was to change the *pipEndpoint* format to include some fields inside it, as shown on listing 5.9. As this also did not fix the issue, several days were waited to see if there were other responses. The following answer said that *pipEndpoint* was not the only updated field, so it was then tried to create new resources on the Connector to see which changes were made. When creating new resources, it was noted that the Artefact and Contract models were the ones updated. So, experiments 2, 3 and 4 were designed.

After creating the new resources, the endpoint *POST /api/ids/resource/update* was rerun with success.

```
1 ...
2 "ids:pipEndpoint": {
3     "@type": "ids:PIP",
4     "ids:interfaceDescription": {
5         "@id": "https://ids.org/PIP/askIfPurpose"
6     },
```

```
7 ...
```

Listing 5.9: *pipEndpoint* changes from GitHub issue

5.5.4.2 Experiment 2: Only update the Contract

Given the problem explained in section Structure error when registering resources at Broker and Experiment 1: Create new resources, was created an experiment to test the resource update without needing to create a new resource.

The update contract endpoint, *PUT /api/contracts/{id}*, from the Provider connector swagger page was used to execute this The changes made were to add the *provider* field on Contract. This field contains the ID for the Contract provider.

This change was insufficient to make the offered resource work with the Broker.

5.5.4.3 Experiment 3: Only update artefact

Given the problem described in section Structure error when registering resources at Broker and on the previous experiment, another experiment was created to test only the resource update without needing to create a new resource.

Experiment 3 used the artefact update endpoint, *PUT /api/artifacts/{id}*. The changes made were to add the *apiKey* field on the artefact.

This change was also not enough to make the offered resource work with the Broker.

5.5.4.4 Experiment 4: Update artefact and contract

This experiment was developed based on the challenge Structure error when registering resources at Broker and the last two experiments. Since neither updating an artefact nor a contract solved the issue described, an experiment was created that updated both structures.

The experiment consisted of adding the *apiKey* field on the artefact and the *provider* field on the contract.

These changes were also not enough to make the offered resource work with the Broker.

After experiments 2, 3 and 4, it is possible to see that updating a resource does not work. It is necessary to create a new one.

5.5.4.5 Experiment 5: Find the value that throws the error

Based on the previous experiments, a new one was developed to see if a new resource created with the existing resource offer values is able to be registered on the Broker.

Every data model structure was created with the offer values that previous throw the error. The *GET* and *POST* endpoints from the Catalogue, Resource offer, Representation, Artefact, Contract and Rule were used to accomplish this, as well as the endpoints responsible for linking the resources created.

Given that after this process, the new resource was still throwing the same error when registering it at the Broker, it was studied the different values between the resource that could be registered and the one that could not. After identifying the differences, one at a time, each value on the resource that did not work was replaced with the one that did, and the endpoint *POST /api/ids/resource/update* was rerun.

With this process, the Rule was identified as the problem. After an experiment with the Rule values, it was noted that the more complex rules, e.g. Connector-restricted, do not work with the Broker. Listing 5.10 shows a rule that *Provides access* to all Connectors, which can be registered on a Broker. If a rule restricts access to specific Connectors, it is not accepted.

This experiment proves that the Connector-Broker interaction still needs some work since the rules are essential in guaranteeing data sovereignty.

```
{
    "@context": {
        "ids": "https: //w3id.org/idsa/core/",
        "idsc": "https://w3id.org/idsa/code/"
    },
    "@type": "ids:Permission",
    "@id": "https://w3id.
        org/idsa/autogen/permission/10114e7b-722c-4dc6-89a3-2bd5352637f3",
    "ids:description": [
        {
            "@value": "string",
            "@type": "http://www.w3.org/2001/XMLSchema#string"
        }
    ],
    "ids:title": [
        {
            "@value": "PROVIDE_ACCESS",
            "@type": "http://www.w3.org/2001/XMLSchema#string"
        }
    ],
    "ids:action": [
        {
            "@id": "https://w3id.org/idsa/code/USE"
        }
    ]
}
```

Listing 5.10: *Provide access* Rule

5.5.4.6 Experiment 6: Get everything inside a Metadata Broker

There are two options for querying a Broker: endpoint query, which uses SPARQL language, and endpoint search, which uses keywords. Given that SPARQL language allows for a more specific

search, the *POST api/ids/query* was the endpoint used on experiment 6.

To get everything that is inside a Broker, the query is:

```
SELECT ?subject ?predicate ?object
WHERE {
    ?subject ?predicate ?object
}
```

Listing 5.11: Get everything that is inside a broke

The query returns a list with everything that is inside the Broker. Figure 5.13 shows an excerpt of the query response, where all the information inside the Broker is displayed, from the Connector self-description to the available resource catalogue As this is hard to read and understand, the specific query from experiment Experiment 8: Find Connector by resource was used.

Figure 5.13: List everything that is inside the broke

5.5.4.7 Experiment 7: Search by keywords

As said in the previous experiment, there are two options for querying a Broker: keyword and SPARQL commands. This experiment shows the keywords search, *POST /api/ids/search*, and results.

Listing 5.12 shows the search result with the keyword *eBIZ*. Listing 5.13 shows the result of searching with the keyword Connector.

With this experiment, it is possible to see that the keywords endpoint does not allow for a specific query, nor does it allow us to specify what we want it to return us. It also shows us that the result is complicated to read and interpret.

```
1 ?resultUri  ?res  ?type ?accessUrl  ?internalUri
2 <https://vcese11.inesctec.pt:8080/producer/api/artifacts/dd7b5aa8-348a-4f28-9b8b-
    db3e0c9f1af9>  "EBIZ-subcontracted-producer-feedback-control"    <https://
    vcese11.inesctec.pt:8080/producer/api/ids/data>  <https://vcese19.inesctec.pt/
    connectors/-375040176/-1271902210/749153681/-1899575848/-2001520518>
3 <https://vcese11.inesctec.pt:8080/producer/api/offers/26759479-047f-44fb-bb03-
    a6b5faf6703c> "EBIZ-producer-subcontracted-channel 2"@en  <https://w3id.org/
    idsa/core/Resource> <https://vcese11.inesctec.pt:8080/producer/api/ids/data>  <
    https://vcese19.inesctec.pt/connectors/-375040176/-1271902210/749153681>
4 <https://vcese11.inesctec.pt:8080/producer/api/offers/26759479-047f-44fb-bb03-
    a6b5faf6703c> "EBIZ-producer-subcontracted-channel resource holding data sent
    from 'producer' towards 'subcontracted'"@en <https://w3id.org/idsa/core/
    Resource> <https://vcese11.inesctec.pt:8080/producer/api/ids/data>  <https://
    vcese19.inesctec.pt/connectors/-375040176/-1271902210/749153681>
5 <https://vcese11.inesctec.pt:8080/producer/api/offers/26759479-047f-44fb-bb03-
    a6b5faf6703c> "EBIZ"@en <https://w3id.org/idsa/core/Resource> <https://vcese11.
    inesctec.pt:8080/producer/api/ids/data>  <https://vcese19.inesctec.pt/
    connectors/-375040176/-1271902210/749153681>
```

Listing 5.12: Results from Search Broker by *eBiz* Keyword

```
1 ?resultUri  ?res  ?type ?accessUrl  ?internalUri
2 <https://vcese12.inesctec.pt:8080/stvgodigital/subcontracted-01>  "Dataspace
    Connector" <https://w3id.org/idsa/core/BaseConnector>  <https://vcese12.
    inesctec.pt:8080/subcontracted/api/ids/data> <https://vcese19.inesctec.pt/
    connectors/-394898234>
3 <https://vcese12.inesctec.pt:8080/stvgodigital/subcontracted-01>  "IDS Connector
    reference implementation"  <https://w3id.org/idsa/core/BaseConnector>  <https
    ://vcese12.inesctec.pt:8080/subcontracted/api/ids/data> <https://vcese19.
    inesctec.pt/connectors/-394898234>
4 <https://vcese11.inesctec.pt:8080/stvgodigital/producer-01> "Dataspace Connector" <
    https://w3id.org/idsa/core/BaseConnector>  <https://vcese11.inesctec.pt:8080/
    producer/api/ids/data>  <https://vcese19.inesctec.pt/connectors/-375040176>
5 <https://vcese11.inesctec.pt:8080/stvgodigital/producer-01> "IDS Connector
    reference implementation"  <https://w3id.org/idsa/core/BaseConnector>  <https
    ://vcese11.inesctec.pt:8080/producer/api/ids/data>  <https://vcese19.inesctec.
    pt/connectors/-375040176>
```

Listing 5.13: Results from Search Broker by *Connector* Keyword

5.5.4.8 Experiment 8: Find Connector by resource

This experiment simulates the working process of querying a Broker.

First, create resources on the producer connector and register them on the Metadata Broker. When wanting to connect a consumer connector to a remote resource, first consult the Broker to find out which connector provides the resource in question. Then connect to the remote Connector.

For the same reason as Experiment 6: Get everything inside a Metadata Broker, these queries were executed on endpoint *POST api/ids/query*. Various queries were tested as there can be various ways to obtain the desired output from a Broker. Furthermore, all these queries are constructed to return a subject (which will be used later on for obtaining more information about the resource), predicate (equivalent to the name on JSON) and object (the value associated with the predicate name).

- **Get everything for the Resource(s) under a Connector:**

  On subcontracted Swagger page run:

```
1  PREFIX ids: <https://w3id.org/idsa/core/>
2  SELECT ?subject ?predicate ?object
3  WHERE {
4        ?subject a ids:Resource .
5        ?subject ?predicate ?object
6    }
```

Listing 5.14: Get everything for the Resource(s) under a Connector

Figure 5.14 on page 74 shows the query response. There, it is possible to see the offered resources with their information, such as the title, language and keywords.

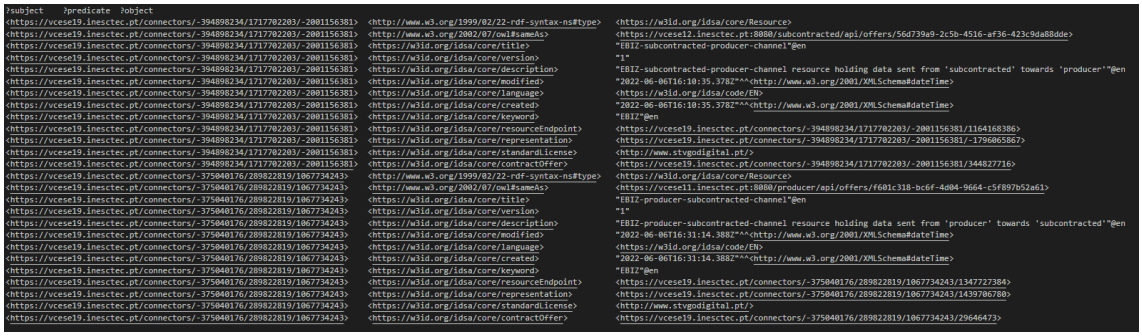


Figure 5.14: List Resource information that is inside the broke

- **Get the available resource's title:**

  This query returns all resource titles available. On subcontracted Swagger page run:

```
PREFIX ids: <https://w3id.org/idsa/core/>
SELECT ?subject ?predicate ?object
WHERE {
      ?subject a ids:Resource.
      ?subject ?predicate ?object
      FILTER (regex(str(?predicate), 'title','i'))
}
```

Listing 5.15: Get the title of the available resource

Figure 5.15 shows the query response, consisting of the offered resources titles. This query facilitates the Broker usage since it only returns the resource's titles instead of all the resource information, which can become hard to read.

```
?subject    ?predicate  ?object
<https://vcese19.inesctec.pt/connectors/-394898234/1717702203/-2001156381>  <https://w3id.org/idsa/core/title>  "EBIZ-subcontracted-producer-channel"@en
<https://vcese19.inesctec.pt/connectors/-375040176/289822819/1067734243>    <https://w3id.org/idsa/core/title>  "EBIZ-producer-subcontracted-channel"@en
```

Figure 5.15: List of Resources title

- **Which resources exist with titles equal to:**

  This query will be one of the more used options. With this, it is possible to search the Broker by resource title. On subcontracted Swagger page run:

```
PREFIX ids: <https://w3id.org/idsa/core/>
SELECT ?subject ?predicate ?object
WHERE {
    ?subject a ids:Resource.
    ?subject ?predicate ?object
    FILTER (regex(str(?predicate), 'title','i'))
    FILTER (regex(str(?object ), 'eBiz','i'))
  }
```

Listing 5.16: Get resources with title equal to

Figure 5.16 shows the query response, which consists of the offered resources with the word *eBiz* on the title.

```
?subject    ?predicate  ?object
<https://vcese19.inesctec.pt/connectors/-394898234/1717702203/-2001156381>  <https://w3id.org/idsa/core/title>  "EBIZ-subcontracted-producer-channel"@en
<https://vcese19.inesctec.pt/connectors/-375040176/289822819/1067734243>    <https://w3id.org/idsa/core/title>  "EBIZ-producer-subcontracted-channel"@en
```

Figure 5.16: List of Resources that have the word "eBiz" in the title

- **Get the available artefacts by filename:**

  This query searches the Broker by artefact instead of resources. On subcontracted Swagger page run:

```
PREFIX ids: <https://w3id.org/idsa/core/>
SELECT ?predicate ?object
WHERE {
    ?subject a ids:Artifact.
    ?subject ?predicate ?object
    FILTER (regex(str(?predicate), 'fileName','i'))
    FILTER (regex(str(?object ), 'EBIZ','i'))
  }
```

Listing 5.17: Get the available artefacts by filename

Figure 5.17 shows the query response, which consists of the artefacts titles. This query only returns the predicate and object, making it easier to read. However, if wanting to know more information about any value from the response, the subject is necessary.

```
?predicate  ?object
<https://w3id.org/idsa/core/fileName>   "EBIZ-subcontracted-producer-feedback-control"
<https://w3id.org/idsa/core/fileName>   "EBIZ-subcontracted-producer-outbound-control"
<https://w3id.org/idsa/core/fileName>   "EBIZ-subcontracted-producer-feedback-control"
```

Figure 5.17: List of available Artefacts in which the filename has the word "EBIZ"

- **Get available Connectors:**

  The query from listing 5.18 returns all Connectors registered on a Metadata Broker. On subcontracted Swagger page run:

```
PREFIX ids: <https://w3id.org/idsa/core/>
SELECT ?subject ?predicate ?object
WHERE {
      ?subject a ids:BaseConnector.
      ?subject ?predicate ?object
      FILTER (regex(str(?predicate), 'owl#sameAs','i'))
}
```

Listing 5.18: Get available connectors

Figure 5.18 shows the query response, consisting of all connectors' names registered on the Broker.

Figure 5.18: List Connectors registered on the broker

The Dataspace Connector currently does not support IDS-REST. Thereby, to see more information about the resource (e.g. Connector ID that provides the resource wanted), it is necessary to use the description endpoint, *POST /api/ids/description*, with the reference from the resource (subject returned from the previous queries). This endpoint does not yet support the Broker ID, so it is essential to use the Broker URL directly. An example of this request is:

Recipient: `https://vcese19.inesctec.pt:443/infrastructure`

ElementId: `https://vcese19.inesctec.pt/Connectors/-394898234/1717702203/-2001156381`

The following listing, 5.19, shows the response from the previous request. It presents the details from the resource with name equal to *"EBIZ-producer-subcontracted-channel"* taken from listing Get resources with title equal to with the result shown on figure List of Resources that have the word "eBiz" in the title.

Only after this last step, it is possible to see the provider ID on the response, on *ids:ContractOffer*.

```
{
  "@graph": [
    {
      "@id": "https://vcese19.inesctec.pt/connectors
          /-375040176/289822819/1067734243",
      "@type": "ids:Resource",
      "sameAs": "https://vcese11.inesctec.pt:8080/producer/api/offers/f601c318-bc6f
          -4d04-9664-c5f897b52a61",
      "contractOffer": "https://vcese19.inesctec.pt/connectors
          /-375040176/289822819/1067734243/29646473",
      "created": "2022-06-06T16:31:14.388Z",
      "description": {
        "@language": "en",
        "@value": "EBIZ-producer-subcontracted-channel resource holding data sent
            from 'producer' towards 'subcontracted'"
      },
      "keyword": {
        "@language": "en",
        "@value": "EBIZ"
      },
      "language": "https://w3id.org/idsa/code/EN",
      "modified": "2022-06-06T16:31:14.388Z",
      "representation": "https://vcese19.inesctec.pt/connectors
          /-375040176/289822819/1067734243/1439706780",
```

```
      "resourceEndpoint": "https://vcese19.inesctec.pt/connectors
          /-375040176/289822819/1067734243/1347727384",
      "standardLicense": "http://www.stvgodigital.pt/",
      "title": {
        "@language": "en",
        "@value": "EBIZ-producer-subcontracted-channel"
      },
      "version": "1"
    },
    {
      "@id": "https://vcese19.inesctec.pt/connectors
          /-375040176/289822819/1067734243/1347727384",
      "@type": "ids:ConnectorEndpoint",
      "sameAs": "https://w3id.org/idsa/autogen/connectorEndpoint/9b02ac15-c25d-4a8e
          -ad60-3bce36f575a1",
      "accessURL": "https://vcese19.inesctec.pt/connectors
          /-375040176/289822819/1067734243"
    },
    {
      "@id": "https://vcese19.inesctec.pt/connectors
          /-375040176/289822819/1067734243/1439706780",
      "@type": "ids:Representation",
      "sameAs": "https://vcese11.inesctec.pt:8080/producer/api/representations/
          cae621a2-9bd5-4c65-a5a5-a3bc05c0a088",
      "created": "2022-06-06T16:33:37.833Z",
      "instance": "https://vcese19.inesctec.pt/connectors
          /-375040176/289822819/1067734243/1439706780/517273664",
      "language": "https://w3id.org/idsa/code/EN",
      "mediaType": "https://w3id.org/idsa/autogen/iANAMediaType/05bd8418-be25-4e05
          -9feb-90bdd1f305f1",
      "modified": "2022-06-06T16:33:37.833Z",
      "representationStandard": "http://trader1.pt/aisov/data-vocabulary.html"
    },
    {
      "@id": "https://vcese19.inesctec.pt/connectors
          /-375040176/289822819/1067734243/1439706780/517273664",
      "@type": "ids:Artifact",
      "sameAs": "https://vcese11.inesctec.pt:8080/producer/api/artifacts/9d986d8d-
          cf19-4cab-85af-49d6c842e5b7",
      "ids:byteSize": 207,
      "checkSum": "2703831839",
      "creationDate": "2022-06-06T16:36:45.477Z",
      "fileName": "EBIZ-subcontracted-producer-feedback-control"
    },
    {
      "@id": "https://vcese19.inesctec.pt/connectors
          /-375040176/289822819/1067734243/29646473",
      "@type": "ids:ContractOffer",
```

```
      "sameAs": "https://vcese11.inesctec.pt:8080/producer/api/contracts/a65f4ebc
          -27d5-414b-ab90-fb602e73aba3",
      "consumer": "https://stvgodigital.pt/ids/entity/id/subcontracted",
      "contractDate": "2022-06-06T16:52:23.281Z",
      "contractEnd": "2023-06-06T16:11:17Z",
      "contractStart": "2022-06-06T16:11:17Z",
      "permission": "https://vcese19.inesctec.pt/connectors
          /-375040176/289822819/1067734243/29646473/771981982",
      "provider": "https://stvgodigital.pt/ids/entity/id/producer"
    },
    {
      "@id": "https://vcese19.inesctec.pt/connectors
          /-375040176/289822819/1067734243/29646473/771981982",
      "@type": "ids:Permission",
      "sameAs": "https://vcese11.inesctec.pt:8080/producer/api/rules/48683ec4-704b
          -4f31-bb36-d1b1fdf426a4",
      "action": "https://w3id.org/idsa/code/USE",
      "description": "provide-access",
      "title": "Allow Data Usage"
    },
    {
      "@id": "https://w3id.org/idsa/autogen/iANAMediaType/05bd8418-be25-4e05-9feb
          -90bdd1f305f1",
      "@type": "ids:IANAMediaType",
      "filenameExtension": "application/json;application/xml;application/octet-
          stream;text/plain;text/xml"
    },
    {
      "@id": "https://w3id.org/idsa/code/EN",
      "@type": "ids:Language"
    },
    {
      "@id": "https://w3id.org/idsa/code/USE",
      "@type": "ids:Action"
    }
  ],
  "@context": {
    "sameAs": {
      "@id": "http://www.w3.org/2002/07/owl#sameAs",
      "@type": "@id"
    },
    "representationStandard": {
      "@id": "https://w3id.org/idsa/core/representationStandard",
      "@type": "@id"
    },
    "modified": {
      "@id": "https://w3id.org/idsa/core/modified",
      "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
    },
```

```
100     "mediaType": {
101       "@id": "https://w3id.org/idsa/core/mediaType",
102       "@type": "@id"
103     },
104     "language": {
105       "@id": "https://w3id.org/idsa/core/language",
106       "@type": "@id"
107     },
108     "instance": {
109       "@id": "https://w3id.org/idsa/core/instance",
110       "@type": "@id"
111     },
112     "created": {
113       "@id": "https://w3id.org/idsa/core/created",
114       "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
115     },
116     "provider": {
117       "@id": "https://w3id.org/idsa/core/provider",
118       "@type": "@id"
119     },
120     "permission": {
121       "@id": "https://w3id.org/idsa/core/permission",
122       "@type": "@id"
123     },
124     "contractStart": {
125       "@id": "https://w3id.org/idsa/core/contractStart",
126       "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
127     },
128     "contractEnd": {
129       "@id": "https://w3id.org/idsa/core/contractEnd",
130       "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
131     },
132     "contractDate": {
133       "@id": "https://w3id.org/idsa/core/contractDate",
134       "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
135     },
136     "consumer": {
137       "@id": "https://w3id.org/idsa/core/consumer",
138       "@type": "@id"
139     },
140     "title": {
141       "@id": "https://w3id.org/idsa/core/title"
142     },
143     "description": {
144       "@id": "https://w3id.org/idsa/core/description"
145     },
146     "action": {
147       "@id": "https://w3id.org/idsa/core/action",
148       "@type": "@id"
```

```
    },
    "fileName": {
      "@id": "https://w3id.org/idsa/core/fileName"
    },
    "creationDate": {
      "@id": "https://w3id.org/idsa/core/creationDate",
      "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
    },
    "checkSum": {
      "@id": "https://w3id.org/idsa/core/checkSum"
    },
    "byteSize": {
      "@id": "https://w3id.org/idsa/core/byteSize",
      "@type": "http://www.w3.org/2001/XMLSchema#integer"
    },
    "filenameExtension": {
      "@id": "https://w3id.org/idsa/core/filenameExtension"
    },
    "accessURL": {
      "@id": "https://w3id.org/idsa/core/accessURL",
      "@type": "@id"
    },
    "version": {
      "@id": "https://w3id.org/idsa/core/version"
    },
    "keyword": {
      "@id": "https://w3id.org/idsa/core/keyword"
    },
    "representation": {
      "@id": "https://w3id.org/idsa/core/representation",
      "@type": "@id"
    },
    "contractOffer": {
      "@id": "https://w3id.org/idsa/core/contractOffer",
      "@type": "@id"
    },
    "standardLicense": {
      "@id": "https://w3id.org/idsa/core/standardLicense",
      "@type": "@id"
    },
    "resourceEndpoint": {
      "@id": "https://w3id.org/idsa/core/resourceEndpoint",
      "@type": "@id"
    },
    "owl": "http://www.w3.org/2002/07/owl#",
    "ids": "https://w3id.org/idsa/core/"
  }
}
```

Listing 5.19: Metadata Broker Description

5.5.4.9 Experiment 9: Create a Broker

Experiment 9 was developed to verify if creating a Broker on a connector (*POST /api/Brokers*) would be enough for the connector to appear when querying said Broker.

Firstly, the subcontracted Connector was used to create a Broker. Secondly, on the producer Connector, a query (listing 5.18) was run to check if the Broker appeared or not. Figure 5.19 has the query's response and the experiment result, showing that creating the Broker entity is not enough for the Connector to appear when querying said Broker.

As the ideal result when querying the Broker was for the subcontracted to appear, another step was employed to make that happen. Endpoint connector update (*POST /api/ids/connector/update*) was used on the subcontracted to register a connector on the Broker. Figure 5.20 shows the Broker changes after rerunning the query.

This experiment proves that creating a Broker entity is only helpful for having an ID to update a Connector (*POST /api/ids/connector/update*).

Figure 5.19: Connectors Registered on broker

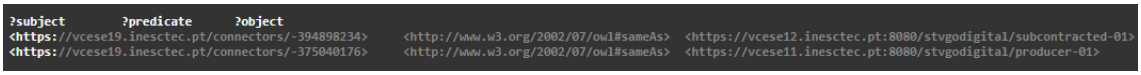


Figure 5.20: Connectors Registered on broker

5.5.4.10 Experiment 10: Register Connector to a Broker without creating a Broker

This experiment was developed to investigate if creating a Broker was indeed necessary.

First, the query described on listing 5.18, named Get available connectors, was executed to guarantee that the subcontracted Connector was not registered on the Broker.

After that, the experiment itself started. On the subcontracted Connector, the endpoint *POST /api/ids/connector/update* was executed with *https://vcese19.inesctec.pt:443/infrastructure* as the recipient instead of a Broker id. The response was a code 200, which means all was successful. Lastly, the query from step one was again executed on the producer Connector to see if the subcontracted appeared.

Figure 5.21 demonstrates that creating a Broker entity is not necessary. However, it facilitates referencing the Broker since, without a Broker entity, it is necessary to use the URL when referencing it.

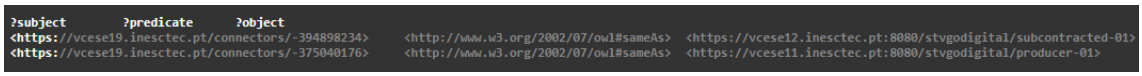


Figure 5.21: Connectors Registered on broker

5.5.4.11 Experiment 11: Querying Broker without subscribing to it

Experiment 11 was developed to test if it is possible to query a Broker that the Connector is not subscribed to. For this experiment, the subcontracted connector was used.

The experiment used the endpoint *POST /api/ids/query* with *https://vcese19.inesctec.pt:443/infrastructure* as the parameter and the query from listing 5.18, Get available connectors, on the request body.

Figure 5.22 shows the response, which proves that it is unnecessary to subscribe to a Broker for querying it.

Figure 5.22: Connectors Registered on broker

5.5.4.12 Experiment 12: Offers persistence

This experiment was designed to check if turning the connectors down impacted the offered resources available in the Broker.

First, the resources were registered on the Broker using endpoint *POST /api/ids/resource/update*, followed by endpoint *GET /api/brokers/{id}/offers* to see the Brokers resources. Then, query Get the title of the available resource was used to compare the result with the previous endpoint.

Secondly, the connector was turned off, waited 1 minute and then turned on.

Lastly, the Broker was queried with the same query as before, listing 5.15, to see if the resources appear or not and then with endpoint *GET /api/brokers/{id}/offers*. The responses are figures 5.23 and 5.24, respectively. There it is possible to see the successful response.

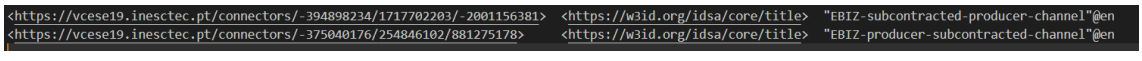


Figure 5.23: Available Offers response after turning connector down

5.5.4.13 Experiment 13: Connector-Broker persistence

Following the previous experiment, Experiment 13: Connector-Broker persistence was developed to see if removing the connector volumes creates any inconsistency.

The steps to test the persistence were similar to those from the previous experiment. First, the resources were registered on the Broker using endpoint *POST /api/ids/resource/update*, followed by endpoint *GET /api/brokers/{id}/offers* to see the Brokers resources. Then, the query Get the

```
{
  "_embedded": {
    "resources": [
      {
        "creationDate": "2022-06-07T12:14:22.266+0000",
        "modificationDate": "2022-06-07T12:14:22.266+0000",
        "title": "EBIZ-producer-subcontracted-channel",
        "description": "EBIZ-producer-subcontracted-channel resource holding data sent from 'producer' towards 'subcontracted'",
        "keywords": [
          "EBIZ"
        ],
        "publisher": "producer",
        "language": "pt-PT",
        "license": "http://www.stvgodigital.pt/",
        "version": 1,
        "sovereign": "producer",
        "endpointDocumentation": "endpointDocumentation",
        "paymentModality": "undefined",
        "samples": [],
        "additional": {
          "bootstrapId": "bootstrapId"
        },
        "_links": {
          "self": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/61b61528-d379-42c4-ae8c-e12c13892530"
          },
          "contracts": {
```


Figure 5.24: Offers registered on Metadata Broker after turning connector down

title of the available resource was used to compare the result with the previous endpoint. The results are shown in figures 5.25 and 5.26, respectively.

Secondly, the connectors were turned off and the volumes removed. After 1 minute, the connectors were turned back on.

Lastly, the Broker was created and then queried with the query Get the title of the available resource to see if the resources appear or not, and then with endpoint *GET /api/brokers/{id}/offers*. The responses are figures 5.27 and 5.28 on pages 85 and 86, respectively. They are both empty responses, which is both consistent and correct since when the connector's volumes are removed, the resources should also be removed from the Broker.

5.5.4.14 Experiment 14: Connector-Broker persistence: Remove Broker volumes

This experiment was developed to test what would happen if only the Broker volumes were eliminated.

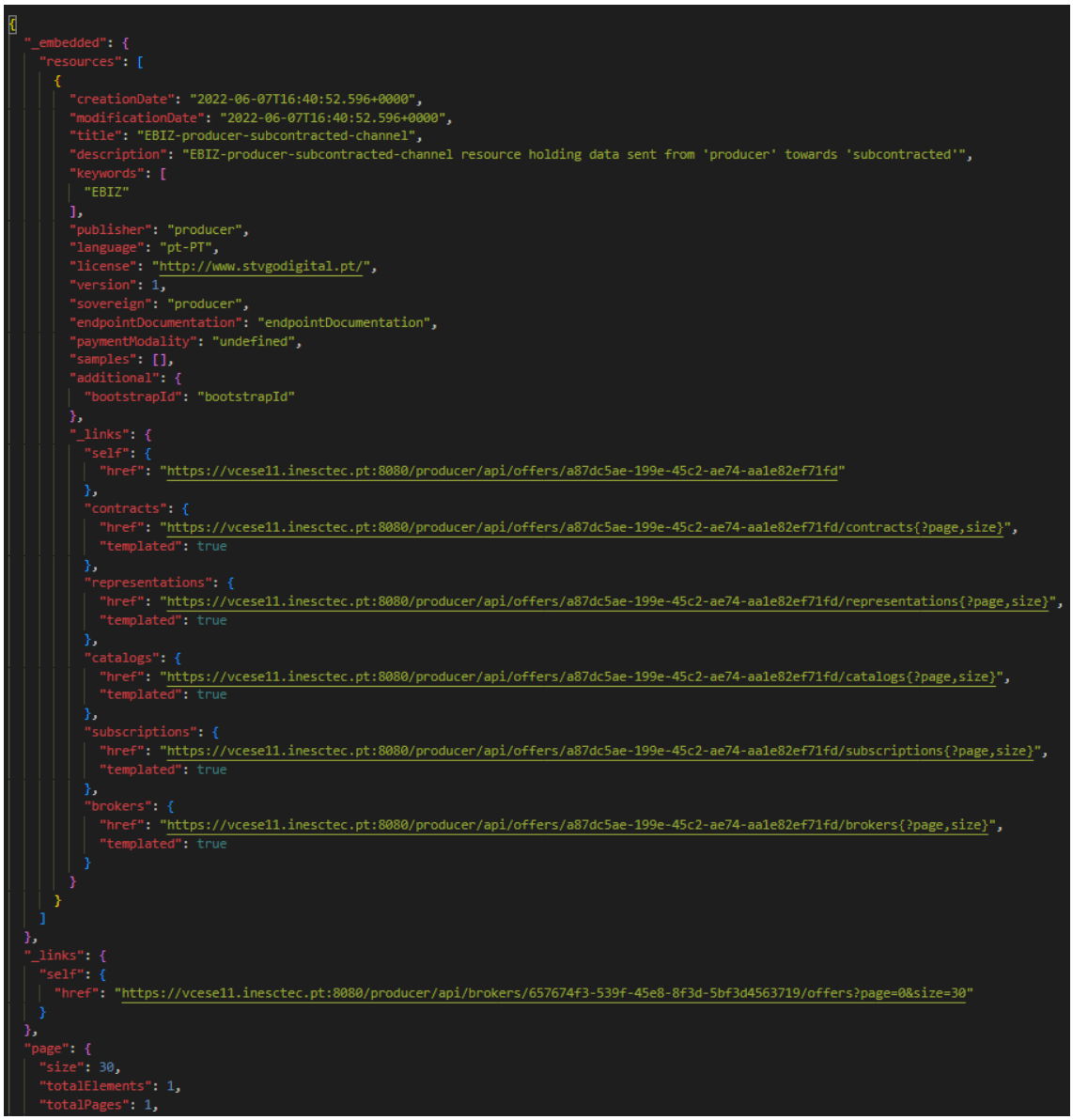
To replicate this experiment, Broker has to have already resources registered.

Secondly, the Metadata Broker was turned off and the volumes removed. One minute waited, and then the Broker was turned back on.

Lastly, the producer connector queried the Broker with the query Get the title of the available resource to see if the resources appear or not and then with endpoint *GET /api/brokers/{id}/offers*. The responses are figures 5.29 and 5.30, respectively.

There, it is possible to see the different responses, with 5.29 throwing an error and 5.30 being successful when the same response was expected. This shows that the Broker information on the connector is outdated, considering it says that offers are registered on the Broker and that the Broker is registered on the connector.

To solve the discrepancies it was used endpoints *POST /api/ids/connector/update* and *POST /api/ids/resource/update* to register the connector at the Broker again and then the resources, and



```
{
  "_embedded": {
    "resources": [
      {
        "creationDate": "2022-06-07T16:40:52.596+0000",
        "modificationDate": "2022-06-07T16:40:52.596+0000",
        "title": "EBIZ-producer-subcontracted-channel",
        "description": "EBIZ-producer-subcontracted-channel resource holding data sent from 'producer' towards 'subcontracted'",
        "keywords": [
          "EBIZ"
        ],
        "publisher": "producer",
        "language": "pt-PT",
        "license": "http://www.stvgodigital.pt/",
        "version": 1,
        "sovereign": "producer",
        "endpointDocumentation": "endpointDocumentation",
        "paymentModality": "undefined",
        "samples": [],
        "additional": {
          "bootstrapId": "bootstrapId"
        },
        "_links": {
          "self": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/a87dc5ae-199e-45c2-ae74-aa1e82ef71fd"
          },
          "contracts": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/a87dc5ae-199e-45c2-ae74-aa1e82ef71fd/contracts{?page,size}",
            "templated": true
          },
          "representations": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/a87dc5ae-199e-45c2-ae74-aa1e82ef71fd/representations{?page,size}",
            "templated": true
          },
          "catalogs": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/a87dc5ae-199e-45c2-ae74-aa1e82ef71fd/catalogs{?page,size}",
            "templated": true
          },
          "subscriptions": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/a87dc5ae-199e-45c2-ae74-aa1e82ef71fd/subscriptions{?page,size}",
            "templated": true
          },
          "brokers": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/a87dc5ae-199e-45c2-ae74-aa1e82ef71fd/brokers{?page,size}",
            "templated": true
          }
        }
      }
    ]
  },
  "_links": {
    "self": {
      "href": "https://vcese11.inesctec.pt:8080/producer/api/brokers/657674f3-539f-45e8-8f3d-5bf3d4563719/offers?page=0&size=30"
    }
  },
  "page": {
    "size": 30,
    "totalElements": 1,
    "totalPages": 1,
```


Figure 5.25: List of offers registered on a broker before removing connector volume

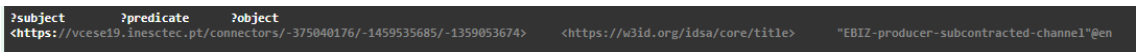


Figure 5.26: List of available offers before removing connector volume

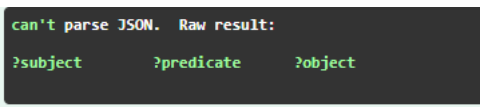


Figure 5.27: Empty broker

Response body


```
{
  "_embedded": {
    "resources": []
  },
  "_links": {
    "self": {
      "href": "https://vcese11.inesctec.pt:8080/producer/api/brokers/20c7b755-9daf-40ca-bd75-15311bd1a4c5/offers?page=0&size=30"
    }
  },
  "page": {
    "size": 30,
    "totalElements": 0,
    "totalPages": 0,
    "number": 0
  }
}
```


Figure 5.28: Empty broker

then both previous tests were run. While the endpoint *GET /api/brokers/{id}/offers* response was the same (as it should be), the query response was now successful, as seen in figure 5.31.

This unexpected behaviour proves that the Metadata Broker and the Dataspace Connector are not yet ready to use without caution.


```
?subject        ?predicate      ?object
<https://vcese19.inesctec.pt/connectors/-375040176/-1459535685/-1359053674>    <https://w3id.org/idsa/core/title>    "EBIZ-producer-subcontracted-channel"@en
```


Figure 5.29: Listing 5.15 response after removing Broker volume

417

Error: response status is 417

Response body


```
{
  "details": {
    "reason": {
      "properties": null,
      "@id": "https://w3id.org/idsa/code/NOT_FOUND"
    },
    "payload": "The index is empty - your query could not be evaluated.",
    "type": "de.fraunhofer.iais.eis.RejectionMessageImpl"
  },
  "message": "Received unexpected response message."
}
```


Figure 5.30: *GET /api/brokers/{id}/offers* response after removing broker volume

5.5.4.15 Experiment 15: Register resources

This experiment tests registering a resource on a connector when the connector is already registered on a Broker. It aims to check if an offer is automatically made available to a Broker.

First, the steps to guarantee that a resource is registered on a Broker were executed, steps 1 to 3 from section Testing procedures on page 66.

```
?subject  ?predicate  ?object
<https://vcese19.inesctec.pt/connectors/-375040176/-1271902210/143002872> <https://w3id.org/idsa/core/title>  "EBIZ-producer-subcontracted-channel"@en
```

Figure 5.31: Listing 5.15 response after updating the broker

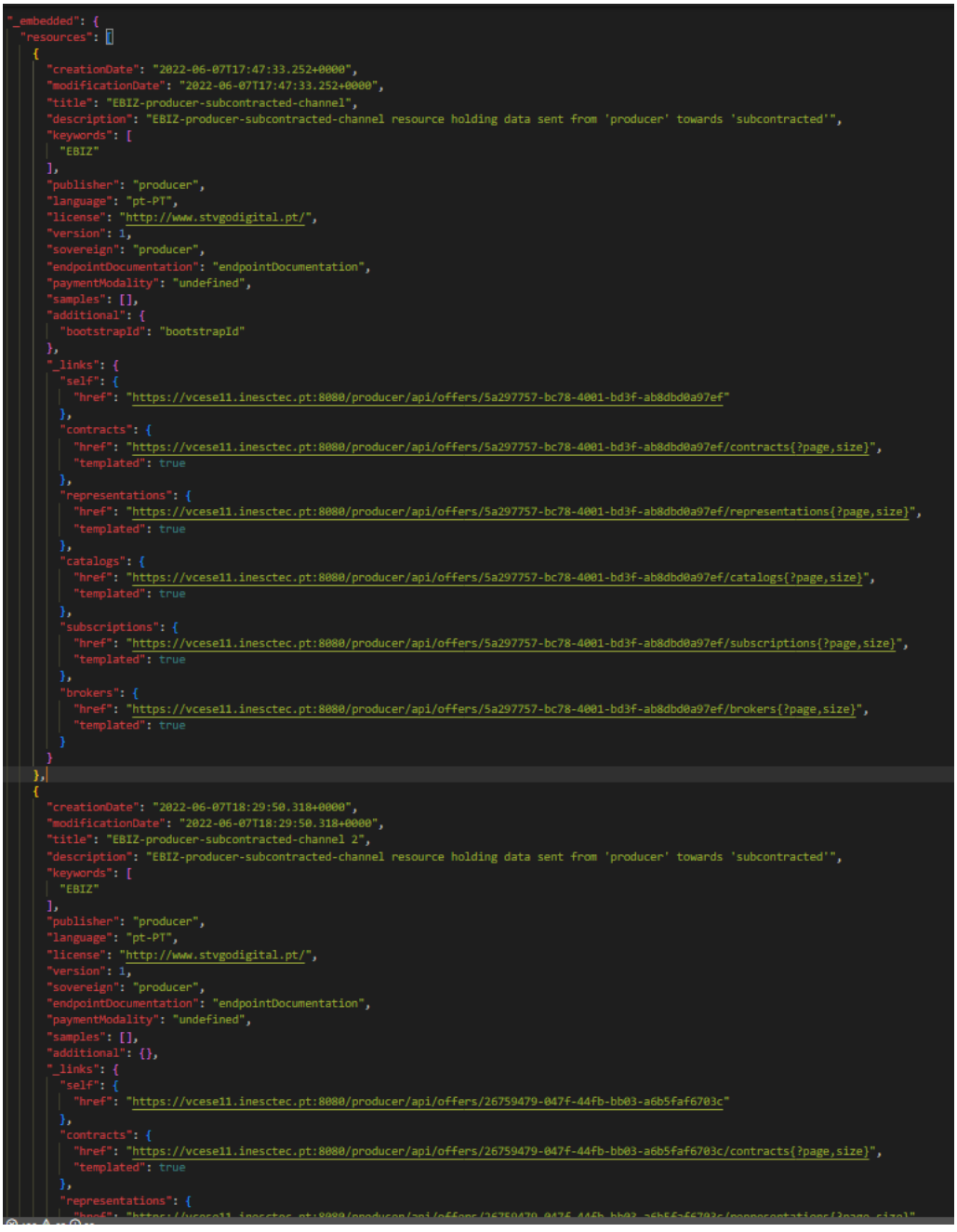
Then, a new offer was created to check if the resource was available to the Broker. This included creating Resource Offers, Representations, Artefacts, Contracts, and Rules and then connecting these structures. Next, the Broker was queried to see if the new resource was available. Figure 5.32 shows that it is not.


```
{
  "_embedded": {
    "resources": [
      {
        "creationDate": "2022-06-07T17:47:33.252+0000",
        "modificationDate": "2022-06-07T17:47:33.252+0000",
        "title": "EBIZ-producer-subcontracted-channel",
        "description": "EBIZ-producer-subcontracted-channel resource holding data sent from 'producer' towards 'subcontracted'",
        "keywords": [
          "EBIZ"
        ],
        "publisher": "producer",
        "language": "pt-PT",
        "license": "http://www.stvgodigital.pt/",
        "version": 1,
        "sovereign": "producer",
        "endpointDocumentation": "endpointDocumentation",
        "paymentModality": "undefined",
        "samples": [],
        "additional": {
          "bootstrapId": "bootstrapId"
        },
        "_links": {
          "self": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/5a297757-bc78-4001-bd3f-ab8dbd0a97ef"
          },
          "contracts": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/5a297757-bc78-4001-bd3f-ab8dbd0a97ef/contracts{?page,size}",
            "templated": true
          },
          "representations": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/5a297757-bc78-4001-bd3f-ab8dbd0a97ef/representations{?page,size}",
            "templated": true
          },
          "catalogs": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/5a297757-bc78-4001-bd3f-ab8dbd0a97ef/catalogs{?page,size}",
            "templated": true
          },
          "subscriptions": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/5a297757-bc78-4001-bd3f-ab8dbd0a97ef/subscriptions{?page,size}",
            "templated": true
          },
          "brokers": {
            "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/5a297757-bc78-4001-bd3f-ab8dbd0a97ef/brokers{?page,size}",
            "templated": true
          }
        }
      }
    ]
  },
  "_links": {
    "self": {
```


Figure 5.32: Offers registered on broker

Finally, the new resource was registered on the Broker. Figure 5.33 shows an excerpt of the new response, where more resources appear.

This experiment result was as expected, given that the resources should not be automatically available on a Broker. Furthermore, the data provider should always have the option to choose



```
"_embedded": {
  "resources": [
    {
      "creationDate": "2022-06-07T17:47:33.252+0000",
      "modificationDate": "2022-06-07T17:47:33.252+0000",
      "title": "EBIZ-producer-subcontracted-channel",
      "description": "EBIZ-producer-subcontracted-channel resource holding data sent from 'producer' towards 'subcontracted'",
      "keywords": [
        "EBIZ"
      ],
      "publisher": "producer",
      "language": "pt-PT",
      "license": "http://www.stvgodigital.pt/",
      "version": 1,
      "sovereign": "producer",
      "endpointDocumentation": "endpointDocumentation",
      "paymentModality": "undefined",
      "samples": [],
      "additional": {
        "bootstrapId": "bootstrapId"
      },
      "_links": {
        "self": {
          "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/5a297757-bc78-4001-bd3f-ab8dbd0a97ef"
        },
        "contracts": {
          "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/5a297757-bc78-4001-bd3f-ab8dbd0a97ef/contracts{?page,size}",
          "templated": true
        },
        "representations": {
          "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/5a297757-bc78-4001-bd3f-ab8dbd0a97ef/representations{?page,size}",
          "templated": true
        },
        "catalogs": {
          "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/5a297757-bc78-4001-bd3f-ab8dbd0a97ef/catalogs{?page,size}",
          "templated": true
        },
        "subscriptions": {
          "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/5a297757-bc78-4001-bd3f-ab8dbd0a97ef/subscriptions{?page,size}",
          "templated": true
        },
        "brokers": {
          "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/5a297757-bc78-4001-bd3f-ab8dbd0a97ef/brokers{?page,size}",
          "templated": true
        }
      }
    },
    {
      "creationDate": "2022-06-07T18:29:50.318+0000",
      "modificationDate": "2022-06-07T18:29:50.318+0000",
      "title": "EBIZ-producer-subcontracted-channel 2",
      "description": "EBIZ-producer-subcontracted-channel resource holding data sent from 'producer' towards 'subcontracted'",
      "keywords": [
        "EBIZ"
      ],
      "publisher": "producer",
      "language": "pt-PT",
      "license": "http://www.stvgodigital.pt/",
      "version": 1,
      "sovereign": "producer",
      "endpointDocumentation": "endpointDocumentation",
      "paymentModality": "undefined",
      "samples": [],
      "additional": {},
      "_links": {
        "self": {
          "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/26759479-047f-44fb-bb03-a6b5faf6703c"
        },
        "contracts": {
          "href": "https://vcese11.inesctec.pt:8080/producer/api/offers/26759479-047f-44fb-bb03-a6b5faf6703c/contracts{?page,size}",
          "templated": true
        },
        "representations": {
```


Figure 5.33: Offers registered on broker

which resources are available in each Broker, if any.

5.5.5 Adaption to other environments

This section shows the steps to implement this Broker in a different environment. Since the Broker changed virtual machines quite often, some changes needed to be made to swap environments. This information can be helpful when deploying another Broker on a different virtual machine.

The files that need swapping on the bind mounts are the *isstBroker-keystore.jks*, *server.crt* and *server.key*. The *isstBroker-keystore.jks* is the certificate assigned by IDS to, in this instance, the *vcese19.inesctec.pt* virtual machine in the *jks* format. The *server.crt* and *server.key* files are the certificates created by INESC TEC to the virtual machine **vcese19.inesctec.pt**.

5.6 App Store

The App Store behaves like a central repository for Data Apps. Essentially, after making an app available, any connector that uses that App Store can download that app. Right now, it is only possible to download docker images.

Four components comprise the IDS App Store environment:

- **Harbor Registry:** For uploading and downloading app images using the Docker protocol.
- **App Store:** Managing metadata about Data Apps.
- **Client Connector:** This connector is the one that wants to search and download an app.
- **Portainer:** The client portainer. It takes care of Docker container management.

The connections between the components are shown in the following figure taken from the International Data Spaces Association, App Store Implementation Documentation[5]:

The AppStore project is still under development by the Fraunhofer FIT[6], and is based on the Dataspace Connector open source project.

5.6.1 Installation Guide

The first step in the App Store installation is installing and configuring Harbor. For the App Store to run, it is necessary to have already a Harbor registry running on a machine. Both the App Store and the Harbor were installed on virtual machine *vcese19.inesctec.pt*.

Only after the Harbor is configured can an App Store instance work.

A Portainer container on the client (i.e. connector) machine was also implemented. Despite both virtual machines, *vcese11.inesctec.pt* and *vcese12.inesctec.pt*, already having a Portainer container from another project, it was necessary to make some alterations.

[5] https://international-data-spaces-association.github.io/IDS-AppStore/deployment
[6] https://www.fit.fraunhofer.de/en.html

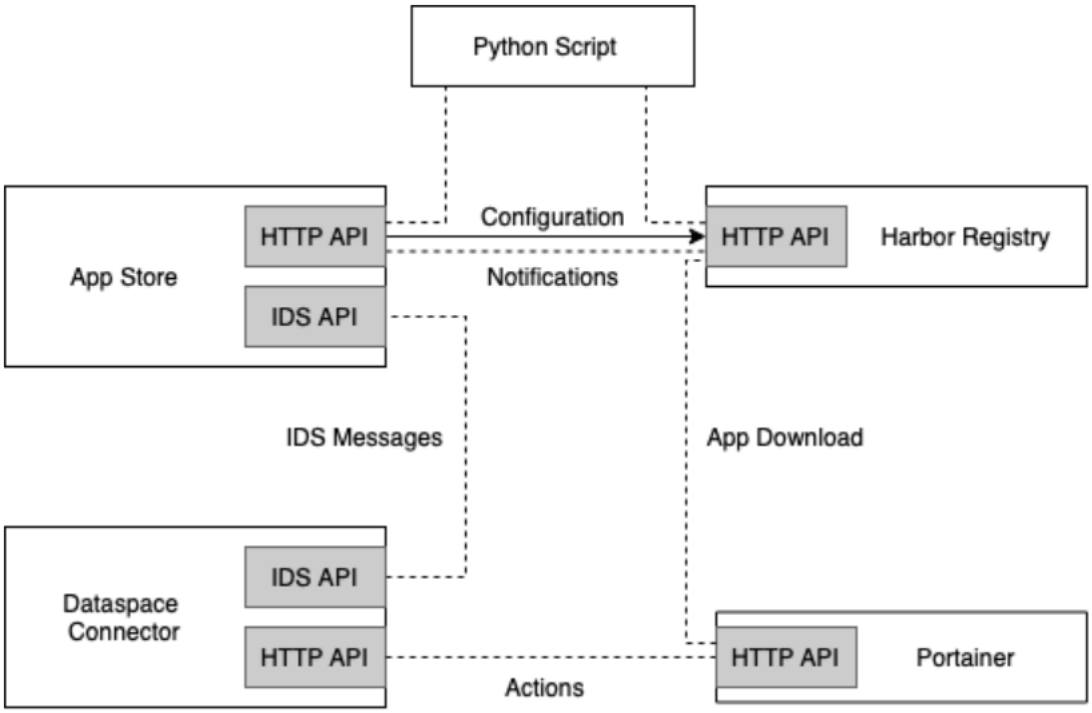


Figure 5.34: App Store Components Connections

- **Harbor**

  The Harbor registry was downloaded from GitHub and extracted to the virtual machine.

  Next, it was necessary to configure the HTTPS access to Harbor by following the guide from `https://goharbor.io/docs/2.4.0/install-config/configure-https/`. This step was done by using the certificates created by INESC TEC. Appendix E, Harbor Configuration, describes the detailed steps for the Harbor configuration and deployment.

- **AppStore**

  Considering the App Store is developed to run using a *jar file*, the steps to install and deploy one are below:

  - Change the *application.properties* file to have the correct values. This includes the Harbor and App Store URL, App Store ports, Harbor registry host, API URL, username, password, project name, certificates names and passwords and Swagger page credentials;
  - Create and upload the correct certificates for the virtual machine;
  - Configure the IDS certification, due to the App Store being a specialization of the Dataspace Connector. This is done by adding the *vcese19.inesctec.pt* IDS-issued certificate to the App Store configuration folder, *conf*. It is also necessary to change the App Store configuration file, *config.json*, to include *vcese19.inesctec.pt* as the ID and endpoint instead of *Fraunhofer Appstore*;

– Run Maven to generate a new *jar* file and then run the file 5.20

```
sudo chmod +x mvnw
sudo ./mvnw clean package -DskipTests -Dmaven.javadoc.skip=true
cd /target
sudo java -jar appstore-2.0.0.jar
```

Listing 5.20: Run AppStore with Jar file

Given that INESC TEC wants to use the International Data Spaces Association App Store as is, some changes were applied to be able to run the code with a docker-compose and without modifications to the original repository code. The changes are described below:

– Utilize environment variables to override the *application.properties* changes. Those were created on file *connector.env*, loaded on the docker-compose, listing C.2;

– Upload the App Store configuration folder into the deploy root folder. This folder will have a *config.json* file, an IDS-issued certificate, and a TrustStore file. The *config.json* will be uploaded into the App Store container as a volume and overrides the original configuration file;

– Create an App Store Docker image and publish it on a Docker Hub repository. This image is the one used on the docker-compose;

– Change or create a reverse proxy. Despite the repository not being ready to run as a docker application, they give a basic docker-compose and reverse proxy. This proxy was later on exchanged with one explicitly created to test the App Store (a detailed description of the reason is in section 5.6.3.4);

– Create and upload the correct certificates for the virtual machine and use them as bind mounts for the reverse proxy.

The final docker-compose and environment variables files are shown on Appendix C.

- **Portainer**

As said before, the virtual machines running the client connectors already had a portainer instance from another project. So, that portainer was separated from the project and instantiated on a docker-compose specific for this purpose. This way, the virtual machine only has a portainer instantiated, and both projects can use it.

However, when using it, some problems were noticed. Those and their solutions are listed and described on Appendix D.

5.6.2 Testing procedures

To deploy the App Store environment is first necessary to create a Webhook on the Harbor registry page. The Webhook URL should be `https://vcese19.inesctec.pt:9443/api/webhook/registry`, with the port being the App Store port.

Second comes uploading an app to the App Store. Currently, the App Store only allows uploading apps using a python script, *create_resources_and_upload_app_local.py*. To run it, it is necessary to go to the project root directory and run:

python3 ../scripts/tests/create_resources_and_upload_app_local.py

The App Store testing with the client connector is done via the Connector and App Store Swagger pages. Furthermore, the steps to test the App Store are as follows, on the client connector:

1. **Create App Store:** *POST /api/appstore*

   Request Body:

```
{
  "title": "App Store",
  "description": "App Store",
  "location": "https://vcese19.inesctec.pt:9443/api/ids/data"
}
```

Listing 5.21: Create App Store Request Body

2. **Query AppStore** on endpoint *POST /api/ids/description*:

   Recipient URL: `https://vcese19.inesctec.pt:9443/api/ids/data`

   ElementID: `https://vcese19.inesctec.pt:9443/api/catalogs/<CATALOG_ID>`

   The *ElementID* was taken from endpoint *GET /api/catalogs* from the App Store swagger page.

3. **Download App** from App Store *POST /api/ids/app*:

   Recipient URL: `https://vcese19.inesctec.pt:9443/api/ids/data`

   AppID: This ID is taken from field *ids:AppResource* from the previous endpoint response.

5.6.3 Challenges Encountered

Given that there was not much documentation regarding the App Store deploy and use, some challenges were encountered.

The sections below describe the challenges and solutions used.

5.6.3.1 Challenge 1: Certificates only work with *vcese18.inesctec.pt* name

The virtual machine *vcese18.inesctec.pt* was very isolated. So, most of the time, the machine hostname was not recognised.

Because of that, its IP address was used on the Dataspace connectors instead of its hostname. However, as the certificates were being verified and the IP was not on the certificates, an error was thrown.

The solution found was to pass the App Store to another virtual machine. Later, a less isolated machine, *vcese19.inesctec.pt*, was created, and the components (Clearing House, Metadata Broker, App Store and DAPS) were moved there.

5.6.3.2 Challenge 2: Certificate not validated on the browser

Despite configuring the certificates as the App Store documentation demanded, the browsers still could not validate them since it said there was insufficient information. This behaviour was happening with the Harbor and the App Store containers.

To solve this problem was created and used a *pem* file containing the certificate and key instead of the *cert* file documented.

5.6.3.3 Challenge 3: Proxy not connecting with Swagger links

The Nginx proxy provided by the IDSA in the repository could not correctly connect with Swagger.

When using `https://vcese19.inesctec.pt:9443/api/docs`, the proxy converted the link correctly and the page opened. However, the container could not resolve the name: */v3/api-docs* which was defined on the proxy configuration file. If changed manually to */v3/api-docs/*, it connected. As this link is generated automatically, changing it on the code was impossible.

The solution found was to change the *location /v3/api-docs/* to *location /v3/api-docs* and the *location /api/docs/* to *location /api/docs*

5.6.3.4 Challenge 4: Error uploading apps with python script when using proxy container

When using the docker-compose and proxy provided by the International Data Spaces Association, a connection error was thrown when trying to upload apps to the App Store with the python script.

After searching, it was noted that the server URL generated lacked the port, so the App Store tried to connect with *vcese19.inesctec.pt* instead of *vcese19.inesctec.pt:9443*. The logs generated by the python script follow:

```
1 'https://vcese19.inesctec.pt/api/catalogs/21a01a10-f655-4982-a8a5-c7468ddf674d'
2 'https://vcese19.inesctec.pt/api/resources/8455c917-85ac-46f5-bcdb-1828b005565f'
3 'https://vcese19.inesctec.pt/api/representations/ca8e5782-1069-4f09-b222-04366
      dece571'
4 'https://vcese19.inesctec.pt/api/apps/8aad5561-e9f1-4db7-843b-5c56f851ebf4'
```

```
5 'https://vcese19.inesctec.pt/api/endpoints/bfa69fc8-616e-423f-81cb-b5a1a1dc64dd'
6 'https://vcese19.inesctec.pt/api/artifacts/d2b9a2f3-a4cf-4486-9c55-acc3e1f14da5'
7 'https://vcese19.inesctec.pt/api/contracts/4510284f-f317-40ed-9cd4-cf5200f971da'
8 'https://vcese19.inesctec.pt/api/rules/84d5a572-4c7f-43fe-8a23-1ce5e1209ad3'
```

When searching the App Store GitHub issues pages, one already addressed this problem: https://github.com/International-Data-Spaces-Association/IDS-AppStore/issues/53. However, it did not gave any solution.

As the python script and the application work with the *jar* file, the problem could only be on the docker-compose, dockerfile or proxy configuration.

When studying all these files, it was decided that the proxy was the more likely to cause this issue since the previous challenge showed that it was not working as intended. So, another proxy was created to test if the problem was indeed generated by it. This new proxy was tested and was able to connect with the correct URL.

From this challenge, it was seen that the proxy configuration file supplied by the IDSA has some errors in its configuration.

5.6.3.5 Challenge 5: Inbound Model incompatibility

Given that the App Store does not support the latest Inbound Model version, incompatibility problems arise when trying to communicate with connectors that use the latest version. This error was first encountered when trying to query the App Store on the connector endpoint *GET /ids/api/description*.

This issue[7] is already registered on the App Store GitHub page and in the process of being resolved by the developers. However, it was first encountered on March 30 and is still not ready.

As there is still no solution, a quick fix was created. It consists in downgrading the connector Outbound Model Version to coincide with the App Store Inbound Model versions it supports. However, downgrading is not a solution and was only used to test this component.

5.6.3.6 Challenge 6: Error downloading App

When trying to download an app using the connector endpoint *POST /api/ids/app* the following error is thrown: **ERROR - An unhandled exception has been caught. [exception=(Cannot invoke "java.net.URI.toString()" because "uri" is null)]**

There was already a closed issue on GitHub that reported this problem, and the solution the App Store developers gave was to separate the App Store and the Harbor from the Dataspace Connector. Furthermore, the issue creator did not comment if it resolved the issue or not. The developers closed it after responding.

A new issue was created since the App Store and Harbor were already separated from the connectors. Nevertheless, it was tried to separate the App Store and Harbor to test if it resolved the issue.

[7] https://github.com/International-Data-Spaces-Association/IDS-AppStore/issues/65

It is important to note that when testing it, the connector could download an app once, as shown in figure 5.35, and when trying to repeat the action, the error returned. This led to conclude that this behaviour comes from a bug in the code.

5.6.4 Experiment 1: Download an App

This experiment is the App Store test. Moreover, the steps were the ones explained in the section Testing procedures on page 92. This experiment uses the subcontracted Connector.

First, the App Store was created using the values shown on listing 5.21.

Second, it was queried using endpoint *POST /api/ids/description* to see which apps it offers. The experiment response is depicted in listing 5.22.

```
{
  "@context": {
    "ids": "https://w3id.org/idsa/core/",
    "idsc": "https://w3id.org/idsa/code/"
  },
  "@type": "ids:ResourceCatalog",
  "@id": "https://vcese19.inesctec.pt:9443/api/catalogs/1dad314e-ca24-404c-97e4
      -350134d339aa",
  "ids:offeredResource": [
    {
      "@type": "ids:AppResource",
      "@id": "https://vcese19.inesctec.pt:9443/api/resources/e799241f-b2bf-44e2-
          a4ca-1705295ee491",
      "ids:language": [
        {
          "@id": "https://w3id.org/idsa/code/EN"
        }
      ],
      "ids:version": "1",
      "ids:title": [
        {
          "@value": "DataProcessingApp",
          "@language": "EN"
        }
      ],
      "ids:theme": [],
      "ids:description": [
        {
          "@value": "data app for processing data.",
          "@language": "EN"
        }
      ],
      "ids:representation": [
        {
          "@type": "ids:AppRepresentation",
```

```
          "@id": "https://vcese19.inesctec.pt:9443/api/representations/58116ec3
              -6884-4757-a656-21a2cfb525ae",
          "ids:instance": [
            {
              "@type": "ids:Artifact",
              "@id": "https://vcese19.inesctec.pt:9443/api/artifacts/f480a395-c6cc
                  -47b6-a6e7-beeac6236bf5",
              "ids:fileName": "DataApp Template",
              "ids:creationDate": {
                "@value": "2022-06-09T15:06:42.342Z",
                "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
              },
              "ids:byteSize": 0,
              "ids:checkSum": "0"
            }
          ],
          "ids:language": {
            "@id": "https://w3id.org/idsa/code/EN"
          },
          "ids:mediaType": {
            "@type": "ids:IANAMediaType",
            "@id": "https://w3id.org/idsa/autogen/iANAMediaType/12f21475-7305-4ded
                -9bdd-d4e0030771bc",
            "ids:filenameExtension": ""
          },
          "ids:title": [],
          "ids:description": [],
          "ids:modified": {
            "@value": "2022-06-09T15:06:41.627Z",
            "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
          },
          "ids:created": {
            "@value": "2022-06-09T15:06:41.627Z",
            "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
          },
          "ids:dataAppRuntimeEnvironment": "docker",
          "ids:representationStandard": {
            "@id": ""
          },
          "ids:dataAppDistributionService": {
            "@id": "https://app.registry.example.org"
          },
          "ids:dataAppInformation": {
            "@type": "ids:SmartDataApp",
            "@id": "https://vcese19.inesctec.pt:9443/api/apps/a13c4070-abda-47da-95
                dd-2d772a9a7ea5",
            "ids:supportedUsagePolicies": [
              {
                "@id": "https://w3id.org/idsa/code/ALLOW_DATA_USAGE"
```

```
            }
          ],
          "ids:appEnvironmentVariables": "dbUser=sa;dbPasswd=passwd",
          "ids:appDocumentation": "App-related human-readable documentation.",
          "ids:appStorageConfiguration": "-v /data",
          "ids:appEndpoint": [
            {
              "@type": "ids:AppEndpoint",
              "@id": "https://vcese19.inesctec.pt:9443/api/endpoints/a4fc32ba-21
                  be-4593-a262-abdddff840d5",
              "ids:path": "/input/",
              "ids:appEndpointType": {
                "@id": "https://w3id.org/idsa/code/INPUT_ENDPOINT"
              },
              "ids:endpointDocumentation": [
                {
                  "@id": "https://app.swaggerhub.com/apis/app/1337"
                }
              ],
              "ids:appEndpointPort": 8080,
              "ids:appEndpointMediaType": {
                "@type": "ids:IANAMediaType",
                "@id": "https://w3id.org/idsa/autogen/iANAMediaType/a34532b2
                    -9075-4c0e-83f0-12df66d2ae98",
                "ids:filenameExtension": "application/json"
              },
              "ids:appEndpointProtocol": "HTTP/1.1",
              "ids:accessURL": {
                "@id": "/input"
              },
              "ids:endpointInformation": [
                {
                  "@value": "More information about the endpoint",
                  "@language": "EN"
                }
              ]
            }
          ],
          "language": "EN"
        }
      }
    ],
    "ids:publisher": {
      "@id": "https://fit.fraunhofer.de"
    },
    "ids:sovereign": {
      "@id": "https://fit.fraunhofer.de"
    },
    "ids:paymentModality": {
```

```
        "@id": "https://w3id.org/idsa/code/FREE"
      },
      "ids:modified": {
        "@value": "2022-06-09T15:06:41.083Z",
        "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
      },
      "ids:temporalCoverage": [],
      "ids:spatialCoverage": [],
      "ids:created": {
        "@value": "2022-06-09T15:06:41.083Z",
        "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
      },
      "ids:defaultRepresentation": [],
      "ids:keyword": [
        {
          "@value": "data",
          "@language": "EN"
        },
        {
          "@value": "processing",
          "@language": "EN"
        },
        {
          "@value": "fit",
          "@language": "EN"
        }
      ],
      "ids:contentPart": [],
      "ids:standardLicense": {
        "@id": "https://www.apache.org/licenses/LICENSE-2.0"
      },
      "ids:resourcePart": [],
      "ids:resourceEndpoint": [
        {
          "@type": "ids:ConnectorEndpoint",
          "@id": "https://w3id.org/idsa/autogen/connectorEndpoint/6f0cb8d1-10ec-41
              ee-be46-7a5bec16fe87",
          "ids:endpointDocumentation": [
            {
              "@id": ""
            }
          ],
          "ids:accessURL": {
            "@id": "https://vcese19.inesctec.pt:9443/api/resources/e799241f-b2bf-44
                e2-a4ca-1705295ee491"
          },
          "ids:endpointInformation": []
        }
      ],
```

```
    "ids:contractOffer": [
      {
        "@type": "ids:ContractOffer",
        "@id": "https://vcese19.inesctec.pt:9443/api/contracts/1f329eb1-f4ad-4f66
            -92e6-33f8006e8890",
        "ids:permission": [
          {
            "@type": "ids:Permission",
            "@id": "https://vcese19.inesctec.pt:9443/api/rules/4edfdc40-211d-4c92
                -80e2-70a727f4d76c",
            "ids:title": [
              {
                "@value": "Example Usage Policy",
                "@type": "http://www.w3.org/2001/XMLSchema#string"
              }
            ],
            "ids:description": [
              {
                "@value": "n-times-usage",
                "@type": "http://www.w3.org/2001/XMLSchema#string"
              }
            ],
            "ids:assignee": [],
            "ids:assigner": [],
            "ids:postDuty": [],
            "ids:constraint": [
              {
                "@type": "ids:Constraint",
                "@id": "https://w3id.org/idsa/autogen/constraint/e0a353a2-ef1d
                    -4932-b3cf-a5a0a5a1455e",
                "ids:operator": {
                  "@id": "https://w3id.org/idsa/code/LTEQ"
                },
                "ids:leftOperand": {
                  "@id": "https://w3id.org/idsa/code/COUNT"
                },
                "ids:rightOperand": {
                  "@value": "5",
                  "@type": "http://www.w3.org/2001/XMLSchema#double"
                }
              }
            ],
            "ids:action": [
              {
                "@id": "https://w3id.org/idsa/code/USE"
              }
            ],
            "ids:preDuty": []
          }
```

```
          ],
          "ids:provider": {
            "@id": ""
          },
          "ids:consumer": {
            "@id": ""
          },
          "ids:contractEnd": {
            "@value": "2021-12-06T11:33:44.995Z",
            "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
          },
          "ids:prohibition": [],
          "ids:obligation": [],
          "ids:contractStart": {
            "@value": "2021-12-06T11:33:44.995Z",
            "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
          },
          "ids:contractDate": {
            "@value": "2022-06-09T15:11:11.282Z",
            "@type": "http://www.w3.org/2001/XMLSchema#dateTimeStamp"
          }
        }
      ],
      "ids:sample": []
    }
  ],
  "ids:requestedResource": []
}
```


Listing 5.22: App Store Description

Third, the app from the App Store was downloaded using endpoint *POST /api/ids/app*. For this example, the Recipient URL was `https://vcese11.inesctec.pt:9443/api/ids/data`, and the AppID was `https://vcese11.inesctec.pt:9443/api/resources/ff2148d7-8443-4e29-af7c-33c989`. The response is displayed in figure 5.35. This figure does not coincide with the previous listing since it was taken when testing the App Store on the virtual machine *vcese11.inesctec.pt*.

After this, the connectors should still be able to perform certain actions for downloaded apps. However, it was impossible to test past the app download functionality since it was when it failed every time.

The inability to reproduce all the steps from the Connector-App Store interaction concludes that the App Store still needs further maturing.

```
{
  "creationDate": "2022-06-10T14:24:51.315+0000",
  "modificationDate": "2022-06-10T14:24:51.315+0000",
  "title": "DataProcessingApp",
  "description": "data app for processing data.",
  "remoteId": "https://vcese11.inesctec.pt:9443/api/resources/ff2148d7-8443-4e29-af7c-33c9899575e1",
  "remoteAddress": "https://vcese11.inesctec.pt:9443/api/resources/ff2148d7-8443-4e29-af7c-33c9899575e1",
  "docs": "App-related human-readable documentation.",
  "envVariables": "dbUser=sa;dbPasswd=passwd",
  "storageConfig": "-v /data",
  "supportedPolicies": [
    "PROVIDE_ACCESS"
  ],
  "keywords": [
    "processing",
    "data",
    "fit"
  ],
  "publisher": "https://fit.fraunhofer.de",
  "sovereign": "https://fit.fraunhofer.de",
  "language": "https://w3id.org/idsa/code/EN",
  "license": "https://www.apache.org/licenses/LICENSE-2.0",
  "endpointDocumentation": "https://app.com",
  "distributionService": "https://app.registry.example.org",
  "runtimeEnvironment": "docker",
  "additional": {
    "ids:byteSize": "0",
    "ids:checkSum": "0",
    "ids:mediaType": "",
    "ids:fileName": "DataApp Template",
    "ids:language": "https://w3id.org/idsa/code/EN"
  },
  "_links": {
    "self": {
      "href": "https://vcese12.inesctec.pt:8080/subcontracted/api/apps/15d9df24-cd94-433b-a51d-8ea5da9ad50a"
    },
    "endpoints": {
      "href": "https://vcese12.inesctec.pt:8080/subcontracted/api/apps/15d9df24-cd94-433b-a51d-8ea5da9ad50a/endpoints{?page,size}",
      "templated": true
    }
  }
}
```


Figure 5.35: App Download Response

5.6.5 Adaption to other environments

This section shows the steps to implement this App Store in a different environment. Since the store changed virtual machines quite often, some changes needed to be made to swap environments. This information helps deploy an App Store on another virtual machine.

If the Harbor also changes machines, the file *harbor.yml* needs to be updated to swap the hostname, certificate name and key and port to the new one. The steps from the Harbor configuration depicted in appendix E also need to be repeated with the new machine certificates. The latter are the ones created by a CA for the virtual machine.

Regarding the App Store, the files updated are the *connector.env*, the proxy and the files on the connector configuration folder. The *connector.env* has the environment variables that override the *application.properties*. The variables containing the machine, port, the certificates names and passwords are updated on that file. These certificates are the ones created by Fraunhofer for the virtual machine.

The changes on the proxy file are related to the certificate name and ports, if necessary.

Finally, the files on folder *conf* need to be updated. This folder has files regarding the connector the App Store is based on. Moreover, The IDS certificate needs to be changed and the configuration file updated to have the new file.

5.7 Identity Provider

The Identity Provider is responsible for verifying the authenticity of the IDS participants and providing all the characteristics related to identity management.

It consists of three components: Certificate Authority (CA), Dynamic Attribute Provisioning Service (DAPS) and Participant Information System (ParIS).

The Participant Information System only has a few commits (i.e. 19 commits), so it was not considered ready to start implementing. Moreover, the Dataspace Connector does not offer any ParIS interaction[8].

The DAPS implementation was the responsibility of a project partner. Nevertheless, to replace the Fraunhofer DAPS, tests and changes needed to be made on the connectors from the existing environment. One change was the definition of environment variables that override the default DAPS configuration.

The DAPS repository[9] is being developed by IDSA and has two submodules, each developed by Fraunhofer-AISEC. The submodles are the omejdn-ui[10] and the omejdn-server[11]. The first is responsible for providing a user interface to interact with the DAPS. The latter has all the Omejdn part of the DAPS, which is used as the Dynamic Attribute Provisioning Service on the IDS prototype and is a minimal but extensible OAuth 2.0/OpenID connect server.

[8] https://international-data-spaces-association.github.io/DataspaceConnector/CommunicationGuide/v6/IdsEcosystem/IdentityProvider

[9] https://github.com/International-Data-Spaces-Association/omejdn-daps

[10] https://github.com/Fraunhofer-AISEC/omejdn-ui/tree/8837233a05808669c91556895ca8de730fd046c4

[11] https://github.com/Fraunhofer-AISEC/omejdn-server/tree/06a009da26d2543648c327fb826218ca410c0e2

This repository provides two test scripts that simulate the connector-DAPS interactions. One creates the certificates to send to the connectors, and the other simulates the connector behaviour to obtain the token.

Various experiments were made to replace the Fraunhofer component with the repository implementation since there was no documentation regarding the interaction with the connectors.

Despite the project partner having implemented the DAPS, he only implemented the server backend, not the DAPS user interface or a proxy to validate the machine certificates. This made it challenging to have the server constantly running or changing the configuration endpoint or ports, for instance. So, Experiment 1: Docker-compose setup was developed to make the DAPS run with docker-compose. The other reason for his change was the test scripts returning errors.

5.7.1 Testing procedures

This test procedure was discovered by experimenting with the connectors and the DAPS. As said before, there is no information regarding the steps to implement the DAPS with the Dataspace Connector or the other IDS components.

Therefore, these steps were discovered by developing and trying the experiments from section 5.7.3 and by solving the challenges depicted in the following section. Figures detailing the steps are shown on the section 5.7.3.

DAPS Server

1. Start DAPS with docker-compose.
2. Register component. Using the script by running *scripts/register_connector.sh <CONNECTOR_NAME> <VIRTUAL_MACHINE_NAME>*. Without the script by creating the certificates and then creating a verification key.
3. Send certificate and key to the client.

Connector

1. Import the certificate and key to the connector machine.
2. Create a *.p12* file using the provided certificates and the command shown in Experiment 3: Connector-DAPS Interaction.
3. Insert the file into the connector configuration folder.
4. Give permissions to file using command *chmod +rw <CONF_FOLDER>/<CERT_NAME>.p12*.
5. Update the configuration file (*config.json*) to use the file created.
6. Update the docker-compose to use the new *.p12* file name and password and redirect to the DAPS on virtual machine *vcese19.inesctec.pt*.
7. Remove volumes and restart.

Metadata Broker

1. Turn the broker off and remove its volumes.
2. Import the certificate, key and DAPS TLS certificate into the broker machine.
3. Convert the new certificates into *p12* format with the OpenSSL command: *openssl pkcs12 -export -name broker_cert -in <CERT_NAME>.crt -inkey <KEY_NAME>.key -out <NEW_FILE_NAME>.p12*
4. Insert the *p12* file into the *jks* by creating it with the command:
   keytool -importkeystore -destkeystore isstbroker-keystore.jks -srckeystore <FILE_NAME>.p12 -srcstoretype pkcs12 -alias broker_cert
5. Give the previously created files read permissions.
6. On the *broker-core/src/main/resources*:
   - Insert the *jks*.
   - Update the DAPS variables to the following values on file *application.properties*:

```
component.uri=http://vcese19.inesctec.pt:8080/
component.catalogUri=http://vcese19.inesctec.pt:8080/connectors/

# DAPS
daps.url=https://vcese19.inesctec.pt/auth/token
daps.validateIncoming=true

# Security-related
jwks.trustedHosts=daps.aisec.fraunhofer.de,omejdn,vcese19.inesctec.pt
ssl.certificatePath=/etc/cert/server.crt
```

7. Run the following commands on the project root directory to create a *jar* file:

```
mvn clean package
cp broker-core/target/broker-core-5.0.0-SNAPSHOT.jar docker/broker-core
```

8. On the *docker/broker-core* directory insert the *crt*, *jks* and *jar* files. If using DAPS with HTTPS, the *crt* file is the DAPS TLS certificate.
9. On the *docker/composefiles* folder:
   - Move the files sent by the broker and the *isstbroker-keystore.jks* into a new directory to be used as bind mount.
   - Update the docker-compose to redirect to the new DAPS, figure 5.42.
10. Change the *crt* and *key* files names to *server.crt* and *server.key*.
11. Restart the broker.

Clearing House

1. Turn the component down and remove the volumes.
2. Import the certificate and key into the CH machine.
3. Import the DAPS TLS certificate into the CH machine.
4. Create the *p12* file, guaranteeing the name is *1* by running the command:
   openssl pkcs12 -export -name 1 -in <CERT_NAME>.cert -inkey <KEY_NAME>.key -out <NEW_FILE_NAME>.p12
5. Create the *der* file with the command:
   openssl x509 -inform pem -in <CERT_NAME>.cert -outform der -out <NEW_FILE_NAME>.der
6. Replace the *keystore.p12* file on the trusted connector with the previously created *p12* file, as shown in figure 5.45 on page 116.
7. Replace the DAPS environment variable with *https://vcese19.inesctec.pt*.
8. Give read permissions to the new certificate files.
9. Insert the *crt* certificate into the folders where the dockerfiles are located and update the certificate name on the file, as seen in figure 5.46.
10. Update the *Rocket.toml* file to point to the new DAPS, as shown in figure 5.44.

5.7.2 Challenges

There was not much documentation regarding the DAPS interaction with the different IDS components, so some challenges were encountered.

The sections below describe the challenges and solutions used.

5.7.2.1 Challenge 1: Omejdn configuration file badly generated

The docker-compose setup uses an environment variables file to override the values on the docker-compose and on the omejdn configuration files. Despite this, some variables from the latter were updated with the variables' names instead of their values when the server was started. This means that instead of the variables being instantiated with **http://vcese19.inesctec.pt:4567/auth**, they were with **`${OMEJDN_PROTOCOL}://${OMEJDN_DOMAIN}${OMEJDN_PATH}`**.

This action made it impossible for the endpoints to work and resulted in errors when logging into the UI since they use the information from the variables *issuer* and *front_url* from the omejdn configuration file for the DAPS endpoints.

To solve this, the variable that generated the error was altered. Rather than being instantiated with the combination of variables, it was instantiated with their value. Listing 5.23 shows the final values.

```
1 issuer: http://vcese19.inesctec.pt:4567/auth
2 front_url: http://vcese19.inesctec.pt:4567/auth
```

Listing 5.23: *omejdn.yml* variables values

5.7.2.2 Challenge 2: Client unknown

When trying to update the connector certificates to use the newly created by the local DAPS, a **Client unknown** error was thrown on the connector logs.

After re-testing the DAPS implementation and guaranteeing that the client ID was indeed on the DAPS client's file, it was known the error was on the connector part.

The solution found was to remove the volumes. Despite the connector docker-compose being built and the certificate imported, the certificate on the connector configuration file was not overridden. After removing and then restarting the volumes, the problem was solved.

5.7.2.3 Challenge 3: Could not find KeyStore at system scope

This challenge arose when the connectors were trying to import the certificates.

It was discovered that the certificate and the directory where it is stored need the *+rw* permissions. After giving them, the connector volumes were removed, and the docker-compose was rebuilt without throwing the error.

5.7.2.4 Challenge 4: No verification key available

This problem arose when making Experiment 4: Create certificates without the script. As this experiment consisted of trying to create the certificates without using the provided script, this error was thrown.

After searching, it was concluded that the **create_connector** script not only creates connectors certificates and keys but also creates and moves the verification key to a specific folder. The lack of a verification key in a specific location was the reason for this error.

As this problem comes from the DAPS implementation, not the interaction with the other IDS components, the partner responsible for implementing the DAPS was warned of this behaviour. Meanwhile, the script was altered to include the subject name since it is necessary for interacting with the Broker, Clearing House and App Store.

5.7.2.5 Challenge 5: Unable to login on DAPS UI

One of the reasons to use a docker-compose was to be able to use the UI. However, it is impossible to log in despite the UI username and password being defined on the environment variables file.

As the user interface is unnecessary since everything can be done on the command line, this error was registered, and the implementation continued.

5.7.2.6 Challenge 6: No key was found in keystore for given alias

This problem was generated when the Clearing House was restarted using the newly created certificates.

When the trusted connector on the docker-compose was updated to use the new certificate instead of the older one, the error **No key was found in keystore for given alias** was thrown.

It was found that the problem was the file alias name. The CH code made it mandatory for the certificate alias to be "1". After the change, this problem seizes to exist. Figure 5.36 shows the certificate.

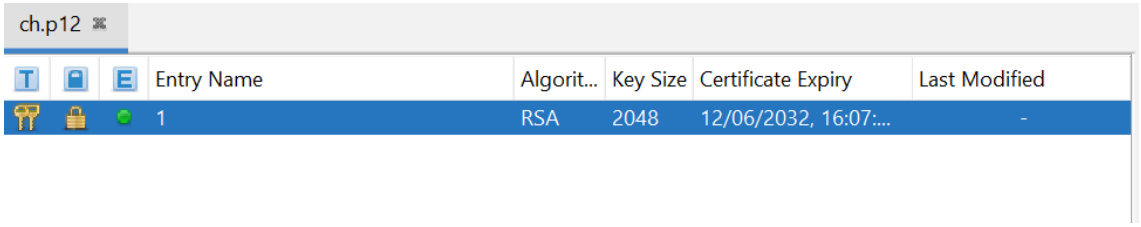


Figure 5.36: Clearing House certificate

5.7.2.7 Challenge 7: DAPS token route

This challenge was caused by the route used by the *Logging System* to retrieve the DAPS token.

This route is hard-coded to the one used by the Fraunhofer-hosted DAPS. Nevertheless, our DAPS uses a different one.

A search was done to find the place where the route is defined. The constant found was overridden on the services configuration files *Rocker.toml*. Regardless, this did not change the route.

Next, the variable was instantiated with the updated DAPS endpoint and the docker-compose restarted. This also did not fix the issue.

Lastly, the DAPS proxy was updated to redirect the route */.well-known/jwks.json* to the *auth/jwks.json* . Not only did this fix this issue, but it also fixed the **Authentication error** in Experiment 6: Connector-Broker-DAPS Interaction.

5.7.2.8 Challenge 8: Error retrieving DAT

This error appears when connecting the CH and the DAPS. Figure 5.37 shows the DSC logs.

The error comes from the *tc-core* container. Moreover, two lines were added to the routes file to include the stack trace, and then the containers were restarted to have more information regarding it. The logs are shown in figure 5.38. There, it is possible to see the error comes when the container tries to retrieve the DAPS token.

It was discovered that the trusted connector has problems connecting to a DAPS with a port. So, the DAPS was changed to production mode, Experiment 8: DAPS in production mode, since the port 80 was closed on the virtual machine, but the 443 was open.

After updating the DAPS environment, the docker-compose and *Rocket.toml* files were updated to use *https://vcese19.inesctec.pt* instead of *http://vcese19.inesctec.pt:4567*.

Removing the ports from the DAPS instantiation on the trusted connector solves this problem.

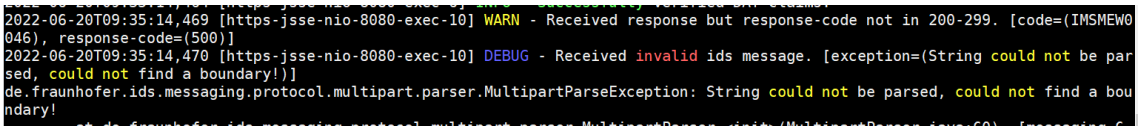


Figure 5.37: Subcontracted container logs

5.7.2.9 Challenge 9: Error regarding CA certificate

When using the production mode DAPS and the CH, the *logging-system* container throws the error shown in figure 5.39.

To see the certificate returned by the DAPS, it was run, inside the container, the command *openssl s_client -connect vcese19.inesctec.pt:443 -CAfile /etc/ssl/certs/daps_cachain.pem* . This is returning the incorrect certificate, figure 5.40.

It was discovered that this error is generated because the certificate sent to the Logging System, Keyring and Document API is incorrect. The CH documentation says it is the same as the one used by the trusted connector bind mounts and the *cert* folder on the bind mount. However, it should be the DAPS certificate itself (the one on DAPS bind mounts, which is the DAPS TLS certificate).

After updating the certificates on folder *ids-clearing-house-service/docker* to the one used on DAPS, the error was solved.

5.7.3 Experiments

Some experiments were developed to test the DAPS maturity and interaction with the other IDS components, e.g. Dataspace Connector, Metadata Broker and Clearing House. Those are described in detail in the following sections.

5.7.3.1 Experiment 1: Docker-compose setup

The IDSA already had a sample docker-compose with the environment variables defined on file *.env*. This docker-compose comprises a reverse proxy, DAPS server and DAPS user interface.

The first step in implementing this environment was to update the environment variables file. This included the certificate path, administrator credentials and virtual machine domain. It also involved updating the file since it was not compatible with the docker version on the virtual machine.

The test scripts the IDSA makes available were used to test if the values were correct. With the docker-compose, all the tests are successful, which was not true with the previous implementation.

This docker environment makes the DAPS only work with HTTP.

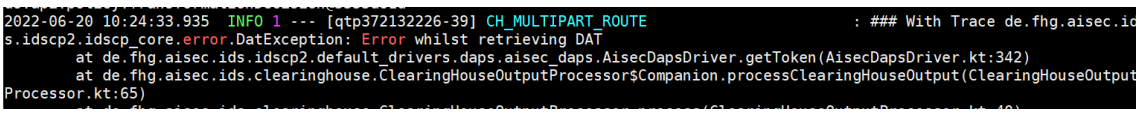


Figure 5.38: Trusted Connector logs

```
[ERROR] did not receive response from https://vcese19.inesctec.pt/.well-known/jwks.json:
Error(Hyper(Error(Connect, Custom { kind: Other, error: Ssl(Error { code: ErrorCode(1), cause:
    Some(Ssl(ErrorStack([Error { code: 337047686, library: "SSL routines", function:
    "tls_process_server_certificate", reason: "certificate verify failed",
    file: "../ssl/statem/statem_clnt.c", line: 1914 }]))) },
    X509VerifyResult { code: 20, error: "unable to get local issuer certificate" }) })),
    "https://vcese19.inesctec.pt/.well-known/jwks.json")
[2022-06-22][10:37:05][core_lib::api::auth][ERROR] ... failed to get jwks from daps!
[2022-06-22][10:37:05][_][WARN] Request guard `ApiKey < IdsClaims, Empty >` is forwarding.
```

Figure 5.39: Logging Service Error

```
CONNECTED(00000003)
depth=0 C = PT, ST = Porto, O = "INESC TEC - Instituto de Eng de Sist e Comp, Tecn e Ci\C3\AAncia", CN = vcese19.inesctec.pt
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = PT, ST = Porto, O = "INESC TEC - Instituto de Eng de Sist e Comp, Tecn e Ci\C3\AAncia", CN = vcese19.inesctec.pt
verify error:num=21:unable to verify the first certificate
verify return:1
depth=0 C = PT, ST = Porto, O = "INESC TEC - Instituto de Eng de Sist e Comp, Tecn e Ci\C3\AAncia", CN = vcese19.inesctec.pt
verify return:1
---
Certificate chain
 0 s:C = PT, ST = Porto, O = "INESC TEC - Instituto de Eng de Sist e Comp, Tecn e Ci\C3\AAncia", CN = vcese19.inesctec.pt
   i:C = NL, O = GEANT Vereniging, CN = GEANT OV RSA CA 4
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIHdjCCBV6gAwIBAgIQRPkXUO3oFlcr04PCiIW+5DANBgkqhkiG9w0BAQwFADBE
MQswCQYDVQQGEwJOTDEZMBcGA1UEChMQR0VBTlQgVmVyZW5pZ2luZzEaMBgGA1UE
AxMRR0VBTlQgT1YgUlNBIENBIDQwHhcNMjIwNTA0MDAwMDAwWhcNMjMwNTA0MjM1
OTU5WjCBgjELMAkGA1UEBhMCUFQxDjAMBgNVBAgTBVBvcnRvMUUwQwYDVQQKDDxJ
TkVTQyBURUMgLSBJbnN0aXR1dG8gZGUgRW5nIGRlIFNpc3QgZSBDb21wLCBUZWNu
IGUgQ2nDqm5jaWExHDAaBgNVBAMTE3ZjZXNlMTkuaW5lc2N0ZWMucHQwggEiMA0G
CSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC8KlZJh4iuli0ArS81E6io9B1aj1O3
2qXj8kISRCQS3SPK+e61rmBoGkribwxOzFkQSDy7WxwIFXtWMGBBFLz6sCSHXIEP
BaX5c40IrF+ZyB5zTscPwSp3wpzpuejK1/5qKznH/qQqMT4VKwdVsylTlPDRpN01
KYwcAppxo2lMQGnlwVwsbydWC9S5RvVqRRtgDO3vIn5+v1FcoMWAb7xkXccfTzrK
0wqwOMqDPI7cfTMTJyWV9yH5vFvasrujatQzD7qqcyp8MCYq1X3cjQZM+6HZZ3FJ
```

Figure 5.40: Returned Certificate

5.7.3.2 Experiment 2: Generate and test certificates

As the Fraunhofer-hosted DAPS creates certificates and sends them to the connectors, this experiment was created to replicate that behaviour.

First, the script provided to register connectors was run. Not only does it creates certificates and keys, but it also registers the clients' ID on the DAPS server.

Then, the test script was run. It reads the key and certificate, figures out the token endpoint, and sends a request just as a connector would. Figure 5.41 shows the script response using the producer certificate.

```
cese@vcese19:~/daps_v2/omejdn-daps$ scripts/test.sh conn_producer
Trying to get a DAT for conn_producer from http://vcese19.inesctec.pt:4567/auth
Derived connector ID: 2F:C1:27:26:83:35:CC:A2:B5:73:FB:9E:89:0F:27:0C:68:4F:CF:74:keyid:2F:C1:27:26:83:35:CC:A2:B5:73:FB:9E:89:
0F:27:0C:68:4F:CF:74
Aquiring the server's metadata document from http://vcese19.inesctec.pt:4567/.well-known/oauth-authorization-server/auth
The token endpoint is http://vcese19.inesctec.pt:4567/auth/token
Requesting a DAT from the above token endpoint
Here is the DAT Header:
{
  "typ": "at+jwt",
  "kid": "60beff1ad662e38fd8996639286e3c24e2b366e52f87d88251b854096ef78c39",
  "alg": "RS256"
}
Here is the DAT Body:
{
  "scope": "idsc:IDS_CONNECTOR_ATTRIBUTES_ALL",
  "aud": [
    "idsc:IDS_CONNECTORS_ALL"
  ],
  "iss": "http://vcese19.inesctec.pt:4567/auth",
  "sub": "2F:C1:27:26:83:35:CC:A2:B5:73:FB:9E:89:0F:27:0C:68:4F:CF:74:keyid:2F:C1:27:26:83:35:CC:A2:B5:73:FB:9E:89:0F:27:0C:68:
4F:CF:74",
  "nbf": 1654104376,
  "iat": 1654104376,
  "jti": "842b5ff2-8bcc-4ad4-b914-de4301aee4aa",
  "exp": 1654107976,
  "client_id": "2F:C1:27:26:83:35:CC:A2:B5:73:FB:9E:89:0F:27:0C:68:4F:CF:74:keyid:2F:C1:27:26:83:35:CC:A2:B5:73:FB:9E:89:0F:27:
0C:68:4F:CF:74",
  "securityProfile": "idsc:BASE_SECURITY_PROFILE",
  "referringConnector": "http://conn_producer.demo",
  "@type": "ids:DatPayload",
  "@context": "https://w3id.org/idsa/contexts/context.jsonld",
  "transportCertsSha256": "83e759fe21299923913cf33f9e6917cc14dd2e3fffe111c5ae3cda78469906b8"
}
```

Figure 5.41: DAPS test script response

Given that the test script ran without errors (which was not happening without the docker-compose, leading to believe the original configuration was incorrect), this experiment was successful.

5.7.3.3 Experiment 3: Connector-DAPS Interaction

This experiment is divided into two parts. First, the certificates were inserted into the connectors. Second, the docker-compose was updated to redirect to the new DAPS. It aims to simulate the original DAPS behaviour by sending the certificates and keys to the client.

First, the files generated are sent to the connector virtual machine to simulate the Fraunhofer DAPS behaviour. There, they were converted into *.p12* format with a password using the OpenSSL command:

openssl pkcs12 -export -out <NAME>.p12 -inkey <KEY_NAME>.key -in <CERTIFICATE_NAME>.cert -passout pass:<PASSWORD>

Next, the connectors docker-compose was updated to redirect to the DAPS implemented and also to update the certificates. The Connector *config.json* file was also updated to use the *.p12* file created. Listing 5.24 shows the docker-compose changes to redirect to the new DAPS.

```
1     - 'DAPS_URL=http://vcese19.inesctec.pt:4567'
2     - 'DAPS_TOKEN_URL=http://vcese19.inesctec.pt:4567/auth/token'
3     - 'DAPS_KEY_URL=http://vcese19.inesctec.pt:4567/auth/jwks.json'
4     - 'DAPS_INCOMING_DAT_DEFAULT_WELLKNOWN=/jwks.json'
```

Listing 5.24: DAPS Configuration on Connector

Challenge 2: Client unknown and Challenge 3: Could not find KeyStore at system scope explain the issues this experiment arose. However, this experiment was successful after solving them and represents the DAPS-Connector Interaction.

It is important to note that the process of changing the certificates was made in the producer and the subcontracted connectors since, as they communicate, they need to use the same DAPS.

5.7.3.4 Experiment 4: Create certificates without the script

The script to register connectors is optional since it is possible to create the certificate and key without it.

This experiment tests that, given that it does not allow defining a subject name, which is vital for components like the Metadata Broker and App Store that validate the machine name.

First, the certificate and key were generated using an OpenSSL command. Then, they were sent to the connector, and the necessary step to use them were made.

When the connector tried to access the token endpoint, the message **No verification key available** was thrown. Challenge 5.7.2.4 describes the reason for the error. Essentially, it means the DAPS could not find the verification key, which is automatically created when running the create certificate script. The solution was updating the script to add the subject name and the issuer.

5.7.3.5 Experiment 5: Use different DAPS on the connectors

This experiment was developed to guarantee that the connectors could not communicate using different DAPS.

Only the producer connector was updated to use the local DAPS for this case. When communicating with the subcontracted connector, the logs read:

```
1 Requesting public key of token issuer. [url=(https://daps.aisec.fraunhofer.de/jwks.
      json), kid=(TCUFeCNazalKHg9KzrzLIAzRWTMDDWSa7LcM9ZwHMyo)]
2 2022-06-06T10:38:28,354 [https-jsse-nio-8080-exec-10] ERROR - Exception while
      requesting public key from token issuer! [code=(IMSCOE0148), message=[Parsing
      error: org.jose4j.json.internal.json_simple.parser.ParseException: Unexpected
      character (<) at position 0.]]
```

```
3 2022-06-06T10:38:28,361 [https-jsse-nio-8080-exec-10] ERROR - An unhandled
    exception has been caught. [exception=(publicKey is marked non-null but is null
    )]
4 java.lang.NullPointerException: publicKey is marked non-null but is null
```

As expected, this experiment failed.

5.7.3.6 Experiment 6: Connector-Broker-DAPS Interaction

This experiment was developed to test the communication between the connectors and the Metadata Broker, as well as to find the steps to update the docker-compose to redirect to the local DAPS.

First, the certificates were created with the subject name and sent to the metadata broker machine.

Next, the broker docker-compose was updated to redirect to the new DAPS. Figure 5.42 shows the *broker-core* container of the docker-compose with the DAPS alteration. Then the certificates were converted to the correct format and inserted into the proper directories. Next, the repository *application.properties* file is updated to redirect to the new DAPS. Finally, the dockerfile that inserts the new, untrusted certificates into the system folder is run and then the docker-compose is built.

The detailed steps are described in section Testing procedures.

To test the Broker and DAPS communication, a request by the connectors needs to happen. For this purpose, the description endpoint on the producer connector was used. There, `https://vcese19.inesctec.pt:443/infrastructure` was used as the recipient ID.

This request generated the error message **PKIX path building failed: unable to find valid certification path to requested target**. After searching, it was discovered that this error was generated by the DAPS certificates not being trusted. Despite not being a permanent solution, it was solved by sending the broker's public key to the connector to insert into its *Keystore*.

Again, the description endpoint was run to see if now the communication was working.

This generated an authentication error on the Dataspace Connector logs. While resolving Challenge 7: DAPS token route, this error was fixed, being caused by the DAPS proxy lacking a redirect to the correct token endpoint.

The response is shown is figure 5.43, where the broker's self-description is seen, proving the successful communication between the three IDS components

5.7.3.7 Experiment 7: Connector-Clearing House-DAPS Interaction

This experiment was developed to test the communication between the connectors, the Clearing House and the DAPS, as well as to find the steps for the CH to use the new DAPS.

First, as always, the certificates were created on the DAPS and then sent to the CH machine. There, the certificates were converted into *p12*, *crt* and *der* format and then inserted on the correct directories.

```
broker-core:
    container_name: broker-core
    build:
      context: ../broker-core
      dockerfile: Dockerfile
    volumes:
      - ./cert-DAPS:/etc/cert/
    restart: always
    environment:
    - SPARQL_ENDPOINT=http://broker-fuseki:3030/connectorData
    - ELASTICSEARCH_HOSTNAME=broker-elasticsearch
    - SHACL_VALIDATION=true
    - DAPS_VALIDATE_INCOMING=true
    - IDENTITY_JAVAKEYSTORE=/etc/cert/isstbroker-keystore.jks
    - COMPONENT_URI=https://vcese19.inesctec.pt/
    - COMPONENT_CATALOGURI=https://vcese19.inesctec.pt/connectors/
    - DAPS_URL=http://vcese19.inesctec.pt:4567/auth/token
    - JWKS_TRUSTEDHOSTS=daps.aisec.fraunhofer.de,omejdn,vcese19.inesctec.pt
    - ssl.certificatePath=/etc/cert/server.crt
    - SSL.JAVAKEYSTORE=/etc/cert/isstbroker-keystore.jks
    expose:
    - "8080"
```

Figure 5.42: Broker docker-compose using new DAPS

200

Response body


```
{
  "@context": {
    "ids": "https://w3id.org/idsa/core/",
    "idsc": "https://w3id.org/idsa/code/"
  },
  "@type": "ids:Broker",
  "@id": "https://vcese19.inesctec.pt/",
  "ids:description": [
    {
      "@value": "A Broker with a graph persistence layer",
      "@language": "en"
    }
  ],
  "ids:title": [
    {
      "@value": "IDS Metadata Broker",
      "@language": "en"
    }
  ],
  "ids:hasDefaultEndpoint": {
    "@type": "ids:ConnectorEndpoint",
    "@id": "https://w3id.org/idsa/autogen/connectorEndpoint/0d4e70b1-d6a4-4fc2-bb94-28b8e2d45dae",
    "ids:path": "/",
    "ids:accessURL": {
      "@id": "https://vcese19.inesctec.pt/"
    },
    "ids:endpointInformation": [
      {
```


Figure 5.43: Successful communication between the Broker and Connector using the new DAPS

Next, the Clearing House docker-compose was updated to change the DAPS environment variable on the trusted connector container and replace the Fraunhofer certificate with the one from the local DAPS. This latter generated the error described in section Challenge 6: No key was found in keystore for given alias.

Next, the dockerfiles used to build the services *Document API*, *Keyring API* and *Logging Service* were updated to insert the new file into the container CA trusted certificates directory. This step is essential because a trusted CA does not create these certificates.

Finally, the configuration files used by *Document API*, *Keyring API* and *Logging Service* were altered to redirect to the new DAPS. These files are on the bind mounts.

Figures 5.44, 5.45 and 5.46 show the *Logging Service* final configuration file, the updated *Trusted Connector* container definition and the updated *Logging Service* dockerfile, respectively.

The detailed steps are described in section Testing procedures.

Next, a contract agreement process was started to test the communication between the Clearing House and DAPS, seeing as the CH only communicates with the DAPS when receiving messages.

This generated the error described in Challenge 7: DAPS token route. The solution found was to update the DAPS proxy to redirect to the correct URL.

After fixing this issue, another appears, this time on the *tc-core*. It says that the container cannot retrieve the DAPS token. This error is described in Challenge 8: Error retrieving DAT.

After fixing the previous issue, the CH and the contract agreement process were restarted. This time, the container that generated the error was the *logging service*. It says that it is unable to get a local issuer certificate. Challenge 9: Error regarding CA certificate explains better this challenge and the solution found. Essentially, the certificate the DAPS uses to authenticate itself needs to be inserted into the dockerfiles folder to replace the one created by DAPS.

After solving the last problem, the contract agreement process was restarted, and the DAPS, CH and DSC were able to communicate.

Essentially, to change the DAPS from the Fraunhofer to the one implemented on the virtual machine, the files that need updating are:

- All dockerfiles: to insert the new certificate into the trusted certificates folder of the operative system inside the docker container. This certificate needs to be the one used on DAPS docker-compose.
- docker-compose: update the *tc-core* to redirect to the local DAPS and use the new certificate.
- All Rocket: to redirect to the new DAPS.

5.7.3.8 Experiment 8: DAPS in production mode

This experiment was developed to put the DAPS in production mode. This means the DAPS uses HTTPS and TLS certificates.

```
[global]
limits = { json = 5242880 }
daps_api_url = "http://vcese19.inesctec.pt:4567"
connector_name = "https://clearinghouse.aisec.fraunhofer.de/"
infomodel_version = "4.0.0"
server_agent = "https://clearinghouse.aisec.fraunhofer.de"
signing_key = "keys/private_key.der"

[debug]
address = "0.0.0.0"
port = 8000
log_level = "normal"
limits = { forms = 32768 }
database_url = "mongodb://localhost:27019"
keyring_api_url = "http://localhost:8002"
document_api_url = "http://localhost:8001"
clear_db = true

[release]
address = "0.0.0.0"
port = 8000
log_level = "normal"
limits = { forms = 32768 }
database_url = "mongodb://logging-service-mongo:27017"
keyring_api_url = "http://keyring-api:8002"
document_api_url = "http://document-api:8001"
clear_db = false
```

Figure 5.44: *Logging System* configuration file

```yaml
tc-core:
    container_name: "tc-core"
    image: fraunhoferaisec/trusted-connector-core:6.3.0
    tty: true
    stdin_open: true
    volumes:
        - /var/run/docker.sock:/var/run/docker.sock
        - ./data/trusted-connector/allow-all-flows.pl:/root/deploy/allow-all-flows.pl
        - ./data/trusted-connector/conn_ch2.p12:/root/etc/keystore.p12
        - ./data/trusted-connector/truststore.p12:/root/etc/truststore.p12
        - ./data/trusted-connector/clearing-house-processors-0.8.0.jar:/root/jars/clearing-house-processors.jar
        - ./data/trusted-connector/routes/clearing-house-routes.xml:/root/deploy/clearing-house-routes.xml
    environment:
        TC_DAPS_URL: http://vcese19.inesctec.pt:4567
        TC_KEYSTORE_PW: password
        TC_TRUSTSTORE_PW: password
        TC_CH_ISSUER_CONNECTOR: https://w3id.org/idsa/inesctec/trusted-connector/clearing-house
        TC_CH_AGENT: https://w3id.org/idsa/inesctec/trusted-connector/clearing-house
        server.port: 8085
    ports:
        - "8085:8085"
        - "8443:9999"
```

Figure 5.45: *Trusted Connector* container

```dockerfile
FROM rust as builder
WORKDIR app
COPY LICENSE clearing-house-app ./
RUN cargo build --release

FROM debian:bullseye-slim

RUN apt-get update \
&& echo 'debconf debconf/frontend select Noninteractive' | debconf-set-selections \
&& apt-get --no-install-recommends install -y -q ca-certificates gnupg2 libssl1.1 libc6

# trust the DAPS certificate
COPY docker/ch.crt /usr/local/share/ca-certificates/daps_cachain.crt
RUN update-ca-certificates

RUN mkdir /server
WORKDIR /server

COPY --from=builder /app/target/release/logging-service .
COPY docker/entrypoint.sh .

ENTRYPOINT ["/server/entrypoint.sh"]
CMD ["/server/logging-service"]
```

Figure 5.46: *Logging System* Dockerfile

The IDSA provides a sample proxy configuration for a production environment. So, the line discovered on Challenge 7: DAPS token route and another were added to the file. Then, the certificates inserted into the nginx volumes were updated to be a *pem* trusted file.

Next, the docker-compose was updated to swap the ports. The HTTPS port became 443, and the HTTP became 80 (before was 4567). Using the default ports resolved Challenge 8: Error retrieving DAT.

The component using port 443 was the MB, which was altered to use 4567 instead.

Finally, the *.env* and *omejdn.yml* files were updated. Listings 5.25 and 5.26 show the files updated variables values.

```
1 OMEJDN_ENVIRONMENT=production
2 OMEJDN_PROTOCOL=https
3 OMEJDN_DOMAIN=vcese19.inesctec.pt
4 OMEJDN_ISSUER=https://vcese19.inesctec.pt/auth
```

Listing 5.25: DAPS production environment

```
1 issuer: https://vcese19.inesctec.pt/auth
2 front_url: https://vcese19.inesctec.pt/auth
```

Listing 5.26: DAPS production environment *omejdn.yml*

Then, the components using the DAPS need to be updated to reflect the changes.

On the connectors, the environment variables shown on listing 5.24 need to be updated to use *https://vcese19.inesctec.pt* instead of *http://vcese19.inesctec.pt:4567*. The connectors were restarted to see the result. This generated the error **PKIX path building failed: unable to find valid certification path to requested target**. This is the same error that happened in Experiment 6: Connector-Broker-DAPS Interaction, so the same fix was used: inserting the DAPS machine public key into the connector truststore. This process was repeated for both connector machines by inserting the other connector public key, the DAPS *pem* certificate and the *vcese19.inesctec.pt* DAPS created certificate. Next, the containers were restarted, and the communication between both connectors and the DAPS was working.

On the CH, the *Rocket.toml* and the docker-compose files were updated to use *https://vcese19.inesctec.pt* instead of *http://vcese19.inesctec.pt:4567*. The *vcese11.inesctec.pt*, *vcese12.inesctec.pt* and DAPS TSL certificates were also inserted into the trusted connector truststore.

On the Broker, the files *application.properties* and docker-compose were updated to redirect to the new DAPS address. The docker-compose was also updated to use port 4567 instead of the 443. Finally, the *daps.crt* file on *docker/broker-core* was replaced by the DAPS TLS certificate.

5.8 Discussion

After implementing and testing all the components described in this chapter, the conclusions are briefly described in this section.

Regarding the Clearing House, the Logging System is wholly implemented. Despite this service not being the only component that constitutes the CH, it is the only implementation available and the most important to guarantee data is not misused. The other services may be developed in the future, following the information available on the Clearing House GitHub page.

Analysing the implementation, testing and experiments described in this chapter, the CH runs without errors, and all bugs encountered were solved. From figures 5.10 and 5.11 is seen that each process has at least one contract agreement message and, possibly, artefact request and artefact response messages.

About the experiments, the only inconsistency was that the CH has endpoints to return a single log entry from a process. However, the Dataspace Connector does not yet support that functionality.

From the four messages the CH logs to the Dataspace Connector, three were tested and successful. The last one (i.e. data usage) was not tested since a policy needs to be specified for it to be logged. For this use case, that policy was not necessary.

The communication between the existing environment and the Clearing House is also working flawlessly, as proved by experiment 3 on page 62. It tests if turning the CH down impacted the connector plus App4EDI communication. That experiment was run every time the CH or the connectors and application were updated and was always successful.

Analysing the Metadata Broker challenges section, it is seen that the ones generated by the Broker implementation (i.e. challenge 5.5.3.1 and 5.5.3.4) are either because of a lack of documentation regarding its implementation or by the lack of documentation regarding the communication with the connectors. Experiments 2, 3, 4 and 5 were all necessary to truly solve the problem described on challenge 4 because of the lack of documentation.

Regarding its experiments, experiment 5 proves that the Rule is still not entirely ready for use with the Broker. However, the issues say that this problem is generated by the connector code, not the Broker. Experiments 6 and 7 show that the query part of the Broker is also working perfectly. The only downfall of these experiments is related to the Dataspace Connector not yet being able to work with the component ID. Experiments 12, 13 and 14 test the Connector-Broker persistence. They test if stopping the connector container, removing the connector volumes and removing the broker volumes, respectively, creates an inconsistency between the components. Of these experiments, only the last one failed, needing two extra steps to pass. However, the other experiments had also failed before and were corrected by the last Broker update. Finally, experiment 15 checks that the resources are not automatically inserted into a broker, guaranteeing that the control is still on the Data Providers' side.

Overall, the experiments show that the Broker is mature enough to start using in a production environment, knowing that some additional steps may need to be made to guarantee the persistence

if the Broker volumes are removed.

Regarding the App Store, it was simple to see how not mature it is, based on the challenges and the one experiment that could not even be completed.

Analysing the challenges, some emerged due to the incorrect or misleading documentation, as is the case with challenge 2. Most of them were because the App Store code itself was not ready to use, had some bugs, or the optional code was not working as intended. The fact that the components were configured to only work with *jar* file shows how far it is from being production-ready. Running with a *jar* is not sustainable in a production environment, not only because there is no proxy but also because the certificates are not validated with the file, and there are no volumes to guarantee persistence.

After having the App Store running with the docker-compose, its main problem is discovered on challenge 6, which is the inability to download an app. This experiment was repeated several times, and only in two was the application downloaded without throwing an error. This also does not mean that it works since it returned an empty response when querying the connector to list the apps downloaded. Meaning that even though the app was apparently downloaded, it does not appear on the connector.

The Identity Provider is comprised of a CA, ParIS and DAPS. The ParIS and the DAPS are the components to implement. However, the Dataspace Connector does not yet offer any interaction with the first, leading to the DAPS being the only IdP component to be implemented in this dissertation. Regarding this component, the implementation was being developed by someone else, and the objective was only to test its interaction with the other IDS components.

Analysing the DAPS experiments, it is possible to see that there is no documentation regarding its interaction with the components since the Fraunhofer-hosted DAPS was used by default. Because of that, most challenges surface when configuring the components to communicate with the DAPS and each other using the new DAPS server. Experiments 2, 3, 5, 6 and 7 show that the component works as intended.

To conclude, the more mature component is the Clearing House, followed by the DAPS, Broker, and finally, the App Store.

Chapter 6

Conclusion and Future Work

This chapter consists of a reflection on the objectives satisfaction and ends with a discussion of future work.

6.1 Objectives Satisfaction

The main objective of this dissertation was to create a self-hosted infrastructure based on the International Data Spaces by implementing the IDS main components to create a data space controlled by INESC TEC.

While implementing the components to create the infrastructure, all steps, experiments, challenges and solutions were registered.

Studying the usage control in the IDS context was also essential since it is the main responsible for guaranteeing data sovereignty. This was done in chapter 2, where a deep dive into the IDS usage control, usage control technologies, IDS contracts and policy classes were presented.

Analysing the Clearing House experiments from chapter 5 proves that this component is ready to use in a production environment.

Analysing the Metadata Broker experiments from chapter 5, section 5.5.4, sees from experience 5 that the Rule used on the offered resources registered on the broker cannot be *Connector Restricted*. Experiment 14 shows that removing the broker volumes currently leads to inconsistencies between the Dataspace Connector and the MB. Overall, the broker is mature enough to use in a production environment, always considering the above restrictions.

Analysing the App Store experiment and challenges from chapter 5, sections 5.6.3 and 5.6.4, this component is not ready to be used in a production environment.

Analysing the DAPS experiments from chapter 5, experiments 3, 5, 6, 7 and 8 prove that the component is mature.

To conclude, concerning the architecture of the infrastructure depicted in chapter 5, section 5.2, the only component that cannot be implemented is the App Store.

6.2 Future Work

For future work, the App Store will be retested and implemented when a new version is realised or the repository updated. Finally, the interactions between the ParIS and the Dataspace Connector will be tested, and experiments made to determine the first component maturity when the connector finishes its implementation.

Furthermore, it would be interesting to expand the infrastructure to contain multiple brokers, clearing houses, app stores and connectors to simulate a real scenario better. Multiple instances of these components are helpful since they would allow more specific experiments. Such as a connector communicating with different brokers depending on the information it wants (i.e. to obtain information about the value chain, a value chain broker is queried; or, to obtain information about travelling, the travel broker is queried). It would also allow testing the connectors' communication and problem resolution when using different clearing houses.

Implementing connectors from different vendors and developing more data apps would also be relevant to reflect a broad spectrum of scenarios. Implementing other IDS components, e.g. Trusted Connector or FIWARE TRUE Connector, would be important to test the communication between these components and others like the Clearing House or App Store. Moreover, other scenarios or use cases would be pertinent to implementing data apps with other usage policies and restrictions.

References

[1] International Data Spaces Association. Ids - a standard for data sovereignty and indispensable element of data ecosystems, 2019.

[2] Dr. Harrie Bastiaansen, Georg Bramm, Juan Ceballos, Mark Gall, and Maarten Kolenstart. Idsa white paper specification ids clearing house. 2020.

[3] Ewout Gort BSc. Developing a maturity model based approach supporting the decision to adopt international data spaces, 2021.

[4] Boris Düdder, Wolfgang Maaß, and Julian Schütte. Data ecosystems: Sovereign data exchange among organizations: Report from dagstuhl seminar 19391. *Dagstuhl Reports*, 9, 2019.

[5] Andreas Eitel, Christian Jung, Robin Brandstädter, Arghavan Hosseinzadeh, Sebastian Bader, CHristian Kühnle, Pascal Birnstill, Gerd Brost, Gall, Fabian Bruckner, Norbert Weißenberg, and Benjamin Korth. Usage control in the international data spaces (position paper). 3 2021.

[6] Idsa. International data spaces fact sheet and core statements, 2019.

[7] IDSA. Data spaces use cases, 2021.

[8] A. Munoz-Arcentales, S. López-Pernas, A. Pozo, A. Alonso, J. Salvachúa, and G. Huecas. Data usage and access control in industrial data spaces: Implementation using fiware. *Sustainability (Switzerland)*, 12, 2020.

[9] B. Otto, S. Lohmann, S. Auer, G. Brost, J. Cirullies, A. Eitel, and et al. Reference architecture model, idsa, 4 2019.

[10] Boris Otto, Michael ten Hompel, and Stefan Wrobel. International data spaces, 2019.

[11] Prof. Dr. Boris Otto. Gaia-x and ids. 1 2021.

[12] Sebastian Steinbuss. Idsa rule book. 12 2020.

[13] Sebastian Steinbuss, Matthijs Punter, Fabiana Fourrier, and Inna Skarbovski. Blockchain technology in ids position paper | version 1.0 |. 2019.

Appendix A

Clearing House

A.1 Docker-compose

Listing A.1 shows the final docker-compose.

```
version: '3'

services:
    keyring-mongo:
        container_name: "keyring-mongo"
        image: mongo:5.0.5
        restart: always

    document-mongo:
        container_name: "document-mongo"
        image: mongo:5.0.5
        restart: always

    logging-service-mongo:
        container_name: "logging-service-mongo"
        image: mongo:5.0.5
        restart: always

    keyring-api:
        container_name: "keyring-api"
        build:
            context: ../ids-clearing-house-service
            dockerfile: docker/keyring-api-multistage.Dockerfile
        depends_on:
            - keyring-mongo
        environment:
            # Allowed levels: Off, Error, Warn, Info, Debug, Trace
            - API_LOG_LEVEL=Info
            - RUST_BACKTRACE=full
        volumes:
```

```
31              - ./data/keyring-api/init_db:/server/init_db
32              - ./data/keyring-api/Rocket.toml:/server/Rocket.toml
33              - ./data/certs:/server/certs
34
35      document-api:
36          container_name: "document-api"
37          build:
38              context: ../ids-clearing-house-service
39              dockerfile: docker/document-api-multistage.Dockerfile
40          depends_on:
41              - keyring-api
42              - document-mongo
43          environment:
44              # Allowed levels: Off, Error, Warn, Info, Debug, Trace
45              - API_LOG_LEVEL=Info
46              - RUST_BACKTRACE=full
47          volumes:
48              - ./data/document-api/Rocket.toml:/server/Rocket.toml
49              - ./data/certs:/server/certs
50
51      tc-core:
52          container_name: "tc-core"
53          image: fraunhoferaisec/trusted-connector-core:6.3.0
54          tty: true
55          stdin_open: true
56          volumes:
57              - //var/run/docker.sock://var/run/docker.sock
58              - ./data/trusted-connector/allow-all-flows.pl:/root/deploy/allow-all-
                    flows.pl
59              - ./data/trusted-connector/vcese19.inesctec.pt.p12:/root/etc/keystore.
                    p12
60              - ./data/trusted-connector/truststore.p12:/root/etc/truststore.p12
61              - ./data/trusted-connector/clearing-house-processors-0.8.0.jar:/root/
                    jars/clearing-house-processors.jar
62              - ./data/trusted-connector/routes/clearing-house-routes.xml:/root/
                    deploy/clearing-house-routes.xml
63          environment:
64  #            TC_DAPS_URL: https://daps.aisec.fraunhofer.de
65              TC_DAPS_URL: https://vcese19.inesctec.pt
66              TC_KEYSTORE_PW: password
67              TC_TRUSTSTORE_PW: password
68              TC_CH_ISSUER_CONNECTOR: https://w3id.org/idsa/inesctec/trusted-
                    connector/clearing-house
69              TC_CH_AGENT: https://w3id.org/idsa/inesctec/trusted-connector/clearing-
                    house
70              server.port: 8085
71          ports:
72              - "8085:8085"
73              - "8443:9999"
```

```
74
75    logging-service:
76        container_name: "logging-service"
77        build:
78            context: ../ids-clearing-house-service
79            dockerfile: docker/logging-service-multistage.Dockerfile
80        depends_on:
81            - document-api
82            - keyring-api
83            - logging-service-mongo
84        environment:
85            # Allowed levels: Off, Error, Warn, Info, Debug, Trace
86            - API_LOG_LEVEL=Debug
87            - RUST_BACKTRACE=full
88        volumes:
89            - ./data/Rocket.toml:/server/Rocket.toml
90            - ./data/keys:/server/keys
91            - ./data/certs:/server/certs
```

Listing A.1: Clearing House docker-compose

A.2 Bind mounts information

- Container Keyring API:
  - For the container to work, its database must contain the acceptable document types. So, an initial document type, located on *init_db/default_doc_type.json*, is used to populate the database.
  - Keyring API configuration file, *Rocket.toml*.
  - DAPS certificate.
- Container Document API:
  - Document API configuration file, *Rocket.toml*.
  - DAPS certificate.
- Container Logging Service:
  - Logging Service configuration file, *Rocket.toml*.
  - DAPS certificate.
  - The private key used for signing the receipts.
- Trusted Connector:
  - Keystore file, which is an IDS-issued certificate for the virtual machine.

- Truststore certificate.
- A usage control policy that allows everything.
- *Jar* file built from the Clearing House Processors.
- A file containing the camel routes required by the Clearing House.

Appendix B

Metadata Broker

B.1 Docker-compose

Below, the final docker-compose is shown:

```
version: '3'
services:
  broker-reverseproxy:
    container_name: broker-reverseproxy
    image: registry.gitlab.cc-asp.fraunhofer.de/eis-ids/broker-open/reverseproxy
    volumes:
     - ./cert-DAPS:/etc/cert/
    ports:
    - "4567:443" # IDS-HTTP API
    - "801:80"


  broker-core:
    #image: registry.gitlab.cc-asp.fraunhofer.de/eis-ids/broker-open/core:latest
    container_name: broker-core
    build:
      context: ../broker-core
      dockerfile: Dockerfile
    volumes:
      - ./cert-DAPS:/etc/cert/
    restart: always
    environment:
    - SPARQL_ENDPOINT=http://broker-fuseki:3030/connectorData
    - ELASTICSEARCH_HOSTNAME=broker-elasticsearch
    - SHACL_VALIDATION=true
    - DAPS_VALIDATE_INCOMING=true
    - IDENTITY_JAVAKEYSTORE=/etc/cert/isstbroker-keystore.jks
    - COMPONENT_URI=https://vcese19.inesctec.pt:4567/
    - COMPONENT_CATALOGURI=https://vcese19.inesctec.pt:4567/connectors/
#    - DAPS_URL=https://daps.aisec.fraunhofer.de/v2/token
```

```
31      - DAPS_URL=https://vcese19.inesctec.pt/auth/token
32      - JWKS_TRUSTEDHOSTS=daps.aisec.fraunhofer.de,omejdn,vcese19.inesctec.pt,
            daps_nginx
33      - ssl.certificatePath=/etc/cert/server.crt
34      - SSL.JAVAKEYSTORE=/etc/cert/isstbroker-keystore.jks
35      expose:
36      - "8080"
37
38    broker-fuseki:
39      container_name: broker-fuseki
40      image: registry.gitlab.cc-asp.fraunhofer.de/eis-ids/broker-open/fuseki
41      expose:
42      - "3030"
43      volumes:
44        - broker-fuseki:/fuseki
45
46  volumes:
47      broker-fuseki:
```

Listing B.1: Metadata Broker docker-compose

Appendix C

App Store

C.1 Docker-compose

```
version: '3.5'

services:
  haproxy: # https://hub.docker.com/_/haproxy?tab=description
    build:
      context: .
      dockerfile: haproxy-Dockerfile
    ports:
      - '9443:9443'   # forwarded to appstore
    networks:
      - proxynet
    depends_on:
      - appstore
      - postgres
    restart: always

  postgres:
    image: postgres:13
    container_name: appstore-postgres
    expose:
      - 5432
    env_file:
      - postgres.env
    networks:
      - proxynet
    volumes:
      - postgres-volume:/var/lib/postgresql/data

  appstore:
    container_name: appstore
    hostname: appstore
```

```
#    image: anacarolina1234/app-store:v4
    build:
      context: ..
      dockerfile: appstore.Dockerfile
    expose:
      - 9443
    restart: "always"
    env_file:
      - connector.env
    volumes:
      - search-data:/data/search
      - ./conf:/tmp/conf
    networks:
      - proxynet
    depends_on:
      - postgres

networks:
  proxynet:
    name: reverseproxy_network

volumes:
  postgres-volume:
    name: postgres-volume
  search-data: { }
```

Listing C.1: App Store docker-compose

C.2 Environment variables file

```
SPRING_DATASOURCE_URL=jdbc:postgresql://postgres:5432/connectordb
SPRING_DATASOURCE_USERNAME=idsappstore
SPRING_DATASOURCE_PASSWORD=password1234
SPRING_DATASOURCE_PLATFORM=postgres
SPRING_DATASOURCE_DRIVER-CLASS-NAME=org.postgresql.Driver
SPRING_JPA_DATABASE-PLATFORM=org.hibernate.dialect.PostgreSQLDialect

application.http.base-url=https://vcese19.inesctec.pt:9443
server.port=9443

registry.host=vcese19.inesctec.pt:4432
registry.url=https://vcese19.inesctec.pt:4432
registry.project=library
registry.api.url=https://vcese19.inesctec.pt:4432/api/v2.0
registry.client.user=admin
registry.client.password=Harbor12345
```

```
17
18 server.ssl.enabled=true
19
20 springdoc.swagger-ui.path=/api/docs
21
22 ## CONF
23 spring.security.user.name=admin
24 spring.security.user.password=stv5008
25
26 spring.security.app.name=app
27 spring.security.app.password=stv5008
28
29 configuration.path=conf/config.json
30 configuration.keyStorePassword=password
31 configuration.trustStorePassword=password
32
33 server.ssl.key-store=file:///tmp/conf/vcese19.inesctec.pt.p12
34 server.ssl.key-store-password=password
35 idscp2.keystore=./conf/vcese19.inesctec.pt.p12
```

Listing C.2: App Store Environment Variables

Appendix D

Portainer

When running the Portainer container already instantiated, some problems were noticed. First, when running *docker-compose up -d –build*, the proxy container did not initialise. Second, the Portainer Agent, with the finality of configuring volumes from the user interface, was also not working properly.

D.1 Challenge 1: User not created when building containers

The first error was only generated when running *docker-compose up -d –build*. If using *docker-compose up -d* everything worked fine.

As this was on another project, a quick fix was being used. It consisted of removing the *portainer* folder (automatically created when running the docker-compose), followed by running the *docker-compose up -d –build* and then creating the user on the user interface. This error occurred because the *–build* was not recreating the user.

To fix that behaviour, a portainer environment variable was used to define the administrator password. When defining this user's password, the user is automatically created, which was not happening before. However, before using the variable was first necessary to hash the password using the command:

htpasswd -nb -B admin "your-password" | cut -d ":" -f

On the resulted hash was then duplicated the *$* symbol. Listing D.1 shows the portainer container configuration with the resulting environment variables. After the alterations, the docker-compose was started, and a new error arose that said: **error checking context: 'can't stat '/home-/cese/portainer/bin"**.

The new error was because the folder *portainer/bin* needed root permissions. So, to fix this, a *.dockerignore* file was created to ignore the */portainer* folder, which is automatically created when running the build.

```
1 portainer:
2     image: portainer/portainer-ce:2.11.1
```

```
    volumes:
        - ./portainer:/data
        - /var/run/docker.sock:/var/run/docker.sock
        - ./portainer-certs:/certs
    command:
        - --sslcert
        - /certs/vcese11_inesctec_pt.cer
        - --sslkey
        - /certs/vcese11_inesctec_pt.key
        - --admin-password
        - '$$2y$$05$$jGYoBJr1xvm0htgpfg7T67GLNKtAlBEcVIkRz/xkKzPEd6FHwwGG'
    restart: always
```

Listing D.1: Portainer service excerpt

D.2 Challenge 2: Portainer Agent configuration

Regarding the Portainer Agent, it was necessary to do the following steps:

1. On the proxy container configuration file, configure the portainer agent;
2. On the proxy container from the docker-compose, add the port to forward to the portainer agent and the portainer agent dependency;
3. On the portainer agent container, add the portainer dependency;
4. Run *docker-compose up -d –build*
5. On Portainer UI, select **Environment** from the left menu and **Add Environment**. Figure D.1 shows the values for configuring the agent.

In this environment, the reason for using the portainer agent is to be able to configure the container volumes on the portainer in run-time. The other option is to turn the environment down and update on the docker-compose.

D.3 Docker-compose

```
version: '3.7'

services:

    haproxy:    # https://hub.docker.com/_/haproxy?tab=description
        build:
            context: .
            dockerfile: haproxy-Dockerfile
```

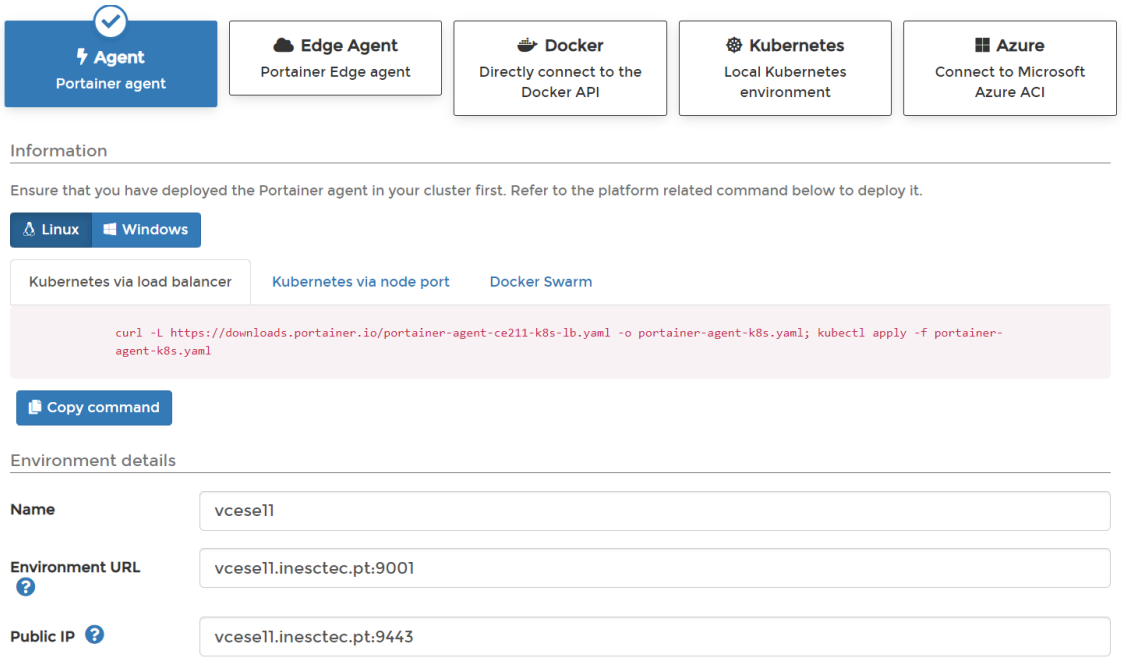


Figure D.1: Portainer Agent Configuration

```
9         ports:
10            - '9443:9443'    # forwarded to portainer
11            - '9001:9001'    # forwarded to portainer_agent
12        networks:
13            - portainer
14        depends_on:
15            - portainer
16            - portainer-agent
17        restart: always
18
19    portainer:
20        image: portainer/portainer-ce:2.11.1
21        container_name: "portainer"
22        networks:
23            - portainer
24        volumes:
25            - ./portainer:/data
26            - /var/run/docker.sock:/var/run/docker.sock
27            - ./portainer-certs:/certs
28        command:
29            - --sslcert
30            - /certs/vcese11_inesctec_pt.cer
31            - --sslkey
32            - /certs/vcese11_inesctec_pt.key
33            - --admin-password
34            - '$$2y$$05$$jGYoBJr1xvm0he3IRgxRDu4GLNKtAlBEcVIkRz/xkKzPEd6FHwwGG'
35        restart: always
```

```
36
37      portainer-agent:
38          image: portainer/agent:2.11.0
39          container_name: "portainer-agent"
40          networks:
41              - portainer
42          volumes:
43              - /var/run/docker.sock:/var/run/docker.sock
44              - /var/lib/docker/volumes:/var/lib/docker/volumes
45          depends_on:
46              - portainer
47          restart: always
48
49  networks:
50      portainer:
```

Listing D.2: Portainer docker-compose

Appendix E

Harbor Configuration

The steps to configure the HTTPS access to Harbor and the Harbor *yml* file are described below.

E.1 Configure the HTTPS access to Harbor

The first step is creating a folder for the Harbor certificates and then copying both certificates into said folder:

```
1    cp vcese19.inesctec.pt.pem /data/cert/
2    cp vcese19.inesctec.pt.key /data/cert/
```

Then, the server certificate, key and CA files are copied into the Docker certificates folder on the Harbor host:

```
1    sudo cp vcese19.inesctec.pt.pem /etc/docker/certs.d/vcese19.inesctec.pt:4432/
2    sudo cp vcese19.inesctec.pt.key /etc/docker/certs.d/vcese19.inesctec.pt:4432/
3    sudo cp ca.crt /etc/docker/certs.d/vcese19.inesctec.pt:4432/
```

Finally, the CA certificates are updated, and the docker is restarted.

```
1    sudo cp ca.crt /usr/local/share/ca-certificates/ca.crt
2    sudo update-ca-certificates
3
4    systemctl restart docker
```

E.2 Configure *harbor.yml* file

The file *harbor.yml.temp* needs to be updated and renamed to *harbor.yml*. It is changed according to the desired VM hostname, ports, certificate and key location, and log level.

Figure E.1 shows an excerpt of the file with the updated values.

```
hostname: vcese19.inesctec.pt

# http related config
http:
  # port for http, default is 80. If https enabled, this port will redirect to https port
  port: 980

# https related config
https:
  # https port for harbor, default is 443
  port: 4432
  # The path of cert and key files for nginx
  certificate: /home/cese/certs/vcese19.inesctec.pt.pem
  private_key: /home/cese/certs/vcese19.inesctec.pt.key
```

Figure E.1: *harbor.yml* file

E.3 Deploy Harbor

When the files were ready, the scripts *./prepare* and *./instal.sh* were run with *sudo*. The *./prepare* enables HTTPS communication and the *./install.sh* creates a docker-compose based on configuration file *harbor.yml*.