TRUSTS
Trusted Secure Data Sharing Space

D2.7 Architecture design and technical specifications document II

Authors: Kim Fidomski, Ahmad Hemid, Benjamin Heitmann (FhG)

December 2021

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 871481.

TRUSTS Trusted Secure Data Sharing Space

D2.7 Architecture design and technical specifications document II

Document Summary Information

Grant Agreement No	871481	Acronym	TRUSTS
Full Title	TRUSTS Trusted Secure Data Sharing Space		
Start Date	01/01/2020	Duration	36 months
Project URL	https://trusts-data.eu/		
Deliverable	D2.7 Architecture design and technical specifications document II		
Work Package	WP2 - Requirements Elicitation & Specification		
Contractual due date	31/12/2021	Actual submission date	17/12/2021
Nature	Report	Dissemination Level	Public
Lead Beneficiary	Fraunhofer (FhG)		
Responsible Author	Benjamin Heitmann		
Contributions from	FhG, SWC, EMC, G1, NOVA, EBOS, LST, REL, FORTH, KNOW, RSA		

Revision history (including peer reviewing & quality control)

Version	Issue Date	% Complete	Changes	Contributor(s)
v0.1	12/10/2021	5	Initial Deliverable Structure	Benjamin Heitmann (FhG), Kim Fidomski (FhG), Ahmad Hemid (FhG)
v0.2	11/11/2021	15	Revision of the architecture requirements from WP3 tasks	Benjamin Heitmann (FhG), Steffen Biehs (FhG), Stefan Gindl (RSA), Victor Mireles Chavez (SWC), Nikos Fourlataras (REL), Dominik Kowald (KNOW), Ohad Arnon (EMC), Abdel Aziz Taha (RSA)
v0.3	18/11/2021	35	Revision of the components of the architecture	Benjamin Heitmann (FhG), Steffen Biehs (FhG), Stefan Gindl (RSA), Victor Mireles Chavez (SWC), Nikos Fourlataras (REL), Dominik Kowald (KNOW)
v0.4	24/11/2021	50	Update of the design considerations	Victor Mireles Chavez (SWC), Nikos Fourlataras (REL), Steffen Biehs (FhG)
v0.5	25/11/2021	75	Revision of the technical architecture of the TRUSTS platform	Victor Mireles Chavez (SWC), George Margetis (FORTH), Stefan Gindl (RSA)
v0.6	02/12/2021	95	Addition of introductory and concluding chapters	Kim Fidomski (FhG)
v1.0	06/12/2021	100	Deliverable ready for peer review	Kim Fidomski (FhG)
v1.1	10/12/2021	100	Peer review	Victor Mireles Chavez (SWC)
v1.2	13/12/2021	100	Revision according to peer review	Benjamin Heitmann (FhG), Kim Fidomski (FhG)
V1.3	16/12/2021	100	Peer review	Nikos Fourlataras (REL)
V1.4	16/12/2021	100	Revision according to peer review	Benjamin Heitmann (FhG), Kim Fidomski (FhG)
v1.5	17/12/2021	100	Final version	Benjamin Heitmann (FhG), Kim Fidomski (FhG)

Disclaimer

Copyright message

Table of Contents

List of Figures

List of Tables

Glossary of terms and abbreviations used

Abbreviation / Term	Description
ACME	Automated Certificate Management Environment
API(s)	Application Programming Interface
AR	Architectural Requirement
ARS	Architectural Requirement Summary
CKAN	Comprehensive Knowledge Archive Network
DAPS	Dynamic Attribute Provisioning System
DSC	Dataspace Connector
DM	Data Marketplace
DMA	Data Market Austria, see [1]
FR	Functional Requirement
GA	Grant Agreement
HTTP	Hypertext Transfer Protocol
ID	Identifier
IDS	International Data Spaces, see [2]
MVP	Minimum Viable Product, see [4,5]
PII	Personally Identifiable Information
UC	Use Case

Executive Summary

This is the second and final deliverable reflecting the status of the architecture design and the collection of technical specifications. The technical architecture of the TRUSTS platform is based on three pillars: (1) the vision for the architecture as a consensus of the technical experts in the project; (2) the functional requirements collected from experts inside and outside of the project as well as from surveys conducted; and (3) the architecture requirements collected from the project participants which are working on tasks related to the implementation of the platform.

Since the first version of this document, an iterative process has been conducted that has resulted in a revised technical architecture and in revised technical specifications. The experience and best practices of two previous initiatives, Data Market Austria (DMA) [1] and International Data Spaces (IDS) [2], for supporting data markets, served as a starting point for developing an architecture for the TRUSTS platform that can be refined iteratively. Accordingly, both initiatives have strongly influenced the architecture design of the TRUSTS platform. The resulting technical architecture follows the design principles of these initiatives and of Gaia-X [3]. As already described in D2.6, on a conceptual level, the roles of participants and the structuring of the federated architecture into several nodes are based on a hybrid of the related concepts from both DMA and IDS. On a technical level, the architecture design prescribes the reuse of infrastructure from both DMA and IDS for the implementation of the platform towards a true hybridized implementation. Taken together, the hybrid architecture will result in a platform implementation which aims to combine the best of both previous initiatives while mitigating strategically important weaknesses.

This deliverable is based on the previous version of this deliverable (D2.6). In particular, D2.7 is a refinement of D2.6. It includes the steps described below to specify the refined architecture design and technical specifications: First, the architectural requirements collected from the project participants which are working on tasks related to the implementation of the platform are revised by them according to their gained knowledge since deliverable D2.6. This is followed by a summary of the revised architectural requirements as well as the revised functional requirements (FRs). The functional requirements addressed are described in D2.3. Software components for the architecture are specified in order to address the collected functional and architectural requirements. These components form the technical architecture of the TRUSTS platform. Due to the iterative process, the technical architecture and the software components have slightly changed compared to the technical architecture and components introduced in D2.6. The updates made to the architecture are pointed out. In addition, the design considerations of the architecture, revised from D2.6, are described. The design considerations document the aspects of the architecture which are represented in the interplay of multiple components, instead of being implemented in a single component.

Of strategic importance are several high-level characteristics derived from Gaia-X, i.e., scalability, extensibility, and federation. By considering high-level characteristics derived from Gaia-X, we expect the architecture of the TRUSTS platform to be future-proof, as we expect Gaia-X to provide important impulses for the data economy in Europe.

Taken together, the strong foundation of the architecture of the TRUSTS platform achieved by reusing concepts and components developed in the DMA and IDS initiatives, and the strategic significance of the high-level characteristics to be realized in the TRUSTS platform, will enable the TRUSTS platform implementation to support new forms of innovation and the development of new business models.

1. Introduction

The architecture design of the TRUSTS platform represents the blueprint for the technical results of the TRUSTS project. As such, it also represents the foundation for instances of the TRUSTS platform which will be provided by one or more TRUSTS operators after the duration of the project. The technical specifications provide the details which are required by technical experts in order to instantiate the platform infrastructure and build on top of it or to extend it with their own components, services and applications.

This deliverable is the second and final document about the architecture design and technical specifications within the lifetime of the project. D2.6 'Architecture design and technical specifications document I' submitted in December 2020 (M12) and D2.7 'Architecture design and technical specifications document II' due for month 24, December 2021.

D2.7 is related to task 2.4. The work in this deliverable is the result of a collaborative and iterative process between all technical experts in the project. The architecture represents the conceptual foundation for the implementation of the TRUSTS platform, and therefore reaching a consensus on the architecture enables all project's technical partners to agree on the most important abstract decisions, before realizing them in their implementation. The architecture also allows the non-technical project partners to contribute cross-cutting requirements of strategic importance as well as future-proof architectural characteristics.

The efforts accomplished to design a technical architecture that addresses the requirements from different groups of stakeholders are summarized in this deliverable. It outlines the insights gained during the iterative process and summarizes the conclusions drawn in a technical architecture. As already discussed in D2.6, of particular importance are the documented decisions for challenges of strategic importance, which have an impact on the project beyond the purely technical aspects of the platform:

- The technical architecture specified in the TRUSTS project is to be designed as an extension of both the Data Market Austria [1] and Industrial Data Spaces [2] initiatives, which represented the state-of-the-art in supporting the development of data markets at the beginning of the project. **What conceptual and technical approaches should be reused in TRUSTS to best combine the existing experience and best practices of both initiatives** within the context of the project description?
- The technical architecture is to be future proof with regard to the emerging data economy in Europe. We expect Gaia-X [3] to give an important impulse for the data economy in Europe by e.g., communicating with important groups of stakeholders to set the agenda and by setting standards for technical and organizational issues such as certification. How can compatibility between the architecture being developed in TRUSTS, and the Gaia-X initiative be ensured? Which high-level characteristics derived from Gaia-X should be addressed by the platform?
- Why should anyone trade their data sets, applications, and services via the platform developed here? The platform must provide good reasons to attract users. Trust between the participants of the platform and in the platform itself is crucial. How can the technical infrastructure help to **enable trust between the participants of a data marketplace, in the platform itself**, and in a secure and private exchange of information? Given the fact that TRUSTS will employ novel approaches to enable this, how can this be anchored in the design of the architecture?
- How can the design of the architecture **enable new forms of assets to be traded and monetized** in a data marketplace? Is there a way to go beyond trading of data sets, and enable for instance the monetization of access to services and apps as part of a data marketplace?

1.1. Mapping Projects' Outputs

The purpose of this section is to map TRUSTS Grant Agreement commitments, both within the formal deliverable and task description, against the project's respective outputs and work performed.

Table 1: Adherence to TRUSTS GA deliverable & tasks descriptions

TRUSTS Task		Respective Document Chapter(s)	Justification
T2.4 Architecture design and technical specifications	*Based on the market analysis and requirements elicitation outcomes that are performed in T2.1 and 2.2, as well as the legal and ethical frameworks and requirements generated in T6.2, this task deals with the specification of an architectural design of the TRUSTS platform. While existing specifications such as the Reference Architecture Model (RAM) of the Industrial Data Space (published by the IDSA, co-edited with FhG) and design documents of the Data Market Austria, will serve as a basis, the additional contributions of this Tasks are to (i) align the collected requirements with these specifications, (ii) identify areas that need to be improved and adapted, and (iii) develop and suggest concrete requests for changes and adaptations of the RAM. The Task involves IDSA as well as key contributors to the DMA platform as the main stakeholders of the specifications, leveraging their established processes (e.g., working groups, technical advisory board meetings) to make sure that the proposed changes are incorporated efficiently with wide support from the industry. The task not only focuses on the architectural level but also on making decisions about technologies and methods to actually implement the architecture. This encompasses the design of data structures, message formats, APIs and protocols, but also the definition of procedures,*	Chapter 2 Chapter 3 Chapter 4 Chapter 5	Refinement of the architectural requirements based on gained knowledge. Requirements for the architecture are derived from: (1) IDS RAM and DMA as the foundation of the TRUSTS platform; (2) all technical experts in the project, who are working on tasks related to implementing the platform; (3) future proof compatibility with Gaia-X. The technical architecture and its components are described in comparison to D2.6. This includes the specification of technical details for the interplay of the components. Any additional design considerations and decisions for the technical architecture are described. The results of this deliverable are summarized, and the benefits of the technical architecture are highlighted.

	such as the deployment of new component instances and their integration into the running TRUSTS platform.		
TRUSTS Deliverable			
D2.7: Architecture design and technical specifications document II (FhG) [M24] R, PU *Second version of the periodically updated report on the TRUSTS platform specification that is based on the results communicated in D2.1 and D2.2 and D2.3. The document describes the architectural decisions taken and their rationale. Furthermore, D2.7 will in addition report on the application of the testing and benchmarking framework provided by T2.3.*			

1.2. Deliverable Overview and Report Structure

This deliverable summarizes the activities in task 2.4. "Architecture design and technical specifications". It serves as a blueprint for the technical results of the TRUSTS project and is primarily a refinement of deliverable D2.6.

The deliverable is structured into the following sections:

- **Technical requirements for the architecture:** This chapter collects the requirements for the architecture. A distinction is made between two types of requirements, collected by different groups of stakeholders. The first set of requirements are the functional requirements (FRs). This chapter refers to the updated set of functional requirements that is described in detail in deliverable D2.3 "Industry specific requirements analysis, definition of the vertical E2E data marketplace functionality and use cases definition II". The initial set of the functional requirements were collected mainly from the non-technical participants of the project. In an iterative process, the functional requirements were updated. New functional requirements were, i.e., collected from surveys. The second set of requirements that are introduced in more detail in this chapter are the architectural requirements (ARs). They were updated since the first version of this deliverable from the project's technical participants and are grouped by the different areas of concern for the TRUSTS architecture.
- **Technical architecture of the TRUSTS platform:** Based on the architectural requirements, a software architecture for the TRUSTS platform is described and compared to the architecture described in D2.6. An overview of the architecture is given. Furthermore, the software components that are needed in order to address the collected functional and architectural requirements are specified. Provided tables list which components address which identified functional and architectural requirements for the architecture.
- **Design considerations for the architecture of the TRUSTS platform:** The design considerations of the architecture are described to document the aspects of the architecture which are represented in the interplay of multiple components, instead of being implemented in a single component.
- **Conclusion:** The deliverable is concluded by listing the project results which are described in this deliverable. The benefits of the architecture are summarized.

1.3. Updates since the previous version of this deliverable

The previous version of this deliverable, D2.6, presented the basis of the architecture and technical specifications described in this deliverable. A technical architecture that gives the technical partners of the project an abstract, conceptual idea and provide details of how the TRUSTS platform should be implemented.

The development of the architecture was an iterative process. In parallel, the project's technical participants have started implementing the platform. The current status of the platform implementation is described in deliverable D3.10. The findings, which were gained through the implementation, have also been incorporated into the revised architecture presented here.

The purpose of this section is to summarize the updates that have been made to the architecture requirements described in the previous version of this deliverable, and thus providing a link between the two deliverables, both of which are related to task 2.4.

The architectural requirements (ARs) described in Section 2.1 have been slightly adjusted in D2.7. While the architectural requirements related to the well-known initiatives TRUSTS used as starting point have remained unchanged, architecture requirements identified by technical partners according to their tasks in the project have partially changed:

- **Modified architectural requirements**: AR 3.4.3, AR 3.4.5, AR 3.4.6, AR 3.4.8, AR 3.4.9, AR 3.6.1, AR 4.3.1
- **Added architectural requirements**: AR 3.4.13, AR 4.3.5

According to the feedback from the project's technical partners, the technical architecture was adapted. The technical architecture diagram (see Figure 8) has been updated, especially regarding the modified component list described in Section 3.2:

- **Removed / Replaced components**: Trusted Connector, Asset Consumer, Usage Control, Service Consumer Adapter, Identity Provider + Key Distribution System
- **Renamed components**: Dynamic Attribute Provisioning System (DAPS)

As other components cover the functionalities previously covered by the removed or replaced components, the tables indicating which components address which identified functional and architectural requirements for the architecture have been adjusted.

Interoperability of the platform is given more attention in the technical architecture diagram. Furthermore, the connections shown in the diagram have been updated and are explained in more detail in Section 3.1.3.

In Chapter 4, the design considerations are described according to the modified technical architecture.

2. Technical requirements for the architecture

The architecture of the TRUSTS platform must meet the requirements and priorities of many different stakeholders. In order to accomplish this, requirements have been collected from different groups of stakeholders. A distinction is made between two sets of requirements:

The first set of requirements are the functional requirements (FRs). They were initially collected mainly from the non-technical participants of the project. The functional requirements, after being modified according to the feedback and experiences of all project participants and adjusted based on surveys conducted, are described in D2.3 “Industry specific requirements analysis, definition of the vertical E2E data marketplace functionality and use cases definition II”. The method used to identify and modify the functional requirements is also described in D2.3. This chapter will only reference the functional requirements described in D2.3, no details about the FRs are given.

The second set of requirements are the architectural requirements (ARs). While functional requirements describe what the system should be able to do, architectural requirements describe which requirements the architecture must fulfill. As already described in D2.6, the architectural requirements were collected from the technical participants of the project and are grouped by the different areas of concern for the TRUSTS architecture. These areas of concern remained unchanged and are as follows:

- Architecture requirements from alignment with Data Market Austria components
- Architecture requirements from alignment with International Data Spaces components
- Architecture requirements from future alignment with Gaia-X
- Architecture requirements related to smart contracts
- Architecture requirements related to interoperability of data marketplaces
- Architecture requirements related to data governance
- Architecture requirements related to platform development and integration
- Architecture requirements related to brokerage and profiles for users and corporates
- Architecture requirements related to privacy enhancing technologies
- Architecture requirements related to anonymization and de-anonymization
- Architecture requirements following from the usage of Comprehensive Knowledge Archive Network software

Based on gained knowledge since the first version of this deliverable, the architectural requirements were revised by the project’s technical participants. In this section, the updated architectural requirements for the different areas of concern for the TRUSTS platform are listed first. This is followed by a summary of the architectural requirements.

2.1. Architectural Requirements

In addition to the functional requirements (FRs), the TRUSTS technical architecture is based on architectural requirements (ARs). The architectural requirements were initially collected and later revised by the project’s technical participants based on their experiences in the first half of the TRUSTS project. The ARs are grouped by the different areas of concern with regards to the architecture.

The first three sections remained unchanged and were taken from D2.6 to provide all the necessary information without switching between results. The ARs have been revised and confirmed by the respective technical experts working in the respective field.

For each area of concern, the requirements stemming from that area of concern are listed. For each requirement an ID, a summarizing title, and a description are provided.

2.1.1. Architecture requirements from alignment with Data Market Austria components

One of the two approaches for supporting data markets on which the TRUSTS platform builds, is the Data Market Austria (DMA) data market [1]. The DMA, which started from a flagship project founded by the Austrian Federal Ministry of Transportation, Innovation and Technology, was envisioned as a single point of entry to a federated network of data and data-service providers. It aimed at providing a user-friendly portal through which individuals interested in consuming datasets and data-services can see a list of available datasets, and then contact the respective provider to gain access to the infrastructure that hosts them. The DMA project finished in August 2019.

The DMA was envisioned as a distributed set of nodes hosting the different assets, and a central node hosting the metadata about the assets and other support services. Each node consists of a set of containers serving a series of core components, which include access control, metadata harvesting, asset ingestion UI and an Etherium blockchain node. The central node additionally executes several metadata management pipelines, along with a user authentication and access control service. Together, these services and an orchestrated exchange of information (e.g., ip-addresses and public keys for verifying authentication tokens) realize the federation of the DMA.

The DMA model of a distributed set of independent nodes and an additional central node has been inherited in the TRUSTS platform architecture presented in this document. It will be further enriched by the security-enhancing technologies that have been developed by the IDSA, as well as a series of components deployed ad hoc for the TRUSTS platform. Three important architectural commonalities exist between the two projects. First, the notion of independent nodes that host the assets offered by the organization. Second, the notion of a centralized metadata catalogue that, along with a set of controlled vocabularies, constitutes a knowledge graph on which applications such as search, and recommendation are powered. Third, the idea of a set of components being developed and distributed to different organizations so that each organization can set up their own participating node by running instances of the provided components.

Likewise, several outstanding differences can be identified between the two, which are introduced by the usage of software components from the International Data Spaces (IDS) [2]. The IDS will be described in the next section. On the one hand, the communication protocol between the different nodes is replaced in TRUSTS by the IDSCPv2 protocol, which incorporates an additional layer of security and trust by the use of cryptographic certificates and third-party attestation. On the other, TRUSTS introduces the notion of a portable application, which in turn necessitates a more adaptable routing mechanism within each node. In the DMA, routing is configured in a reverse proxy configuration which only requires alterations when a new core component is installed. This contrasts with the TRUSTS routing mechanism which allows dynamically for services to appear, disappear or change names within a node. Finally, the TRUSTS platform envisions a federated user information system, distributed across all the nodes.

Since some of the DMA components are repurposed in the TRUST platform, along with many of the design principles and user scenarios, the following architecture requirements are inherited by the TRUSTS architecture from the DMA project:

Requirements for reuse of DMA components and concepts	
AR 3.D.1	**Ability to operate as a distributed set of nodes.** The different providers and consumers of assets must remain capable of operating their own infrastructure in order to maintain data sovereignty. This infrastructure should be connected in a well-defined and easy to set-up manner in order to realize the business models.
AR 3.D.2	**Asset metadata distribution and aggregation.**

	Organizations should be able to offer existing assets on the platform. This requires mechanisms to ingest, map and distribute metadata about assets in a way that is searchable and actionable by the other participants of the platform.
AR 3.DMA.3	**Components must be easily deployable and connected.** The different nodes involved in the platform will each deploy a set of services. These can be developed or packaged by the platform operator, and they should be easily installable by every participating organization.
AR 3.DMA.4	**Node infrastructure metadata accessible to all components in a node.** All components running in a node must have access to a single and up-to-date source of basic metadata about the node, such as name, identification numbers, as well as metadata about other components it interacts with.
AR 3.DMA.5	**Single source of truth for controlled vocabularies.** In order to adequately manage metadata coming from different organizations, some of which might have been created with distinct purposes in hand, it is necessary to have a set of controlled vocabularies that are centrally maintained and which can be used in mapping of metadata schemas and items.

2.1.2. Architecture requirements from alignment with International Data Spaces components

The second approach for supporting data markets on which the TRUSTS platform builds, is the International Data Spaces (IDS) [2], which provides a set of infrastructure components on which data markets can be built. The IDS is a decentral software architecture for exchanging data in a sovereign, secure and interoperable way. Data owners' sovereignty over their data is achieved by certifying the actors that participate in a data space as well as the technical components they operate to exchange data, and by technically controlling the usage of data on the data consumer's side according to metadata by which the data owner described data usage policies.

The IDS Reference Architecture Model is defined by the IDS Association (IDSA) and its 100+ member organizations. In a minimal IDS, participants exchange data peer-to-peer. They do so by operating a standardized communication interface called Connector. A meaningful data space that allows for multiple participants that neither know nor trust each other initially and that is open for additional participants to join requires further infrastructure called *essential services*. For convenience, further non-essential services make sense in a data space. The following table lists them all:

Service	**What is it?**	**Essential?**
Certification body	Governance body empowered to grant IDSA certification for components and participants	Yes
Certification authority	Authority that is in charge of the certification to make sure that only compliant organizations are granted access to the trusted business ecosystem	Yes
Dynamic provisioning service	Management of certifications and metadata for all components and participants	Yes

Participant information system	Registry of certified participants that is accessible to all participants	Yes
Dynamic trust management	Governance body empowered to enforce basic security rules of IDS as a whole	Yes
IDS (metadata) brokers	IDS connectors will register descriptions of data endpoints with IDS brokers. This allows data consumers to find the data they need.	Yes
App store	Outlets provide data apps that can be deployed in IDS Connectors to execute tasks like transformation, aggregation or analytics on the data. Provided by IDS members, certified under IDS standards.	No
Vocabulary provider	Offer "vocabularies" such as ontologies, reference data models and metadata elements, which can be used to annotate and describe datasets.	No
Clearing houses	These intermediaries will provide clearing and settlement services for financial and data exchange transactions in the IDS.	No

We imagine the TRUSTS data space to be an IDS-compatible data space. At the end of the project, organizations should have been identified that take care of certification and trust management. Based on this, the following architecture requirements are inherited by the TRUSTS architecture from the IDS:

Requirements from future alignment with IDS	
AR 3.I.1	**Essential services (technical):** Services for dynamic provisioning, participant information and (metadata) brokerage shall be put into operation.
AR 3.I.2	**IDS Connectors:** All participants required for demonstrating the TRUSTS use cases shall be equipped with IDS Connectors and should be able to expose their data offerings through these connectors' interfaces, including self-describing metadata in terms of the IDS Information Model.
AR 3.I.3	**Essential services (operational):** Organizations that provide the essential services of certification (i.e., certification body and certification authority) shall be identified. A body in charge of dynamic trust management shall be established.

2.1.3. Architecture requirements from future alignment with Gaia-X

Gaia-X [3] is a project for the development of an efficient and competitive, secure and trustworthy federation of data infrastructure and service providers for Europe, which is supported by representatives of business, science and administration from European countries.

Gaia-X follows the principles of openness and transparency of standards, interoperability, federation, i.e., decentral distribution, as well as authenticity and trust (Technical Architecture [3, §1.2]).

These principles are translated into the following technical guidelines (Technical Architecture [3, §1.3]):

- Security-by-design
- Privacy-by-design
- Enabling federation, distribution, and decentralization
- Usage-friendliness and simplicity
- Machine-processability
- Semantic representation

The Gaia-X ecosystem as a whole is structured into a Data Ecosystem and the Infrastructure Ecosystem (Technical Architecture [3, §1.4]). The Data Ecosystem enables Data Spaces as envisioned by the European Data Strategy, where not only data is exchanged, but also advanced smart services are provided. The Infrastructure Ecosystem comprises building blocks from hardware nodes to application containers, where data is stored and services are executed, as well as networks, via which data is transmitted. Infrastructure itself may be provided as a service.

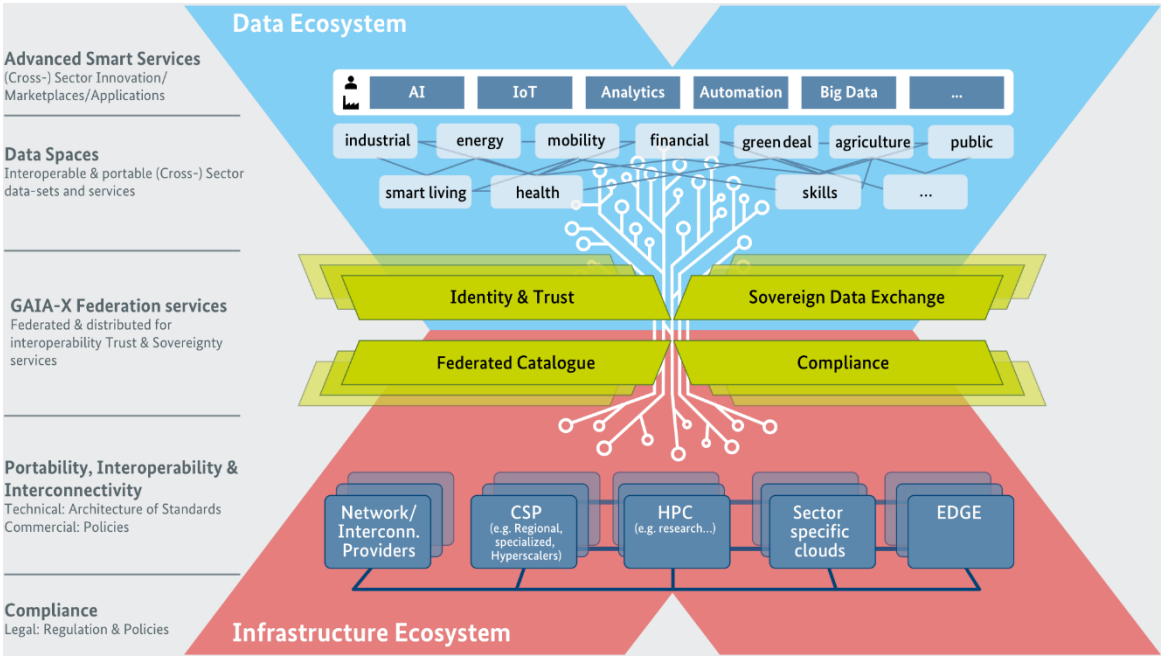


Figure 1: Gaia-X high-level architecture [3]

The Federation Services connect the Data and Infrastructure Ecosystems with concrete functionality that is in line with the architecture principles and technical guidelines. The Federation Services are grouped into the following domains, which are by no means specific to Gaia-X, but generally apply to data and service ecosystems based on cloud technology:

- Identity and trust mechanisms, comprising federated identity management, trust management, and federated access.

- A Federated Catalogue, comprising offerings of assets (i.e., data and services) that describe themselves in a FAIR (findable, accessible, interoperable, reusable) way.
- Sovereign Data Exchange, ensured by policies, usage control, and security concepts.
- Compliance in a sense of organization and governance (defining the rights and obligations between service providers and consumers, and the process of onboarding participants and their assets into the ecosystem), but also supported technically via certification and continuous monitoring

In being compatible with the Federation Services, providers and their assets can fulfil the architectural principles as follows:

Requirements from future alignment with Gaia-X	
AR 3.X.1	**Security by design:** Security considerations are addressed by secure and sovereign data exchange, as well as compliance concerns.
AR 3.X.2	**Privacy by design:** Identity and Trust provide the foundation for privacy. Compliance with privacy requirements is further ensured by the mechanisms for sovereign data exchange.
AR 3.X.3	**Enabling federation, distribution, and decentralization:** The Federation Services themselves are designed to operate in a federated, distributed, and decentralized manner. The Federated Catalogue encourages providers to offer services on all layers of abstraction, from infrastructure as a service to data as a service. Their machine-processable and semantic Self-descriptions enable automated deployment.
AR 3.X.4	**Usage-friendliness and simplicity:** APIs, which can be found via the Federated Catalogue), enable the construction of human user interfaces.
AR 3.X.5	**Machine-processability:** This is ensured by Self-descriptions of assets in the Federated Catalogue, which comprise usage policies and may point to APIs. APIs further enable automation.
AR 3.X.6	**Semantic representation:** Self-descriptions in the Federated Catalogue make assets FAIR (Findable, Accessible, Interoperable, Reusable).

2.1.4. Architecture requirements related to smart contracts

Smart contracts are one of the areas of concern for the TRUSTS architecture. The main goal is to develop the necessary concepts for using smart contracts in the context of the European Data Market delivered by the TRUSTS project. This includes the technical foundations for ensuring the integrity and authenticity of such contracts as well as the analysis of the legal challenges brought about by smart contracts, such as issues of validity, enforceability, and interpretation. Related to this, the technical challenges and the legal issues regarding the fact that smart contracts are written in executable code instead of natural language, will be examined by the partners with legal expertise in the project.

The following requirements for the architecture of the TRUSTS platform are specified:

Requirements related to smart contracts	
AR 3.2.1	**Smart contracts based on policies:** It should be possible to create smart contracts based on policies for data sharing / service usage. The policies define how the data / service can be handled by a data consumer. When a data consumer accepts a policy, a contract is created between the data provider and the data consumer based on the accepted policy. Based on this contract, a transaction will be created by the seller in this contract. The transaction will be sent to a blockchain instance. It contains the contract details, e.g., the seller, the buyer, and information about the data, so they can be recorded in a block within the blockchain for later checks. Before any further transaction is recorded into a new block regarding the above-mentioned contract, a smart contract is triggered which inspects the transaction and approves or denies it. The compliance with the policy can be proved at every time.
AR 3.2.2	**Logging of all transactions:** Every transaction between two participants should be logged. It should be possible to define which information is logged for which kind of transaction.

2.1.5. Architecture requirements related to the interoperability of data marketplaces

One of the concerns of the TRUSTS platform is the interoperability between TRUSTS and other data marketplaces. As described in D2.6, this includes the definition and implementation of interfaces to ensure interoperability with other industrial data marketplaces as well as with EOSC initiatives. In addition, this is related to analyzing and examining existing standards and interfaces with regards to their suitability for interoperability. This will result in the development of an interoperability solution, which provides the necessary technical functionality to interoperate with a selected set of existing data marketplaces and EOSC initiatives. The work already done is described in deliverable D3.5. The interoperability solution is envisaged as a client-server architecture consisting of a data exchange component residing on the premises of TRUSTS as well as another component residing on the premises of an external data marketplace or EOSC initiative, respectively.

The successful implementation of the interoperability solution requires a set of components. The architecture is based on a careful evaluation of both existing data marketplaces as well as EOSC initiatives. This involved the analysis of technical and operational features such as the provided resources (e.g., data assets or search functionality) or available APIs. It involves a metadata schema for data assets based on existing schemas such as the IDS Metadata Model, two components establishing metadata and/or data transfer between TRUSTS and external data marketplaces and EOSC initiatives, a registry of data marketplaces containing up-to-date information about external data marketplaces/EOSC initiatives, as well as a graphical user interface for the interoperability solution. The resulting architecture requirements remained unchanged and are as follows:

Requirements related to the interoperability of data marketplaces	
AR 3.3.1	**Metadata schema for data assets:** A documented metadata schema building on top of existing schemas, such as the IDS Metadata Model for describing resources and datasets but extended to data assets.
AR 3.3.2	**Data exchange client component:** A module that can be integrated into a common data management platform which will enable the exchange of information about data products across a federated data markets

	network. An external data market should be able to integrate this software library with ease and be able to expose information about its data products to TRUSTS as a result. The library should also serve as a basis for connecting TRUSTS with EOSC.
AR 3.3.3	**Data exchange TRUSTS component:** This component resides on the side of TRUSTS and communicates with the Data Exchange Client Component. It receives the input from the Data Exchange Client Component and converts it into the format required by TRUSTS, subsequently storing the received data in the TRUSTS data storage.
AR 3.3.4	**Registry of data markets:** This component will be essential for having up-to-date information about external data markets (metadata schemas, APIs, platforms) currently used in the sector. The information in the registry will assist in identifying the external parties most suitable for the testing/piloting of the TRUSTS developed interoperability solution.
AR 3.3.5	**Administrative interface:** There should be an **administrative interface** for the TRUSTS operator that enables them to **add, update or remove external data markets supporting the developed interoperability solution** from the TRUSTS platform. This will be the mechanism by which the TRUSTS operator decides which data markets to connect to. The process can be facilitated by automatically linking data markets listed in the Registry of data markets.

2.1.6. Architecture requirements related to data governance

The different types of assets that are to be exchanged between participants in the TRUSTS platform have to be described in a consistent and well-defined manner, so that different components involved in the exchange can take appropriate actions. These descriptions, as documented in deliverable 3.7, are of different dimensions which, in turn, are interpreted and acted upon by different classes of components. For example, descriptions pertaining to usage policies are chiefly interpreted by the Smart Contract Executor component, although they also require a human-readable rendering in the different platform interfaces. In this sense, the TRUSTS knowledge graph serves as a vocabulary with which to express the functional requirements of the platform in a way that is sufficient for components to communicate.

The TRUSTS knowledge graph, which comprises the ontology called TRUSTS Information Model, the metadata about assets, participating organizations, policies and components, as well as the controlled vocabularies used in these metadata, is supported on a set of software components. The communication, security and configuration of these components, as well as their specific location within the platform, are dictated by the different uses of the Knowledge Graph across TRUSTS, as well as by intrinsic properties of the components themselves. From these, a set of architectural requirements were first collected and reported upon in Section 2.1.6 of deliverable 2.6. After a first iteration of platform implementation, and the conclusion of the first stage of use-case trials, these requirements have undergone small alterations, as documented below.

AR 3.4.1 (Every TRUSTS platform instance is a distributed set of nodes), **AR 3.4.2** (Each node runs software components), **AR 3.4.4** (Each node has an internal directory of running software components), **AR 3.4.7** (A node must be able to run apps), **AR 3.4.10** (Recommendations in a node), **AR 3.4.10** (Managing vocabularies in a TRUSTS ecosystem), and **AR 3.4.12** (User interfaces in a TRUSTS ecosystem) remain unchanged from the description enclosed in D2.6.

AR 3.4.3 (Each node runs an instance of a connector implementation) has been modified in three ways: i) the reliance on the specific IDS Connector implementation has been relaxed. ii) The requirement for exchange of IDSCP messages has been updated to IDSCPv2, iii) the connector is now required to expose a REST-ful API that can serve as backend to the different platform interfaces. iv) This API must expose endpoints to satisfy

the entire resource-onboarding process: policy, resource, distribution and artifact creation, as well as linking between these different types of entities.

AR 3.4.5 (A node can internally be a distributed node) This requirement, meant to support high-throughput data processing, has yet to be fully realized as of writing. However, it has become apparent that such distributed nodes would require an additional routing mechanism not mentioned in D2.6. Namely, messages exchanged between the different components in this service must be handled in a manner which is transparent to the Connector implementation.

AR 3.4.6 (The TRUSTS platform has at least one Central Node) The importance of the Central Node has become clearer after the first implementation phase. In addition to the details specified in D2.6 about the central node, the following extra requirements for the central node have been identified from the point of view of metadata management and data governance

- It must host an IDS Broker component that can respond to SPARQL queries about the metadata it collects from other nodes,
- It must provide security mechanisms to ensure that only queries coming from the right nodes are served.

AR 3.4.8 (Managing of assets in a node) The notion of an Asset Server, as mentioned in D2.6, has seen several concrete implementations in the first iteration of platform development. Namely, for the case of Datasets, the CKAN component which provides interfaces also provides the functionality of serving assets, and, for the case of services and applications, other containers running within the node satisfy this role. In that sense, an Asset Server is not a specific component but, rather, a role that several components (both TRUSTS- and organization-provided) can play. Any component playing that role must satisfy the following requirements not mentioned in D2.6:

- They must maintain a consistent hostname within the node, which is to be made known by the node's connector and Dataflow Routing mechanism (see also AR 3.4.13 below).
- The exact access procedure to their endpoints must be describable using the TRUSTS-IM (see D3.7).

AR 3.4.9 (Harvesting of metadata in a node). The metadata mapper is tightly coupled to the interoperability with external data initiatives (as per Task 3.3). However, this operation is also envisioned to be deployed in any corporate node in the case where large amounts of datasets are to be offered. In this case, it is required that all dataset ingestion processes be performable in a programmatic manner (i.e., not only through a user interface). For this, the new requirement of the Connector to expose a rest API (See above) must be satisfied. Likewise, routing to whatever component is hosting this large quantity of datasets must be supported by the Dataflow Rouging mechanism.

AR 3.4.13 (Dataflow Management) A new requirement has emerged regarding the routing mechanism. Namely, the routing mechanism must consult the TRUSTS KG in order to properly route requests. Namely, a node consuming an asset must set up an endpoint for the clients (e.g., browser) that it is serving.

2.1.7. Architecture requirements related to platform development and integration

The architecture developed in TRUSTS must accommodate the reuse and integration of existing software components, software components provided by the above-mentioned initiatives. In addition, components developed by the project's technical partners need to be integrated into the TRUSTS platform. The new platform should be easily deployable into widely used operating systems. As many potential participants as possible should be covered, participants who connect new or existing information systems to the TRUSTS platform. The platform should provide access to the TRUSTS ecosystem in a secure way for both corporate and individual users, provided they satisfy the requirements for participation.

Implementation of the TRUSTS platform has already begun at this point in the project. An implementation of the TRUSTS platform that integrates a selection of existing open-source components with newly developed components is being further developed and tested in order to cover the functional requirements of the

project. We have concluded that the platform architecture should be built around flexible and secure components that can be easily deployed. In addition, infrastructure should be offered to provide flexibility with regards to the types of services or protocols used in order to maximize the types of existing solutions which can be offered by participating organizations.

The TRUSTS architecture should be independent from an operating system, easy to deploy, and simple to adapt. In addition, one of the most important considerations is the preservation of security and privacy by the platform. For this reason, the platform should be built as a network of secure and sovereign nodes which can communicate in a peer-to-peer fashion. In addition, each node should have the means to control how any data is used by any of the services and applications which are deployed on that node. This also allows the mitigation of security breaches by easily excluding potentially compromised nodes. Finally, the TRUSTS platform should enable the deployment of services and portable applications independent of the technology or development environment used to implement the service of applications, as long as a small set of API requirements is fulfilled.

The resulting requirements for the architecture remained unchanged and are as follows:

Requirements related to platform development and integration	
AR 3.5.1	**Communication via a connector instance:** Communication between TRUSTS nodes should follow a standardized and secure protocol. It should be possible to easily add new nodes to an existing ecosystem. For this purpose, a connector component should be used as a universal communication component.
AR 3.5.2	**Running a connector instance:** The source code of the connector component should be available for modification, if needed. In addition, any required credentials or security certificates for running the connector, should be provided by the TRUSTS operator.
AR 3.5.3	**Communication between a connector instance and an application:** The Connector should provide means to communicate with any application inside the same node, using routing of incoming and outgoing flows and this routing should allow for dynamic changes. This communication should accommodate a large set of technologies used to implement an application.
AR 3.5.4	**Securing a connector instance:** The Connector should support the (existing) authentication, authorization and security system inside the node.
AR 3.5.5	**Connecting an application to a connector via an adapter:** It should be possible to create adapters for applications inside a node, which cannot be directly connected to the Connector.
AR 3.5.6	**Running components via docker:** All infrastructure which is provided by the TRUSTS platform in the form of shared components, should run on Docker-Compose. This should include the connector, adapters and other shared components.
AR 3.5.7	**Usage control in a TRUSTS node:**

	All components running in a TRUSTS node should be connected to the usage control component in order to check if any user-initiated action is permissible.
AR 3.5.8	**Metadata and security in a TRUSTS node:** All infrastructure components provided by the TRUSTS platform should support the metadata and security infrastructure of TRUSTS.
AR 3.5.9	**Configurability of TRUSTS nodes:** All infrastructure components provided by the TRUSTS platform should allow any changes to TRUSTS functionality with minimum coding. These changes should be done easily and using configuration files where possible.

2.1.8. Architecture requirements related to brokerage and profiles for users and corporates

Another area of concern for the TRUSTS platform is brokerage and profiles for users and corporates (FR 6, FR 7, FR 8, FR 9, FR 17 and FR 25).

Specifically, the main objective related to this is the realization of a mapping between offerings and demands of users, datasets and services. To realize this, a recommender system will be developed with the following three goals: (i) extract, process and store user interactions (e.g., downloads) and metadata of users/corporates, services/applications (we use the term “services” in the following) and datasets, (ii) develop and provide recommendation algorithms to interlink users/corporates, services and datasets in this tripartite recommendation setting, and (iii) collect user feedback (e.g., clicks) to evaluate and tune the recommendation algorithms.

Based on these goals, three requirements for the architecture of the TRUSTS platform were identified. The main requirement for a recommender system to work is data. Thus, the first goal deals with data handling, processing and storing, which requires a mechanism in the TRUSTS platform that informs the recommender system about new potential resources to be recommended (i.e., datasets and services) and interaction data that the recommender system can use to train its algorithms (AR 3.6.1). The second goal deals with the development and integration of recommendation algorithms. Here, the recommender system requires a mechanism in the TRUSTS platform to give context to the recommendation requests (e.g., the current user) and to present recommendation results (AR 3.6.2). Finally, the third goal is related to the evaluation and tuning process of the recommender system. This translates to the requirement of providing users of the TRUSTS platform with functionality for interacting with recommendation results (AR 3.6.3). More details to these requirements are given in the table below:

Requirements related to brokerage and profiles for users and corporates	
AR 3.6.1	**Notification mechanism to provide data for the recommender system.** In order to provide the recommender system with data for training its algorithms, the TRUSTS platform should provide a mechanism to transfer data to the recommender system. Therefore, the recommender system will provide REST-based services to add (i) metadata of datasets, (ii) metadata of services, (iii) metadata of users, and (iv) interactions between those entities (e.g., if a user downloads a dataset). The metadata broker, or other components of the TRUSTS platform, should use these services in order to notify the recommender system when new entities or interactions come into the platform or when existing entities are changed. Alternatively, the metadata broker should provide standardized query interfaces for the recommender to query for changes to the set of offered resources.

AR 3.6.2	**User interface component to show recommendations.** For visualizing recommendation results to users, the TRUSTS platform should provide a user interface component that is capable of showing an ordered list of recommendations. For this purpose, the recommender system will provide REST-based services for six recommendation settings: (i) recommend datasets to users, (ii) recommend services to users, (iii) recommend datasets to services, (iv) recommend services to datasets, (v) recommend datasets to datasets, and (vi) recommend services to services. The TRUSTS platform needs to use these services to query recommendations by providing parameters such as the current user, the currently viewed dataset or service, one of the six mentioned use cases, the algorithm (e.g., collaborative filtering or content-based filtering) and the number of recommendations to generate (the default value is 10).
AR 3.6.3	**User interface component to interact with recommendations.** When recommendations are shown to users, the TRUSTS platform should also allow them to interact with the recommendations, i.e., click on the recommendations to get additional information. Thus, for every recommendation request, the recommender system will generate a unique recommendation ID that is provided with the list of recommendations. The TRUSTS platform should store this recommendation ID and, whenever a user interacts with a recommended entity, inform the recommender system about this interaction, which is interpreted as feedback to the recommendation. With this, the recommender system is able to evaluate the quality of the recommendations and adapt the algorithms if necessary. Furthermore, this allows us to conduct A/B tests and compare the quality of two types of algorithms (e.g., collaborative filtering and content-based filtering).

When referring these requirements to the system architecture of the TRUSTS recommender systems (see D3.12 and figure below), we can see that all interactions between the TRUSTS platform interface and the recommender system happen via the *service provider* component of the recommender system. However, data received via AR 3.6.1 will directly be passed to the Data Modification Layer (and with this to Solr), recommendations for AR 3.6.2 will be generated via the Recommender Engine (with input of the Recommender Customizer), and feedback received via AR 3.6.3 will be processed by the Recommender Evaluator.

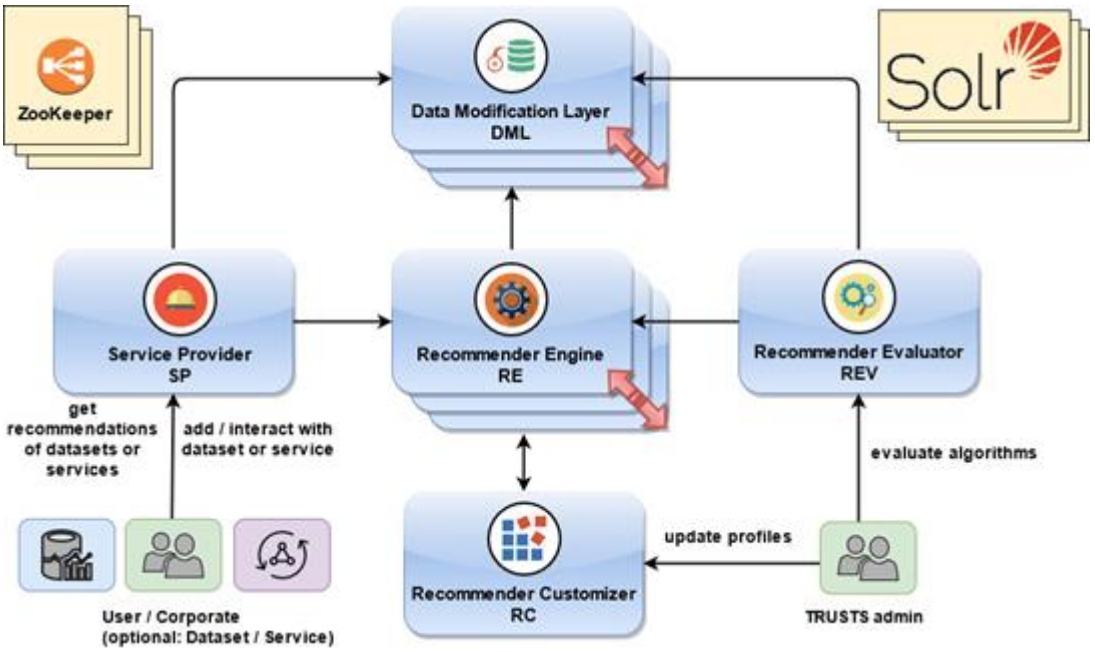


Figure 2: System architecture of TRUSTS recommender system.

Finally, in the current architectural vision of the TRUSTS platform, as described earlier, the sources of the information mentioned above are threefold. First, the Broker which will register metadata on assets will make available messages (or relevant parts thereof) regarding creation/modification of metadata to the recommender system. Second, the contracting system, which will serve as a distributed ledger of transactions, will be the source of data regarding user-asset interactions. Finally, the different user interfaces of the platform will provide information regarding the interactions.

2.1.9. Architecture requirements related to privacy enhancing technologies

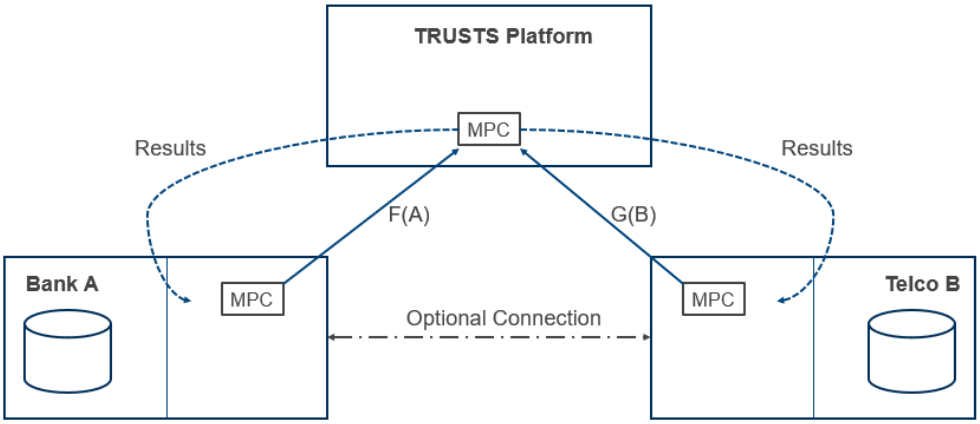
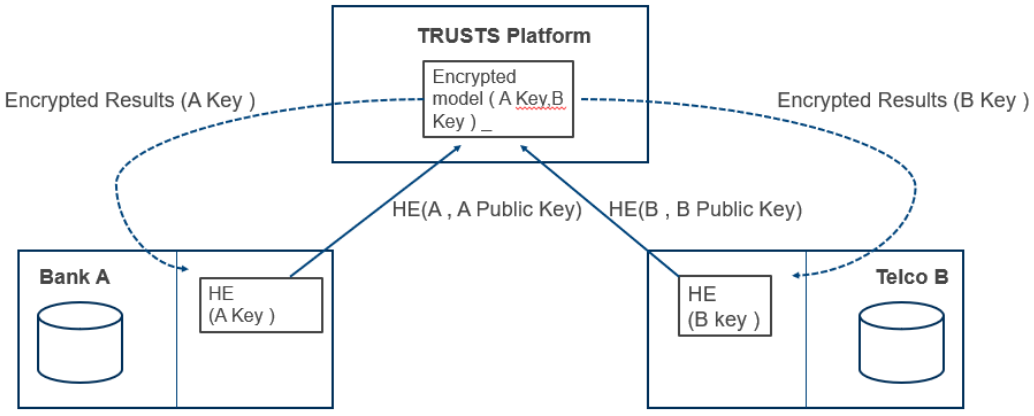
Privacy enhancing technologies (PETs) are an additional concern of the TRUSTS platform. Related to this are the objectives of investigating, designing and improving cryptographically secure protocols that enable data analysis of privacy-sensitive data. Consequently, the focus will be on practical aspects of cryptographic building blocks such as, but not limited to, secure multiparty computation and homomorphic encryption.

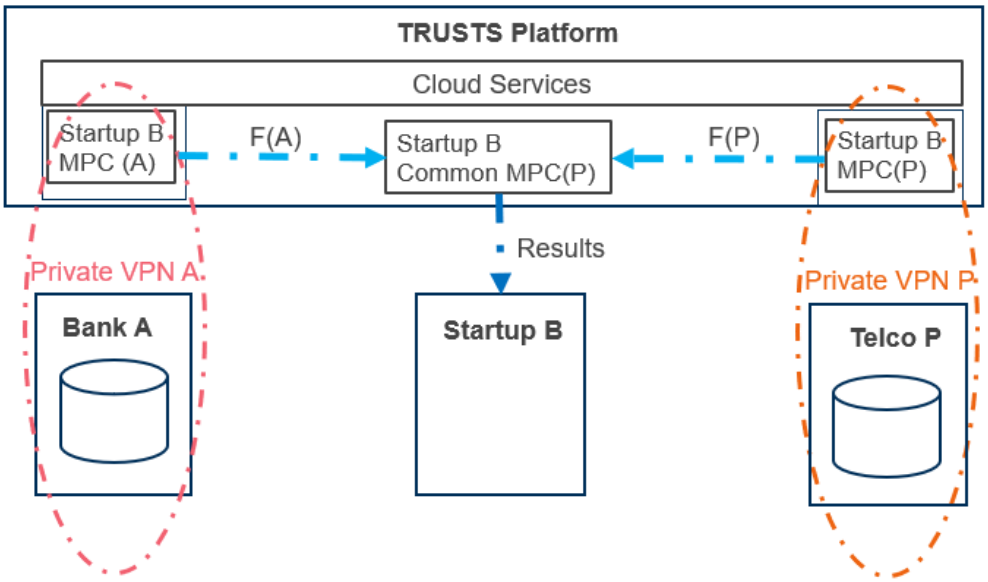
The results of this research will enable parties to collaborate over their private sensitive data and run advanced analytics on multi-party datasets while preserving the data privacy.

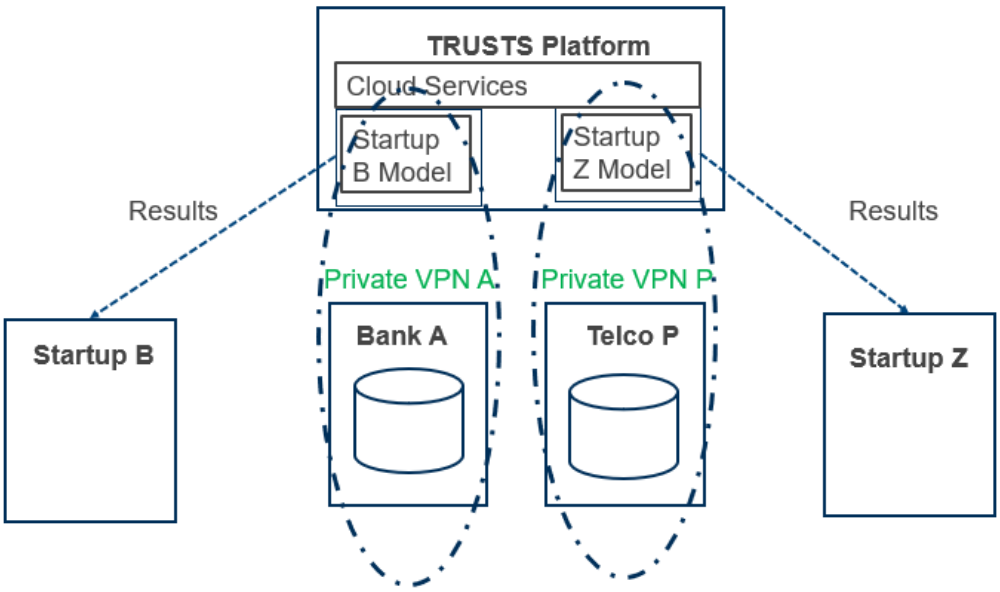
The requirements below support several methods to preserve data privacy. These requirements can be fulfilled by using data encryption methods such as Homomorphic encryption and sharing that encrypted data. Alternatively, other related methods can be used, like Ensemble Modeling that enables advanced analytics over multiple datasets. Each of the data owners will execute the analytics separately on their premises while sharing only the models' algorithm, which will preserve the data privacy.

The work on privacy enhancing technologies for the TRUSTS platform results in the following architecture requirements:

Requirements related to privacy enhancing technologies	
AR 4.1	**Computation using distributed and controlled execution environments:** Since private personal data cannot leave the premises of the data owner, TRUSTS must provide an option to run analytics on the data while preserving the privacy of the users contained in the data. In order to collaborate over private personal data, TRUSTS should

	supply the kind of cloud services that will be part of the computation as described in the illustration below. In case of using Homomorphic Encryption, each party encrypts its own data using its own generated encryption key. This private key that is held all the time by the data owner is used to decrypt the data and the analytic results. Parties do share their encryption public key which is used by the common computation, but it cannot be used for decrypting the data. In other cases where Homomorphic Encryption is not used, the data that is shared for the computation cannot reveal the privacy of the raw data.  Figure 3: Illustration for AR 4.1: Computation using distributed and controlled execution environments
AR 4.2	**Machine Learning using distributed and privacy preserving technologies:** In order to execute Machine Learning models in TRUSTS, the platform must provide a way to develop, test and execute ML models, while supporting MPC /HM/Vertical Federated learning and similar services.  Figure 4: Illustration for AR4.2: Machine Learning using distributed and privacy preserving technologies AR 4.2 illustrates the option of two or more companies that collaborate using HE and MKHE/Spooky key. A common model will be implemented by the parties and will be held in TRUSTS servers. For each inference/prediction each party will have to send its HE data (Encrypted by its own private key). The model will perform prediction over those inputs and will provide encrypted results to each of the parties, while each result is encrypted with the party specific key, therefore it can be decrypted only by the party (using its private key). Federated Learning - Different parties having different data sets with different feature set can also collaborate using Vertical Federated Learning. In D4.1 an Ensemble Learning was presented as one of the options for VFL (New invention). This type of method enables each

	party to develop its own model over its own data and inject the results in a common ensemble model for common prediction/inference.
AR 4.3	**Option of execution of distributed and privacy preserving technologies on servers provided by the TRUSTS operator:** As described in the previous requirements, in order to collaborate over private sensitive data, the data owner must run the relevant ML model on their own servers. In order to give the data owner an alternative to using their servers for computation, TRUSTS should enable the option to execute the relevant computation on TRUSTS servers, while enabling access only to the data owner. This can be done for example using a virtual private network (VPN) created by the data owner. Figure 5 illustrates the generic use case where a party (in this case a startup) needs to develop an analytic module over two other parties' private data (one buyer and two sellers). Since the parties' data cannot be shared, and it can only be used in secure/encrypted methods which require computation, the collaboration will need to include massive computation. Since the parties are not a cloud service provider, they will not provide computation service to the buyer (they only want to sell their "data" and it is not their DNA to provide computation). In that case TRUSTS will have to supply the computation part. In order to protect the data, each party will have to create a private VPN over TRUSTS nodes. The VPN will be created by each of the parties without using TRUSTS assist, and this is in order to protect their data. (The parties will be the only ones who will have access to it). In this architecture, data privacy will be preserved while advanced analytics can be made.  Figure 5: Illustration for AR4.3: Option of execution of distributed and privacy preserving technologies on servers provided by the TRUSTS operator
AR 4.4	**Option of execution of distributed and privacy preserving technologies on servers provided by a TRUSTS participant themselves:** Private personal data cannot leave the servers or the premises of the data owner, therefore trading with private personal data is not a legitimate option. Due to this, the TRUSTS platform needs to provide an option to trade on the analytics performed on this data. In order to enable this, the TRUSTS platform needs to enable computation on its side,

	allowing the data owner to protect themselves with an isolated environment (such as a VPN), as described in the illustration below:  Figure 6: Illustration for AR 4.4: Option of execution of distributed and privacy preserving technologies on servers provided by a TRUSTS participant themselves
AR 4.5	**Development of distributed services using privacy preserving technologies:** In order to preserve privacy of data while collaborating, a third party always needs to be involved for the common computation. The third party is not exposed to the private data, and it is only providing computation over encrypted data, over computed data, or over public data. In our architecture the TRUSTS platform will provide the third-party capabilities. It will execute the HE common model, the ensemble model, it will be part of the MPC etc. In that case the TRUSTS platform will need to provide an option to develop, test, and execute services using privacy preserving technologies. In addition, capabilities for logging and monitoring of these technologies needs to be provided. Those capabilities need to consider the distributed nature of privacy preserving technologies on the TRUSTS platform.
AR 4.6	**Scalability of the TRUSTS platform:** The TRUSTS platform needs to provide scalability for technologies which are dependent on computation and network throughput for their performance.
AR 4.7	**Ensemble Modeling:** The TRUSTS platforms should enable the exchange of pre-trained ML models between participants wishing to execute Ensemble Learning algorithms.

2.1.10. Architecture requirements related to anonymization and de-anonymization

Data controllers, prior to sharing their data, need to become aware of the privacy risks in their data and apply the appropriate anonymization measures. The privacy risks arise because of de-anonymization: the process of identifying individuals in a dataset that does not contain any personally identifiable information (PII), such as full name and address. To confront the privacy risks, anonymization methods have been introduced.

Anonymization methods distort a dataset in such a way, so that it conforms to a privacy model. In relation to this, the aim is to provide an application that offers de-anonymization risk analysis modules for different kinds of data, as well as anonymization methods that offer compliance to various privacy models. The application's aim is to raise awareness on the privacy risks of a data controller's dataset, aid in the decision of the anonymization measures and their extent and help data controllers comply with the general data protection regulation (GDPR) by protecting the privacy of the individuals in their datasets.

The figure below gives an overview of the usage of the application. Given two actors – a data seller and a data buyer – the usage flow is as follows:

1. The data seller (and optionally the data buyer) downloads the application from the TRUSTS platform to their premises. The application needs to be executed on the TRUSTS users' premises because non-anonymized, privacy-sensitive, personal data are processed by the application, and such data should not be uploaded to the platform. Additionally, to ensure that only verified TRUSTS users are using the application, the platform needs to provide means of remote (i.e., from the users' premises) authentication.
2. The data seller imports their non-anonymized, privacy-sensitive, personal data to the application.
3. The data seller uses the de-anonymization risk analysis modules.
4. The data seller uses the anonymization modules.
5. The data seller transfers the anonymized data to the data buyer. This will be feasible using the capabilities of the Dataspace Connector after filling a suitable formula, confirming that they already applied the risk analysis and eventually applied anonymization, and confirming their responsibility for uploading the data.

The application will be packaged as a docker container to facilitate smooth execution and minimal installation and configuration by the TRUSTS users.

From our point of view, the de/anonymization toolkit is not directly related to the TRUSTS pipeline, in the sense that it will not be executed on the TRUSTS platform, since this would require uploading private data, which is not feasible. Rather, the user should be able to download the tools from the TRUSTS platform and then apply the risk analysis / anonymization on their premises. The risk analysis tool implemented so far is in the form of a docker that is capable for this requirement. If the user uploads the data to the TRUSTS platform after applying risk analysis / anonymization, then this is his/her responsibility to be sure that the data is safe. The assumption is that only safe data should be uploaded to the TRUSTS platform. Therefore, we assume that there will be a mechanism to meet the legal issues, e.g., filling a formula prior to data upload, where the users confirm:

I. that they applied the risk analysis and
II. that they take the responsibility for uploading the data.

Also, we assume that the architecture of the TRUSTS Platform will consider providing a possibility for providing our tools for download. The setup described above is the same concept that is already described in the architecture Section 2.1.10.

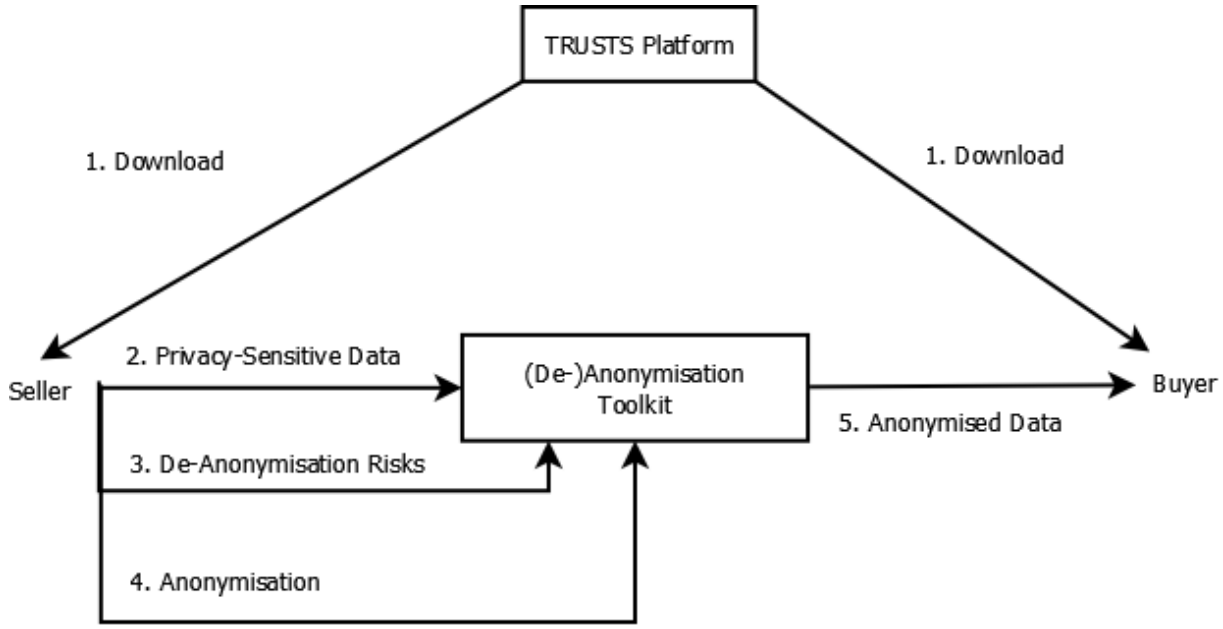


Figure 7: Architecture requirements related to anonymization and de-anonymization

The resulting requirements for the architecture are as follows:

Requirements related to anonymization and de-anonymization	
AR 4.3.1	**Applications need to be available to TRUSTS platform users for download** The task's output is an application with de-anonymization risk analysis and anonymization modules that will be used by data sellers and optionally by buyers if they are willing to double check. Since the application needs to process non-anonymized, private, personal data of a data seller, the application needs to be available for download, in order for it to be executed at the users' premises
AR 4.3.2	**Applications need to be hosted on the platform in a form that allows execution on the users' premises** In order to facilitate smooth execution and minimal installation and configuration, the application will be packaged as a docker container. Such a package, however, could be rather sizable (hundreds of MBs to a few GBs). The platform needs to be able to host such sizable files/packages.
AR 4.3.3	**Login and Authentication for Applications** In order to make sure applications are used only by the TRUSTS platform users, logging in to the application and authentication should require validation by the platform.
AR 4.3.4	**IDS Connector** The transfer of anonymized data from a seller to a buyer has to be done through IDS connectors installed on the premises of the users. The platform should be able to support the installation of IDS connectors on the users' premises.
AR 4.3.5	**Confirmation formula**

	Data seller should fill a suitable formula, confirming that they already applied the risk analysis and eventually applied anonymization, and confirming their responsibility for uploading the data.

2.1.11. Architecture requirements following from the usage of Comprehensive Knowledge Archive Network software

As described in D2.6, the TRUSTS platform includes the pre-existing, widely used, data management software known as CKAN. This component plays the role of user-facing interface, serving as a development framework for all user interfaces that are to be included in the platform. Importantly, the metadata input interfaces that are common to all data management platforms are highly related in functionality to what is required of the TRUSTS platform in terms of depositing and publishing assets. Likewise, CKAN acts as a file serving component into which those wishing to provide a dataset can upload it, in which case CKAN will store it, keep it in sync with its metadata, and prove API's for its access. Finally, the file-storing capacity of CKAN is also exploited for other types of assets, namely, the specification files of applications and services (OpenAPI) are also deposited into and served by CKAN, as well as the deployment files of applications.

CKAN has been selected to become a central component during the establishment of the TRUSTS platform given the following advantages:

- Significant overlap of existing CKAN features with TRUSTS requirements.
- Availability of data cataloging functionality.
- Included search functionality using Solr.
- Deployable using container technology.
- Different publicity levels for assets, i.e., public or private.
- Robust and mature solution. Thousands of productive instances are deployed worldwide.
- Widely used by a plethora of data providers, e.g., governments (e.g., Australian Federal Open Data Portal) or other players (e.g., AQUACROSS).
- Active maintenance by, and responsiveness of, the CKAN community.
- A wide range of existing CKAN extensions that fulfil the requirements of TRUSTS, e.g., a harvesting extension, a DCAT extension, or an OAUTH2 extension.
- The availability of a convenient extension mechanism facilitating the creation of features specific to TRUSTS requirements.

Several other functionalities that were identified in the requirement elicitation process of WP2 will also be satisfied using CKAN, even when the out-of-the-box version does not include them. In this case, extensions are being developed that are to leverage the plug-in mechanisms of CKAN. In particular, the User Interface will be customized and extended, and the metadata captured by the interface will be forwarded to the metadata broker to continue the rest of the ingestion process.

Requirements following from the usage of CKAN	
AR 3.3.9	**CKAN extension mechanism** TRUSTS components should ideally be integrated using CKAN's built-in extension mechanism. Developers need to consider this and design and develop their solutions in a way that they can be easily integrated into CKAN.

2.2. Summary of architecture requirements

In this section, all previously listed architectural requirements are summarized. The architectural requirements are grouped into architecture requirement summaries, labelled ARS. Compared to the summaries listed in D2.6, the ARS have remained the same. A short description, usually one sentence long, of each ARS is provided. The original architectural requirements which are summarized by an ARS, are listed in the right most column. Every original architectural requirement is included in at least one ARS.

Regarding the two new architectural requirements, introduced in the previous section, AR 4.3.5 was added to ARS 5 and AR 3.4.13 was added to ARS 13.

Table 2: Summary of architecture requirements

ARS ID	Short description	Original AR numbers
ARS 1	**Smart contracts, which are based on policies, are supported.**	3.X.1 3.X.5 3.2.1
ARS 2	**All transactions on any node, which is part of a TRUSTS platform instance, are logged.**	3.X.1 3.X.5 3.2.2
ARS 3	**All data assets / data products in a TRUSTS platform instance, use metadata schema or vocabularies.**	3.X.5 3.X.6 3.3.1 3.4.11 3.5.8
ARS 4	**External data marketplaces can be integrated into a TRUSTS platform instance.**	3.X.3 3.3.2 3.3.3 3.3.4
ARS 5	**The operator of a TRUSTS platform instance can use an administrative interface. Users of the platform also have an interface available.**	3.X.4 3.3.5 3.4.12 4.3.5

ARS 6	**Every TRUSTS platform instance is a distributed set of nodes.**	3.X.3 3.4.1
ARS 7	**Each node runs software components, which are distributed as docker containers, and can be configured via configuration files.**	3.X.4 3.X.5 3.4.2 3.5.6 3.5.9
ARS 8	**Each node needs to run an instance of the connector component. It is required for communication between nodes and within a node.**	3.X.1 3.X.2 3.X.3 3.4.3 4.3.4 3.5.1 3.5.2 3.5.3 3.5.5
ARS 9	**Each node has an internal directory of running software components.**	3.X.3 3.X.5 3.4.4
ARS 10	**A node can internally be a distributed node itself.**	3.X.3 3.4.5
ARS 11	**Every TRUSTS platform instance has at least one node which runs an instance of the Broker component.**	3.X.3 3.X.6 3.4.6
ARS 12	**A node must be able to run apps on premise.**	3.X.3 3.4.7 4.3.1 4.3.2
ARS 13	**Assets (such as data assets) in a node can be managed locally.**	3.X.3

		3.X.5 3.4.8 3.4.13
ARS 14	**A node is able to harvest metadata from its local assets.**	3.X.3 3.X.5 3.4.9
ARS 15	**Recommendations in a node are integrated into the TRUSTS platform, which provides data, delivers recommendations and returns feedback.**	3.X.4 3.4.10 3.6.1 3.6.2 3.6.3
ARS 16	**Instances of the connector component support existing security infrastructure inside of a node, and connect it to the usage control of the TRUSTS platform.**	3.X.1 3.X.2 3.5.4 3.5.8 4.3.3
ARS 17	**Computation using distributed and controlled execution environments is possible in every instance of the TRUSTS platform.**	3.X.1 3.X.3 4.1
ARS 18	**Machine Learning using distributed and privacy preserving technologies is possible in every instance of the TRUSTS platform.**	3.X.1 3.X.2 3.X.3 4.2
ARS 19	**Every instance of the TRUSTS platform provides the option of execution of distributed and privacy preserving technologies on nodes provided by the operator of the instance.**	3.X.1 3.X.2 3.X.3 4.3
ARS 20	**Every instance of the TRUSTS platform provides the option of execution of distributed and privacy preserving technologies on servers provided by a TRUSTS participant themselves.**	3.X.1 3.X.2

		3.X.3 4.4
ARS 21	**Development of distributed services using privacy preserving technologies is possible in every instance of the TRUSTS platform. (This refers to Devops.)**	3.X.1 3.X.2 4.5
ARS 22	**Every instance of the TRUSTS platform provides scalability for services and apps running on the platform instance.**	3.X.3 4.6
ARS 23	**The user interfaces of all TRUSTS components are integrated using the CKAN extension mechanism.**	3.X.4 3.3.9

3. Technical architecture of the TRUSTS platform

As the blueprint for the results of the TRUSTS project, the TRUSTS platform technical architecture also represents the foundation for instances of the TRUSTS platform that will be provided by one or more TRUSTS operators after the duration of the project.

This section presents the technical architecture of the TRUSTS platform. The revised technical architecture incorporates feedback from the technical and non-technical project partners. The technical partners provided feedback based on the platform implementation. The non-technical partners provided feedback from an innovation perspective and based on the knowledge gained after the first phase of use case trials, in terms of component descriptions, interconnections and other technical details of the architecture.

First, a general overview of the architecture is given, and the most important concepts of the architecture are summarized. This is followed by a description of each component of the architecture, as well as how components are connected within the platform and to external data marketplaces for the purpose of interoperability and federation. The component descriptions are followed by an overview of the mapping between functional requirements (FRs) and components, and the mapping between architectural requirements (ARs) and components.

3.1. Overview of the technical architecture

The technical functionality of the TRUSTS platform has already been described in D2.6 and is again summarized in this section to provide a high-level overview of the architecture as a whole.

As explained in the first version of this deliverable and as also illustrated in Figure 8, the TRUSTS platform consists of a set of interconnected nodes, which can be of three types: Central Node, Corporate Node, and User Portal Node. Each node is owned and operated by different organizations. The set of interconnected nodes is explained in Section 3.1.1 in more detail. These nodes form the basis for a trusted exchange of digital assets. Each node is a computing environment (e.g., a virtual machine). Each computing environment runs standardized components to support the various functionalities of the platform. Components deployed within a node are interconnected via a network environment provided and secured by the node's operator. Both the nodes and the standardized components running on these nodes are described in more detail below and are illustrated in Figure 8. By using a standardized set of components in a node, a consistent set of functionalities is provided to the TRUSTS platform participants. The functionalities that the platform should provide are defined by the functional requirements, which are described in D2.3. The standardized set of components can also be used to enforce restrictions that affect, for example, trading and access to assets within nodes. The set of components, as well as a description of how these components are used and how they are related to the functional and architectural requirements, can be found in Section 3.2.

Along the components, common to all nodes, organizations can execute their in-house developed applications and services. These applications and services can then be traded through the TRUSTS platform, and can also, in turn, consume and produce other assets through the platform.

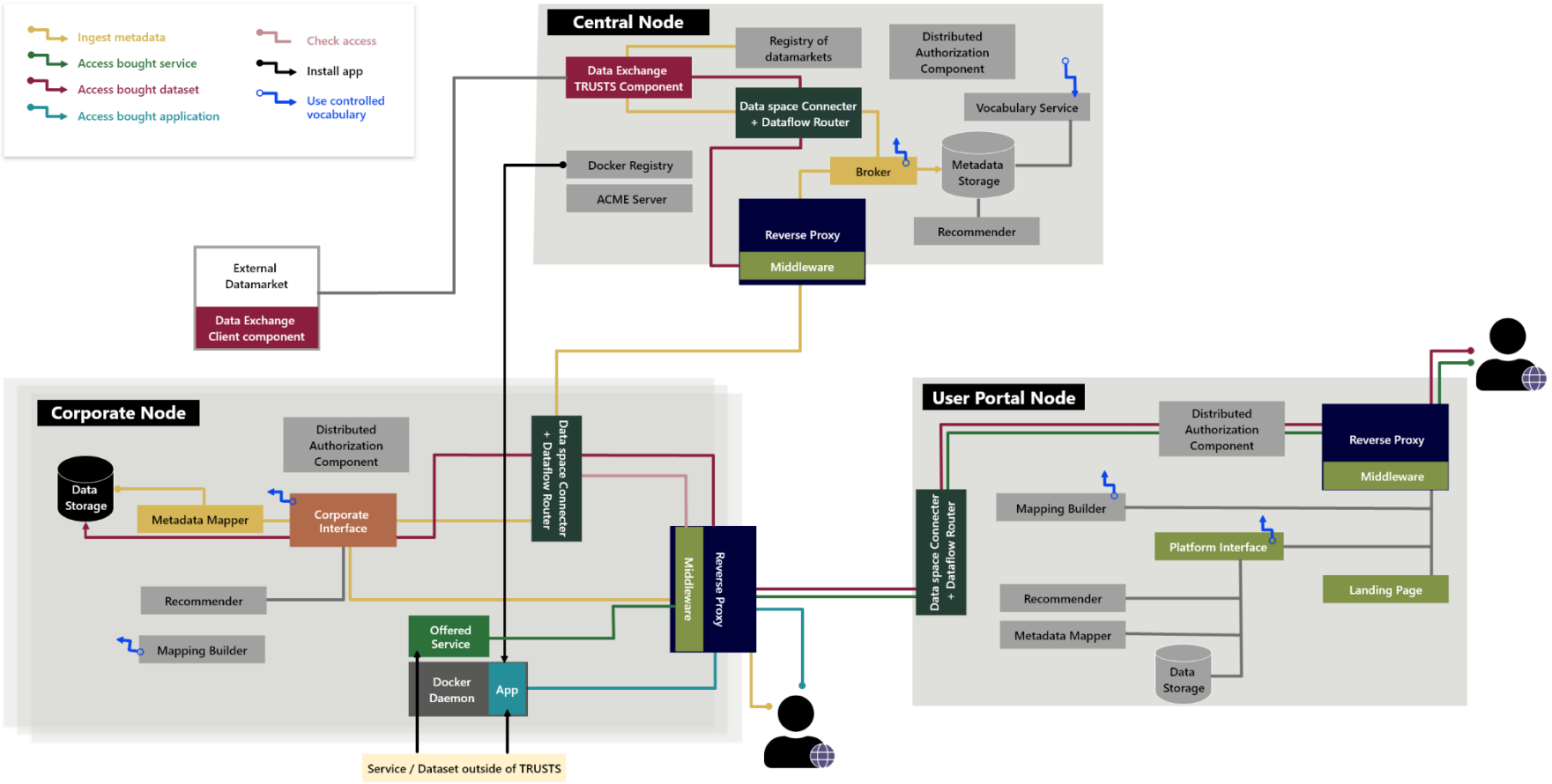


Figure 8: Diagram of the technical architecture

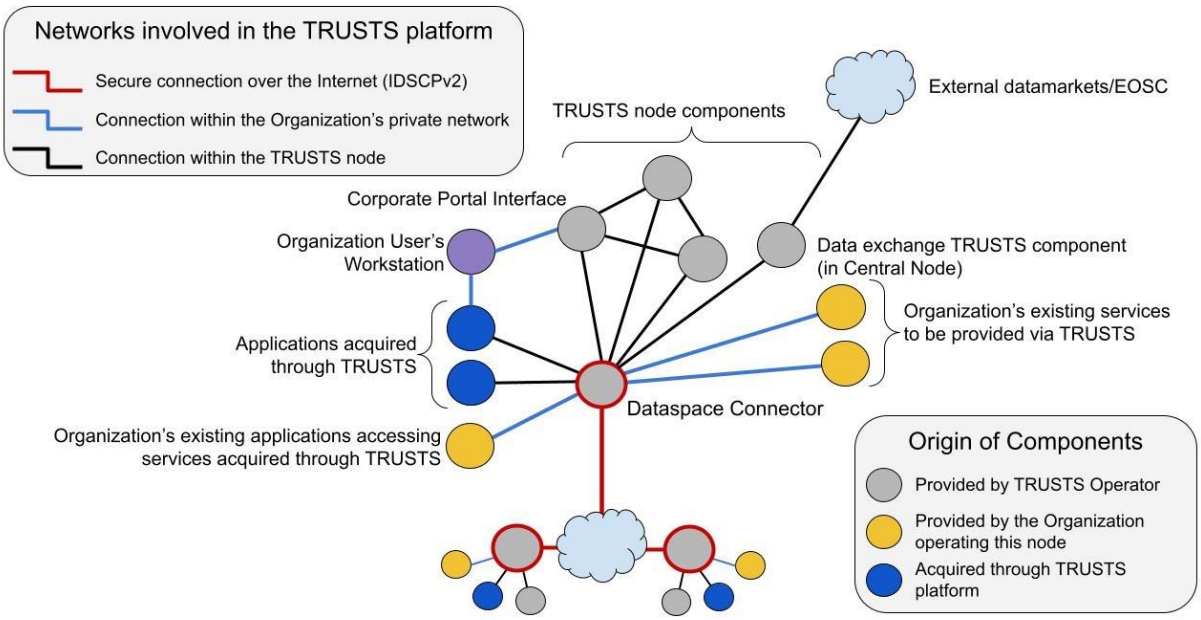


Figure 9: Overview of networks involved in the TRUSTS platform

Connections coming from outside the node are first received by the Connector component. From the Connector component they are distributed to other components. Figure 9 shows an overview of the networks involved in the TRUSTS platform with the connector component as the central entity. In addition, the origin of the components is indicated by color and the respective type of connection is named.

As described in D2.6, the IDSCPv2 communication protocol is used for communication between the interconnected components. The IDSCPv2 protocol, apart from the standard asymmetric-key encryption of modern HTTPS connections, allows for conveying information specific for the functionalities of the platform. This includes user identification tokens, names and ports of the different nodes involved, as well as IDs of the components they are destined to. A schematic of these network connections is shown above.

The previous version of Figure 9 did not include the network connection to external data markets, which is established via the standardized Data exchange TRUSTS component within the Central Node. The interoperability of the TRUSTS platform with external data markets and EOSC initiatives is described in Section 3.1.4.

3.1.1. Types of nodes

The nodes shown in Figure 8 are those that are already described in the previous version of this deliverable: **User Portal Node**, **Central Node**, and **Corporate Node**. The two nodes that must be hosted by a specific organization, referred to as the TRUSTS Operator, are the User Portal Node and the Central Node. These nodes host a set of services necessary for the operation of the platform as a whole. Accordingly, the TRUSTS Operator occupies a special position of trust with respect to the other organizations hosting nodes. Any other node is referred to as a Corporate Node.

- **Corporate Node.** Is used by partners of TRUSTS that have complex infrastructure and participate in TRUSTS. They can have many users and many services and applications that are provided or consumed via the TRUSTS platform. They can set any authorization system and communication structure inside their nodes. The TRUSTS Operator is not responsible for setting up or supporting instances of corporate nodes but will provide detailed instruction manuals and all necessary

software. Among the components being set up, is the Corporate Interface, a web application through which the organization's users can interact with the TRUSTS platform.

- **User Portal Node.** Is created to cover the needs of individual users of the TRUSTS platform. It is set up and maintained by the TRUSTS Operator. By accessing this node, individual users can benefit from some of the functionalities of the TRUSTS platform, without the complexity of setting up a corporate node. Additionally, it is the entry point for all organizations intending to join the TRUSTS platform, as it provides landing pages, legal information, and setup instructions. One of the components included in this node is the platform Interface, which acts as the main point of access to the above-mentioned functionalities. In principle, the TRUSTS platform could have more than one User Portal Node.
- **Central Node.** Exists to support the operation of the whole TRUSTS platform, playing the role of authorization, monitoring, smart contract executor, catalog, application repository, among others. This node is created and maintained by the TRUSTS Operator.

For an organization to install a TRUSTS node, they must enter a subscription agreement with the TRUSTS operator, after which they will be provided with a series of software artifacts, cryptographic certificates and instruction manuals for the set-up. This will involve, among other steps, creating a dedicated computing environment with a controlled set of open ports and other firewall rules. These rules shall imply, for example, that the only connection between the node and the internet will be through the Connector component. Furthermore, it shall allow for other connections inside the organization's private network, in order for applications acquired through the TRUSTS platform to be accessible to the organization's users, for example. Finally, a set of technical information exchanges should be carried out between the TRUSTS operator and the node operator, for example, regarding a set of IP addresses and ports which are to be contact points between the nodes.

3.1.2. Data sets / services / applications

The three types of assets that are envisioned to be traded among the participants of the TRUSTS platform have remained. The three types of assets are listed below.

Dataset. These are static files which can be traded. They are equivalent to the notion of DCAT:Dataset, in terms of metadata and scope. These files are transmitted directly from provider to consumer, and once they have been received, they are, from the technical point of view, mere files in the consumer's file system.

Service. These are computer programs which are executed in the provider's node. They can, upon contractual agreement, be executed on request of a consumer with the input of their choice. Services are envisioned to function as do standard web services: a server is running and accepts connections, processes the input, and returns the output to the requester. There is no prescription on the stateful-ness of services, but they must expose an HTTP interface. Services are accompanied by a description bundle, which consists of a standardized description of the different endpoints that it exposes, in order to enable automatic configuration of routing mechanisms and of clients.

Application These are computer programs which are executed in the consumer's node. They are distributed as description *bundles*, which consist of a combination of container images, deployment scripts, configuration files, and standardized descriptions of the endpoints exposed by the application. The provider of the application has total freedom regarding implementation and functionalities and can choose to make the application accessible only through the consumer node's Dataspace Connector instance, thus enabling the use of TRUSTS provided functionalities, such as user authentication.

3.1.3. Connections within the TRUSTS architecture

Seven specific data flows are illustrated in the general architecture diagram (Figure 8). These are meant as examples of the interactions between the different components in different nodes.

Communication protocols used by components within the TRUSTS platform

To properly and fully integrate components and services within the TRUSTS platform, those components and services need to provide a description of existing interfaces based on the best practices which are summarized by the REST (representational state transfer) paradigm. The REST paradigm for describing services is the most widely used web service interconnection approach and the most common standard in interaction between distributed software components. Additionally, services using REST are much easier to test compared to other service specifications, such as MQTT and SOAP. RESTful services have no restriction on the size of messages and are a lightweight approach to integration. RESTful services are supported by the connector component, which is used in the TRUSTS platform.

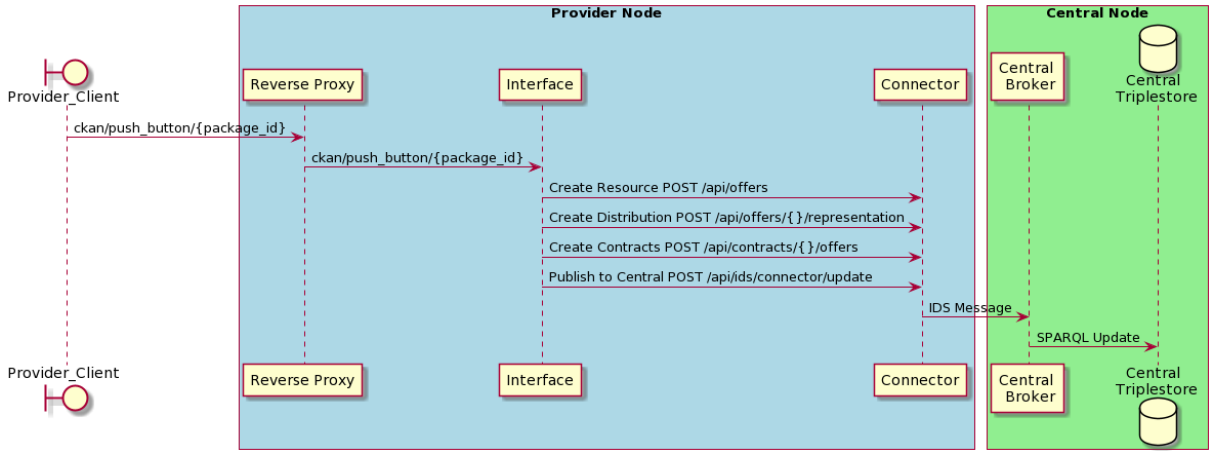
Ingest metadata

When an organization wishes to make an asset, it possesses available through the TRUSTS platform, it must make metadata about the asset available. This process is referred to as onboarding of assets in the requirements documents.

The origin of the metadata can be a metadata storage about datasets (e.g., a directory in a file system, a csv file, a data management platform), a machine-readable description of a web service, or a standardized description of deployment and configuration of an application. These three sources are all accessed, as per input by the user in charge of onboarding, by the Corporate Interface running in the organization's node, which collects the metadata and adds to it metadata about the node itself. This process can be assisted by the Metadata Mapper component which can convert from several metadata schemas into the one being used by the TRUSTS platform. Finally, the Corporate Interface also provides the user with the means to specify the contracting options by which the asset in question will be made available.

When the user is ready to make the asset available for purchase and access through the TRUSTS platform, they can request the Corporate Interface to send information about it to the Broker component present in the Central Node. The broker then stores this metadata in the authoritative central metadata repository, which the Platform Interface in the User Portal Node, or the Corporate Interface in any Corporate Node, will query.

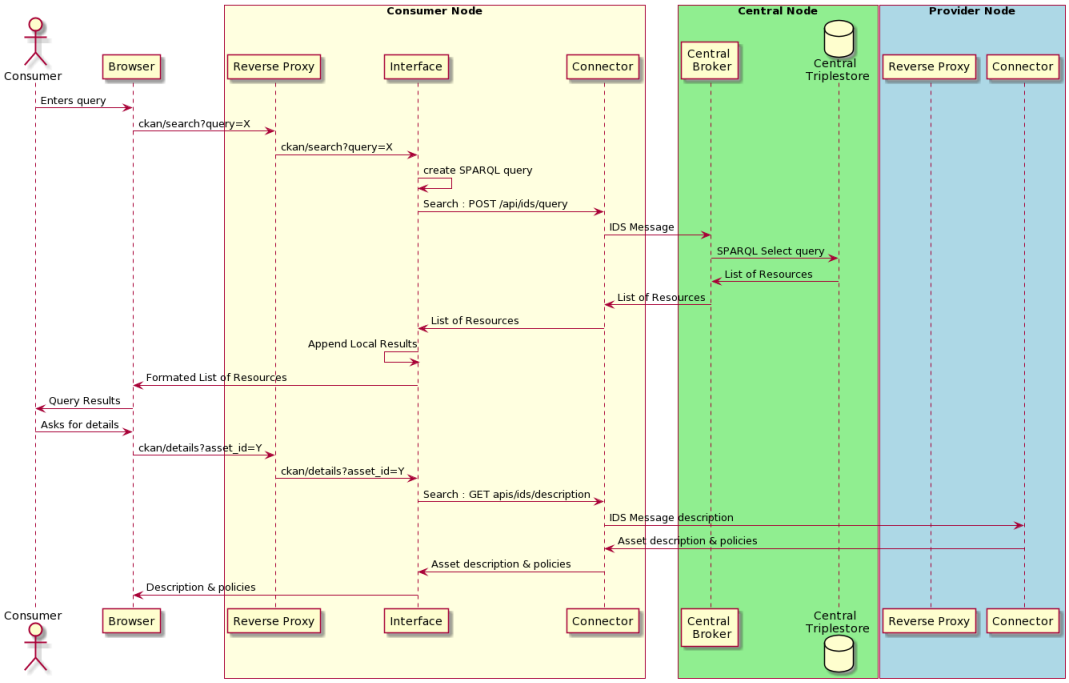
Ingest:



Discover and buy asset

When a user wishes to acquire an asset through the TRUSTS platform, they must first present credentials either in the Platform Interface or in any Corporate Interface to which they have access. This interface will present them with the tools necessary to search, select an asset and select access options (including contracting and billing). Once the process is complete, a record of the transaction is entered into the responsible component in the Central Node. Currently, the Dataspace Connector is responsible for rudimentary contract negotiations. In the future, the Smart Contract Executor will take over this functionality and be added to the architecture accordingly.

Discover:



Buy:

Buying an asset consists in

1. The consumer chooses a contract and sends the offer to the provider.
2. The provider accepts the offer (which can be done automatically).
3. The provider connector grants the consumer the appropriate access rights to the asset and sets up the internal routing so that the asset is reachable.
    1. In the case of datasets, these routes will point to the corresponding asset providing component (CKAN in the case of manually uploaded datasets).
    2. In the case of services and applications, these routes will point to the provider's CKAN instance so that the consumer can download the asset's description bundle.
    3. In the case of services, additional routes must be created on the provider node so that subsequent access to the service is also adequately routed.
4. The provider transmits the consumer the URI by which they can access the specific artifact purchased (an artifact is a concrete copy of an asset, subject to a specific policy; similar to an actual book in a library which the user can take home. See D3.7 for more details).
5. The consumer's connector creates the routes so that the user's client (or browser) can request the asset in their connector, and the request is adequately routed to the provider so that the dataset or description bundle can be downloaded.

6. In the case of services, the description bundle must be downloaded and additional routes are to be generated on the consumer DSC to allow for access to the different REST endpoints provided by the services.

Access bought asset

A user accessing an asset that they have acquired does by connecting to different endpoints in the node corresponding to their organization. These requests will then be processed by their node's Data Space Connector to check for access control, and will then be forwarded to the providing node's DSC which will also check the access rights and, if appropriate, forward the request to the component serving the asset. The exact sequence of these access operations depends on the types of assets being accessed, and are explained in more detail below.

Access to Applications. Accessing an acquired application involves two different operations. The first operation, called *Installation*, consists of 1) The consumer downloading the applications description bundle 2) an execution of a pull operation on the consumer's docker daemon, which results in the application images (as specified in the description bundle) being downloaded to the consumer node. 3) creation, in the consumer's DSC, of the routing necessary so that the application's different endpoints are accessible through the DSC.

Check access

The Dataspace Connector provides a set of rules for contract negotiation. A provider can choose between different types of offers, e.g., a certain number of accesses or a certain period during which access is granted. A consumer can choose between different offers.

Use controlled vocabularies

Several components within the TRUSTS platform make use of controlled vocabularies for, e.g., storing and displaying labels describing assets, or enriching search results. This will be done by accessing a centralized Vocabulary Service, running in the Central Node, which will be kept up to date by the TRUSTS operator. Access to this service will be done using the SPARQL protocol, as well as a set of well-specified API endpoints that the different components can use to check for updates.

3.1.4. Interoperability with external data markets and EOSC Initiatives

TRUSTS will develop a component, the so-called interoperability component, to connect to third-party data markets and EOSC initiatives. This component will allow external stakeholders to connect with TRUSTS and share and exchange their resources on TRUSTS. The interoperability solution consists of two components. One component resides on the premises of the external data market or EOSC initiative. The other component is a TRUSTS component that interacts with the client component. The components exchange information such as metadata, which in turn will be stored in the storages of TRUSTS. The two mentioned components are called Data Exchange Client Component and Data Exchange TRUSTS Component and are supported by the Registry of Data Markets. This is illustrated in Figure 8. The Registry of Data Markets component contains up-to-date information about existing data markets and will help to identify data markets suitable for building the interoperability prototype. The three components mentioned are described in more detail in the following section.

As described in D2.6, a significant challenge of the development of this component is the strong diversity of the technological characteristics of external data markets and EOSC initiatives. Consequently, TRUSTS will not develop a solution for each of them. Instead, TRUSTS aims to build a solution exposing the requirements

of TRUSTS, i.e., the TRUSTS metadata schema, which is in turn an extension of the IDS Information Model. It is an open, non-proprietary model, based on widely accepted standards (such as DCAT) and covers a wide variety of technological requirements. External stakeholders will turn their metadata and data assets into a format understood by TRUSTS. The interoperability solution will be developed based on the requirements of a select set of external data markets and EOSC initiatives. Since interoperability is a two-way action, TRUSTS will also include the means for external parties to transfer the data into the TRUSTS platform. For example, in the case of EOSC, it is planned that TRUSTS becomes a so-called EOSC provider. By becoming an EOSC provider, the data assets of TRUSTS can also be shared with EOSC and made accessible via their infrastructure.

3.2. Components of the architecture

In this section, all components needed to implement the technical architecture above are introduced. For each component the following attributes are listed:

- Component number.
- Component name.
- Short component description.
- Input dependencies: the names of other components which provide input for the component.
- Output dependencies: the names of other components which require output from the component.
- Dependency changes with respect to D2.6 are marked as blue and strike-through text.
- Requirements addressed: references to the functional and architectural requirements which are addressed by the component.

<table>
<tr><th>C1</th><th>Dataspace Connector</th></tr>
<tr><td colspan="2">Interface to nodes in the TRUSTS platform. Provides certified connections between pairs of nodes. It is a tool for data asset definition, creation and access granting or prohibition. It offers a mechanism for data asset selling and buying (creation of offers and contracts). It supports a negotiation mechanism.</td></tr>
<tr><td>Input dependencies:
Dataflow Router
Corporate Interface
Automated Certificate Management Environment (ACME)
Distributed Authorisation Component
Metadata Broker</td><td>Output dependencies:
Corporate Interface
Platform Interface
Broker</td></tr>
<tr><td colspan="2">Requirements addressed:
FR 27, FR 28, FR 29, FR 31, FR 38, FR 39, FR 40, FR 41, FR 42, FR 43, FR 44, NFR 3, NFR 6
ARS 2, ARS 7, ARS 8, ARS 12, ARS 13, ARS 16, ARS 17, ARS 18, ARS 19, ARS 20, ARS 21</td></tr>
</table>

<table>
<tr><th>C2</th><th>Dataflow Router</th></tr>
<tr><td colspan="2">Add-on to the Dataspace Connector that initializes and destroys routes within a node as required by incoming and outgoing connection. For example, if there are N services running in a node (e.g., a CKAN</td></tr>
</table>

instance, some APIs) and the Dataspace Connector receives a request to access one particular service, this component creates a route in the Apache Camel configuration of the connector.

Input dependencies:	**Output dependencies:**
• Corporate Interface • Platform Interface • ~~Usage Control~~ • ~~Trusted Connector~~ • Dataspace Connector • Reverse Proxy • ~~Services Consumer Adaptor~~ • Operation Converter	• Corporate Interface • Platform Interface • ~~Usage Control~~ • ~~Trusted Connector~~ • ~~Services Consumer Adaptor~~ • Operation Converter • Reverse Proxy

Requirements addressed:

FR 27, NFR 6

ARS 8, ARS 9, ARS 10, ARS 16, ARS 22

C3	**Reverse Proxy**

For connections from a web-browser into nodes (e.g., for accessing portals), this component distributes the requests among the different services. Implementation can be Nginx, Traefik, HAProxy or similar. It must support the functionality of "middlewares", namely, to be able to intercept requests, perform arbitrary operations based on their parameters, like contacting external services, and then forward these requests, possibly after URL rewriting.

Input dependencies:	**Output dependencies:**
• Automated Certificate Management Environment	• Corporate Interface • Platform Interface • Mapping Builder • Notification Service • ~~Asset consumer~~ • Business Support Services

Requirements addressed:

NFR 6

ARS 5, ARS 23

C4	**Recommender System**

Provide six recommendation use cases (RUC): (RUC1) recommendation of datasets to users, (RUC2) recommendation of services to users, (RUC3) recommendation of datasets to services, (RUC4) recommendation of services to datasets, (RUC5) recommendation of datasets for a given dataset, and (RUC6) recommendation of services to a given service. For this, the recommender system processes dataset and service metadata as well as interactions between users, services and datasets. Additionally,

the recommender system is evaluated and fine-tuned via implicit feedback (e.g., clicks on recommendations).	
Input dependencies: • Broker + Metadata Storage • Platform Interface	**Output dependencies:** • Platform Interface
Requirements addressed: FR 6, FR 7, FR 8, FR 9, FR 17, FR 25 ARS 15	

C5	**Platform Interface**
This component should support users with opening accounts, searching for datasets and services (free and chargeable) and consuming them. It should provide means to sign contracts with providers of datasets and services. Additionally, it allows consumers to rate dataset and service providers. Finally, it will also serve as default asset provider (i.e., file server) for the case of datasets and description bundles for services being made available through the Platform Interface.	
Input dependencies: • Dataflow Router • Recommender • Vocabulary Service • Metadata Mapper • Broker + Metadata Storage	**Output dependencies:** • Dataflow Router • ~~Trusted Connector~~
Requirements addressed: FR 1, FR 3, FR 4, FR 5, FR 25, FR 27, FR 29, FR 30, FR 31, FR 32, FR 33A, FR 33B, FR 34, FR 36, FR 37, FR 44, NFR 1, NFR 2, NFR 3, NFR 4, NFR 5 ARS 5, ARS 23	

C6	**Landing Page**
A website for non-registered users to: • learn about TRUSTS platform, legal terms, contact points, etc. • enroll and login • reset password	
Input dependencies: • Reverse Proxy	**Output dependencies:** • Platform Interface
Requirements addressed:	

NFR 5 ARS 5

C8	Notification Service
This component informs Consumers about changes in datasets and services that they already have access to.	
Input dependencies: • Broker + Metadata Storage	**Output dependencies:** • Notification Service
Requirements addressed: FR 35, NFR 6 ARS 5, ARS 13, ARS 14	

C9	Metadata Mapper
This component transforms existing metadata that organizations might have in their pre-existing data stores, into the TRUSTS-IM specification (see D3.7). Once this metadata is converted, it can be ingested via programmatic API access into the local Dataspace Connector and then broadcast into the Metadata Broker for advertising to other participants.	
Input dependencies: • Mapping Builder	**Output dependencies:** • Platform Interface • Corporate Interface
Requirements addressed: FR 23 ARS 3, ARS 14	

C11	Mapping Builder
A user interface that helps providers build Mapping files to convert their metadata into the format specified by the TRUSTS-IM (see D3.7). The result will be an RDF file complying with the RML specification.	
Input dependencies: • Vocabulary Services	**Output dependencies:** • Metadata Mapper
Requirements addressed:	

FR 23, FR 33A, FR 33B ARS 3

C12	Corporate Interface

This component allows corporate inside TRUSTS Corporate node to manage the assets that they wish to provide through the TRUSTS platform. Also, it should provide means to sign contracts with consumers of datasets and services. Additionally, it allows consumers to rate dataset and service providers. Finally, it will also serve as default asset provider (i.e., file server) for the case of datasets and description bundles for services and applications.

Input dependencies:	Output dependencies:
~~Usage Control~~ Dataflow Router Metadata Mapper Recommender ~~Services Consumer Adaptor~~	Dataflow Router ~~Trusted Connector~~ ~~Usage Control~~ Recommender

Requirements addressed:

FR 1, FR 3, FR 4, FR 5, FR 27, FR 29, FR 30, FR 31, FR 32, FR 34, FR 36, FR 37, FR 44, NFR 1, NFR 2, NFR 3, NFR 4, NFR 5

ARS 5

C14	Data Exchange TRUSTS Component

This component is deployed in TRUSTS nodes and ingests assets from third-party data-markets and EOSC and its related initiatives into the TRUSTS catalog. It receives input from the Data Exchange Client Component and converts it, if necessary, into the format required by TRUSTS. This component is also connected to the Registry of Data markets, receiving information about data markets and EOSC and its related initiatives connected with TRUSTS. Based on this information, the component connects to the respective Data Exchange Client Components and acquires their metadata or data assets.

Input dependencies:	Output dependencies:
Third Party Marketplaces EOSC and its related initiatives Registry of Data markets	Metadata Mapper ~~Usage Control~~

Requirements addressed:

FR 2, FR 4, FR 23

ARS 4

C15	Data Exchange Client Component

This component resides at the location of a third-party data market or EOSC initiative. It provides an interface to specify and select data assets and to map their metadata schema into a format understood by TRUSTS. The component communicates with the Data Exchange TRUSTS Component. Furthermore, it communicates with the Registry of Data markets and updates in case of relevant changes.

Input dependencies:	Output dependencies:
• Third Party Data market • EOSC and its initiatives • Registry of Data markets	• Registry of Data markets • Data Exchange TRUSTS Component

Requirements addressed:

FR 2, FR 4, FR 23

ARS 4

C16	Registry of Data markets

This component lists existing third-party data markets and relevant initiatives of EOSC. On the one hand, this helps to form a community around TRUSTS. On the other hand, the component serves as an address book routing the communication between the Data Exchange TRUSTS Component and the Data Exchange Client Components installed on the premises of third-party data markets and EOSC initiatives.

Input dependencies:	Output dependencies:
• Data Exchange Client Component	• Data Exchange TRUSTS Component

Requirements addressed:

FR 2, FR 4

ARS 4

C17	Business Support Services

This component will be used for monitoring and administration TRUSTS activities. It allows for the managing the list of connected organizations, monitoring of logs, and managing of fees collectable by the TRUSTS operator.

Input dependencies:	Output dependencies:
• Smart Contract Executor	• Reverse Proxy

Requirements addressed:

FR 33A, FR 33B

C18	Broker + Metadata Storage

The broker, and its accompanying metadata store, serves a central repository of all metadata regarding assets, participants and resources, as specified in the TRUSTS-IM (see D3.7). It is a web application that serves as a wrapper of a Knowledge Graph that is compliant with the TRUSTS-IM ontology. It collects metadata from the different corporate nodes and responds to queries about it that are issued by the different platform interfaces.

Input dependencies:	Output dependencies:
• Corporate Interface	• Smart Contract Executor • Business Support Services

Requirements addressed:

FR 1, FR 3, FR 5, FR 18, FR 21, FR 22, FR 23, FR 25, FR 26

ARS 11

C19	App Store

Apps are docker images that participant organizations can, after signing a contract with the provider, pull and execute in their premises. The app store is a docker registry that is coupled with the TRUSTS platform to ensure that (push or pull) operations are compliant with a contract. The app store must be connected to the authentication and authorization mechanisms of the platform.

Input dependencies:	Output dependencies:
• None	• ~~Trusted Connector~~ • Dataspace Connector

Requirements addressed:

FR 19, FR 24, FR 38, FR 39, FR 40, FR 41, FR 42, FR 43

ARS 12

C21	Vocabulary Services

All assets, organizations and components which are part of the TRUSTS platform are described as much as possible in terms of controlled vocabularies. These include, for example, taxonomies of keywords, types of assets, types of components, etc. These are maintained in this central location which provides other components with access to the list and definition of the terms in the vocabularies.

Input dependencies:	Output dependencies:
• Reverse Proxy	• Mapping Builder • Corporate Interface • Platform Interface

	• Broker + Metadata Storage
Requirements addressed: FR 19, FR 20, FR 21, FR 22, FR 24, FR 25 ARS 3	

C22	**Distributed Authorisation Component**
This is a component to support secure communication, authentication and authorization in TRUSTS infrastructure. It should support ACME server functionality. Can be provided by a combination of the IDS Dynamic Attribute Provisioning System (DAPS) implementation or the Small Step CA.	
Input dependencies: • ~~Trusted Connector~~ • ~~Automated Certificate Management Environment (ACME)~~ • Business Support Services • Reverse Proxy • Dataspace Connector • Internet Browser • Custom Application	**Output dependencies:** • ~~Trusted Connector~~ • Reverse Proxy • Dataspace Connector • Internet Browser • Custom Application
Requirements addressed: FR 38, FR 43, FR 44, NFR 6 ARS 16	

C23	**Automated Certificate Management Environment (ACME)**
This component enables automation for management of certificates in the TRUSTS infrastructure.	
Input dependencies: • Distributed Authorisation Component	**Output dependencies:** • ~~Trusted Connector~~ • Dataspace Connector • Distributed Authorisation Component
Requirements addressed: FR 38, FR 43, FR 44 ARS 16	

C24	Smart Contract Executor
When access to a dataset is registered in a node, this action can result in triggering smart contract rules. The Smart Contract Executor is the component responsible for receiving information about transactions like this, entering it into an appropriate logging mechanism, and performing the operations specified by a respective smart contract. The smart contract checks the ability of the consumer to get/use a dataset/service from a provider. The result then is documented with a logging mechanism and returned to the party which triggered the smart contract by telling whether the transaction was successful or not. The Smart Contract Executor shall be queried by the interested parties at any time to verify the current status of a contract or the sequence of operations pertaining to an asset they control. This component stores smart contracts which have customizable input parameters which can be derived from (1) asset metadata, (2) the offering, and (3) the contract participants to enable a given contract's reuse for different transactions.	
Input dependencies: • Dataspace Connector • Business Support Services • ~~Usage Control~~	**Output dependencies:** • Dataspace Connector • ~~Usage Control~~
Requirements addressed: FR 10, FR 11, FR 12, FR 13, FR 14, FR 15, FR 16, FR 17, FR 30, FR 38, FR 43 ARS 1, ARS 2	

In summary, the following components are considered part of the TRUSTS technical architecture. Removed components are marked as strike-through text:

- ~~C1 - Trusted Connector~~
- C1 - Dataspace Connector
- C2 - Dataflow Router
- C3 - Reverse Proxy
- C4 - Recommender
- C5 - Platform Interface
- C6 - Landing Page
- ~~C7 - Asset Consumer~~
- C8 - Notification Service
- C9 - Metadata Mapper
- ~~C10 - Usage Control~~
- C11 - Mapping Builder
- C12 - Corporate Interface
- ~~C13 - Services Consumer Adapter~~
- C14 - Data Exchange TRUSTS Component
- C15 - Data Exchange Client Component
- C16 - Registry of Data markets
- C17 - Business Support Services
- C18 - Broker + Metadata Storage
- C19 - App Store
- ~~C20 - Identity Provider + Key Distribution System~~
- C21 - Vocabulary Services
- C22 - Distributed Authorisation Component

- C23 - Automated Certificate Management Environment (ACME)
- C24 - Smart Contract Execution

3.2.1. Updates made to the component list since the previous version of this deliverable

Compared to the first version of this deliverable, the following changes have been made to the component list, based primarily on knowledge gained during the reporting period.

Asset Consumer Component. In the previous version of the architecture, the Asset Consumer was responsible for allowing a user to log in to a platform node X using their web browser and access an asset on node Y. The Asset Consumer component was removed because its functionalities are fulfilled by the Dataspace Connector (DSC) component. All static files (Datasets or configuration bundles) that a consumer acquires, will be served, from their perspective, by their own DSC. The routing will then be taken care of by this connector.

Usage Control Component. The Usage Control component was removed for the same reason as the Asset Consumer component, its functionalities are fulfilled by the DSC component. The Usage Control component was an ad-on to the connector implementation. It was intended to provide access and usage control to datasets and services. When a request to access an asset is received, this component should check if there is a contract between provider and consumer for the said asset, and if that contract is in a valid state. The DSC component is able to check whether access to a given asset should be granted.

Service Consumer Adapter Component. This component was used to offer services via the TRUSTS platform that are deployed by a participant in their Corporate Node. The Services Consumer Adapter component was removed, because its functionalities are fulfilled by the DSC component. When a client wishes to make use of a service acquired through TRUSTS, it can access endpoints in its (the consumer's) DSC. These endpoints will be routed accordingly (and DATs attached to the requests) onto the provider's DSC which will be able to check for access control policy fulfillment.

Identity Provider + Key Distribution Service Component. The identity provider + key distribution service has been removed because, for the moment, platform-wide user-level authorization has been descoped. What this entails is that granting of access to assets will be made to node, and authorization will be computed based on the DATs that acompain requests.

Trusted Connector Component. The Trusted Connector has been changed to the Dataspace Connector, since, as explained above, the Dataspace Connector provides functions that can be reused in TRUSTS.

3.2.2. Summary of connections between functional requirements and components

In the following we provide an overview showing which components need to address which functional requirements. In order to be able to recognize the differences to the previous deliverable directly, newly added components and requirements are marked in blue and removed components are crossed out.

Table 3: Summary of connections between functional requirements and components

ID of functional requirement (FR)	IDs of components, which address the requirements
FR 1	C5, C12, C18
FR 2	C14, C15, C16

FR 3	C5, C12, C18
FR 4	C5, C12, C14, C15, C16
FR 5	C5, C12, C18
FR 6	C4
FR 7	C4
FR 8	C4
FR 9	C4, C19
FR 10	C24
FR 11	C24
FR 12	C24
FR 13	C24
FR 14	C24
FR 15	C24
FR 16	C24
FR 17	C4, C24
FR 18	C18
FR 19	C21
FR 20	C21
FR 21	C18, C21
FR 22	C18, C21
FR 23	C9, C11, C14, C15, C18
FR 24	C19, C21
FR 25	C4, C5, C18, C21
FR 26	C18
FR 27	C1, C2, C5, ~~C7,~~ C12
FR 28	C1
FR 29	C1, C5, ~~C7, C10,~~ C12, C20
FR 30	C5, C12, C24

FR 31	C1, C5, C12
FR 32	C5, C12
~~FR 33~~	~~C5, C11, C17~~
FR 33A	C5, C11, C17
FR 33B	C5, C11, C17
FR 34	C5, C12
FR 35	C8
FR 36	C5, C12, C20
FR 37	C5, C12, C20
FR 38	C1, ~~C10,~~ C19, ~~C20,~~ C22, C23, C24
FR 39	C1, C19
FR 40	C1, C19
FR 41	C1, C19
FR 42	C1, C19
FR 43	C1, C19, ~~C20,~~ C22, C23, C24
FR 44	C1, C5, ~~C10,~~ C12, ~~C20,~~ C22, C23
NFR 1	C5, C12
NFR 2	C5, C12
NFR 3	C1, C5, C12
NFR 4	C5, C12
NFR 5	C5, C6, C12
NFR 6	C1, C2, C3, C8, C22

3.2.3. Summary of connections between architecture requirements and components

In the following we provide an overview showing which components address which architecture requirements. The differences to the previous deliverable can directly be recognized. Newly added components are marked in blue and removed components are marked as strike-through text.

Table 4: Summary of connections between architecture requirements and components

ID of architecture requirement (AR/ARS)	IDs of components, which address the requirements
ARS 1	C24
ARS 2	C1, C24
ARS 3	C9, C11, C21
ARS 4	C14, C15, C16
ARS 5	C3, C5, C6, C8, C12
ARS 6	~~C13,~~ C14
ARS 7	C1
ARS 8	C1, C2
ARS 9	C2
ARS 10	C2, ~~C13,~~ C14
ARS 11	C18
ARS 12	C1, C19
ARS 13	C1, ~~C7,~~ C8
ARS 14	~~C7,~~ C8, C9
ARS 15	C4
ARS 16	C1, C2, ~~C10, C20~~, C22, C23
ARS 17	C1
ARS 18	C1
ARS 19	C1
ARS 20	C1
ARS 21	C1
ARS 22	C2
ARS 23	C3, C5

4. Design considerations for the architecture of the TRUSTS platform

The technical architecture of the TRUSTS platform is influenced in three ways: (1) by the functional and (2) by the architectural requirements, but also (3) by the vision of the technical experts in the project.

The initial set of functional requirements were collected mainly from non-technical experts inside and outside of the project and are reported in deliverable D2.2. The updated set of functional requirements is described in D2.3. The architectural requirements, refined by the technical experts in the project, are listed in Chapter 3 “Technical requirements for the architecture”.

As already outlined in D2.6, the vision for the architecture is based on a consensus of the technical experts who are participating in the project. The vision is expressed in several design considerations for the technical architecture, which are described in this chapter.

In the first section of this chapter, the functional requirements are listed which are addressed by the architecture as a whole. Then the functional requirements which are based on the use cases from WP5 are described. This is followed by a description of two unique selling points of the TRUSTS architecture: the ability to handle services as part of a data marketplace, and the ability to handle portable applications as part of a data marketplace. Finally, it is described which components of the TRUSTS architecture enable the establishment of trust between participants of an instance of the TRUSTS platform.

4.1. Functional requirements which are addressed by the architecture as a whole

The use of the components listed here could be summarized as follows: Each organization has a set of components which, along with those deployed centrally by the TRUSTS operator, ensure that the assets it is offering are accessed in a trusted manner and in accordance with the contract it assigns. Likewise, consuming assets that are provided in the TRUSTS platform requires interaction with a set of components, so that privacy, data ownership, and contractual requirements can be enforced.

Apart from the FRs that are addressed by individual components as listed in Section 3.2.2, the FRs that have been of special interest when defining the architecture proposed here have remained the same as described in D2.6:

- **FR 28:** TRUSTS should be able to be deployed as a federation of distributed, interconnected and interoperable nodes.
- **FR 5**: The system should provide rich search mechanisms across all federated nodes for available datasets and services
- **FR 11**: The system should ensure the integrity and authenticity of the smart contracts transactions signed by its users

4.1.1. Functional requirements coming from the preparation for the use case trials

There are functional requirements that have been collected from further specification of the use-case scenarios treated in WP5 and already summarized and referenced in D2.6. The three use cases are to be executed in the frame of the TRUSTS project, as specified in deliverable D5.1. They make use of several features of the architecture proposed in this deliverable. The use cases are addressed by the way services and applications are traded and used in the TRUSTS platform. This is presented in the following according to the architectural changes made in this deliverable.

Handling of portable applications on the TRUSTS Platform

Applications are software components which will be executed in the premises of the consumer, without any data leaving their network. Their installation and execution by the consumer may be the subject of contractual agreements, and the TRUSTS platform will provide technical means to enforce some of their provisions.

As already described in D2.6, application providers onboard their applications through the Corporate Interface in their node. The process of onboarding involves providing the metadata for the application, the necessary files for its configuration and deployment, a machine-readable description of its interfaces, and the directions to an application image available within the provider's node. The Corporate Interface creates the appropriate bundle file. The metadata is included into the central node's catalog.

When a consumer searches in their own Corporate Interface or in the Platform Interface, they will see a description of the application and, upon entering a contractual agreement, will be able to download, on their premises, the corresponding bundle. Then, this bundle will be decompressed, and the application image will be, subject to verification of identity and contract, pulled from the appropriate image repository into the consumer's node. In contrast to the descriptions in D2.6, the messages mediating this pull operation will be inspected and acted upon by the provider's Dataspace Connector component.

Once the application image has been pulled, the configuration and deployment instructions contained in the bundle will be executed and the application will be made available from within the consumer's corporate network, to be used either directly by human users or by other, pre-existing applications.

Optionally, if the application exposes an HTTP interface adequately described (see below), the provider can choose to make their application accessible only through the Dataspace Connector deployed in the consumer's node. This would enable a fine-grained metering of application usage, and execution of smart contract operations upon every request.

Handling of services on the TRUSTS platform

Services are software components which are executed on the provider's premises. The access to these services by the consumer may be the subject of contractual agreements, and the TRUSTS platform will provide technical means to enforce some of their provisions.

As described in D2.6, services will be served by components installed by the provider according to their respective technical guidelines. This will allow, for example, pre-existing applications already deployed to be made available through TRUSTS, regardless of the operating system on which they are deployed, as long as they provide HTTP interfaces.

The service provider is responsible for providing a machine-readable description of the service. This file and the metadata required to describe the service form a bundle that must be assembled in the service provider's Corporate Interface as part of the onboarding process. The metadata is forwarded to the central metadata repository and can thus be found via the platform interface.

When a consumer acquires the service, they will receive in their corporate node the corresponding bundle. This bundle will be processed by the Dataflow Router in order to make the service accessible via the TRUSTS platform. In contrast to the description in deliverable D2.6, a user accesses in the consumer's network then the different endpoints of the service by making HTTP requests to the Dataspace Connector deployed in its node, not to the Service Consumer Adapter. According to its integrated contract mechanisms, the Dataspace Connector verifies authorization for use. If deemed adequate, the request is forwarded to the component in the provider's infrastructure that actually provides the service.

4.2. The role of the components in enabling trust between participants

The connector component enables verification of the origin of all messages received. This verification ensures that each request received comes from the TRUSTS platform and the originating node and user can be reliably identified. The contractual validity of a request can be evaluated by the provider with the architecture presented in this deliverable. This is described in more detail below and, compared to the description in D2.6, adapted according to the revised architecture. As described in Section 3.2 the Dataspace Connector offers rudimentary negotiation mechanisms.

1. Application providers will transfer applications only after the requesting messages have been validated by the Smart Contract Executor / Dataspace Connector. Thus, the provider can rest assured that only contractually valid transfers of their applications take place.
2. If the application is to be made accessible only through the consumer's Dataspace Connector (only possible in the case of applications providing adequately described HTTP interfaces), then the application provider can request from the smart contract executor a detailed log of all access to the application. Importantly, the application provider must develop in their application the security mechanism to ensure that only requests which are accompanied by an adequately signed authorization token are served. In this case, it is possible to utilize fine-grained contracts on an application, for example, based on the number of requests.
3. Services exposed by a provider will be configured to only accept requests being forwarded to them by the node's connector component. The node's connector component will evaluate the request against the contract negotiations. Thus, the provider can rest assured that only contractually valid requests are served.
4. The requests done to a service are logged also by the consumer's node. This disallows incorrect claims by the provider regarding access to the service, as the smart contract executor holds records from both parties regarding access operations.
5. Dataset providers, likewise, can rest assured that all requests served are contractually valid, since only then will they be forwarded to any serving component (e.g., the Corporate Interface).
6. All signatures and tokens that have been described will be forwarded to the destination component, thus enabling any provider to check and log transactions on their own terms. Likewise, all certificates used to issue these signatures and tokens will be known to all participants, as part of the organization onboarding process.

Importantly, the metadata of all assets will include signatures (hashes) of all traded assets (application images, service descriptions, datasets), which will allow the consumer to rest assured that the received asset has not been tampered with. These signatures can, furthermore, be compared directly with those included in the provider's node catalog, thus reducing the need for trusts in the TRUSTS operator.

5. Conclusion

This deliverable summarizes the activities in task 2.4. “Architecture design and technical specifications” and is the second and final version of this deliverable. It documents the blueprint for the technical results of the TRUSTS project. The technical specifications provide the details which are required by technical experts in order to instantiate the platform infrastructure and build on top of it or to extend it with their own components, services and applications.

The second version of this report focused on the iterative refinement of the technical architecture of the TRUSTS platform based on the knowledge gained by the project’s technical partners during the reporting period and on feedback from the use case partners and from the non-technical project partners. The project partners with a technical perspective have refined the architecture and document additional technical specifications based on their experiences gained during the implementation of the platform.

A challenge for the design of the technical architecture of the TRUSTS platform was that the architecture must consider the requirements and priorities of many different stakeholders, both inside and outside the project. To address this challenge, the architectural requirements were collected from the initiatives the TRUSTS platform is based on, namely Data Market Austria and International Data Spaces. In addition, the architectural requirements from the Gaia-X initiative were collected, as we expect that the future compatibility with Gaia-X is of strategic importance to the TRUSTS platform. We further expect Gaia-X to set important impulses for the data economy in Europe by e.g., communicating with important groups of stakeholders to set the agenda and by setting standards for technical and organizational issues, such as certification.

The architectural requirements collected and grouped by areas of concern in the first version of this deliverable have been refined in this deliverable. In total, 59 architectural requirements were collected, grouped into 23 architecture requirements summaries.

In an iterative process and based on the collected architectural requirements, the technical architecture, introduced in the first version of this deliverable, was refined. This deliverable gives an overview of the refined technical architecture. In total, the proposed technical architecture of the TRUSTS platform consists of 20 components, needed to address the collected requirements. It is summarized in tables how the components are connected to the functional requirements and the architecture requirement summaries.

To document the aspects of the architecture which are represented in the interplay of multiple components, design considerations are described, and an explanation is provided of how trust can be enabled between different participants using the TRUSTS platform. Facing the question of why anyone should trust TRUSTS makes clear that security- and privacy-by-design is a high-level characteristic that is of strategic importance and must be addressed by the platform.

The technical architecture provides the technical partners of the project the theoretical and conceptual foundation of how to instantiate the TRUSTS platform. The architecture is evolved by employing an idea from agile development, called Minimum Viable Product (MVP) [4, 5]. A current status report of the platform implementation is given in D3.10 “Status of Platform Implementation II”.

To ensure Gaia-X compliance, the TRUSTS platform will be scalable and extensible by different systems. This is achieved by providing technical interfaces and allowing participants to extend the platform with their own components, services, and applications.

The results of this task will also provide input for standardization experts and can be used as a starting point for standardization efforts with relevant external stakeholder organizations. This is an additional argument that the work on task 2.4 contributes to the sustainable success of the TRUSTS project.

TRUSTS aims to be a federated platform. By considering the architecture requirements related to the interoperability of data marketplaces as well as enabling information sharing between semi-autonomous de-centrally organized applications, the proposed architecture provides the technical basis to achieve this goal.

If additions or changes to the TRUSTS architecture should become necessary in the third year of the project, then these changes will be included as part of deliverable D3.11 “Platform Status Report III”. Deliverable D3.11 is scheduled to be published in the last month of the TRUSTS project duration.

In summary, the designed and iteratively refined technical architecture of the TRUSTS platform attempts to address all the architectural requirements described in Chapter 3 of this deliverable. The technical architecture follows all the design principles of DMA, IDS, and Gaia-X that have been listed as architectural requirements in this deliverable, as well as the architectural requirements formulated by the technical partners of the project.

6. References

[1] M. Traub, et al., "Broker and Assessment Technology Specification and Development Road", Data Market Austria, May 2017.

[2] B. Otto, et al., "Reference Architecture Model Version 3.0", International Data Spaces Association, April 2019.

[3] G. Eggers, et al. "GAIA-X Technical Architecture", Federal Ministry for Economic Affairs and Energy (BMWi), June 2020. Accessed via: https://www.data-infrastructure.eu/GAIAX/Redaktion/EN/Publications/gaia-x-technical-architecture.pdf?__blob=publicationFile&v=5

[4] V. Lenarduzzi, D. Taibi, "MVP Explained: A Systematic Mapping Study on the Definitions of Minimal Viable Product", Euromicro SEAA, 2016.

[5] J. Münch, et al. "Creating minimum viable products in industry-academia collaborations." International Conference on Lean Enterprise Software and Systems. Springer, Berlin, Heidelberg, 2013.