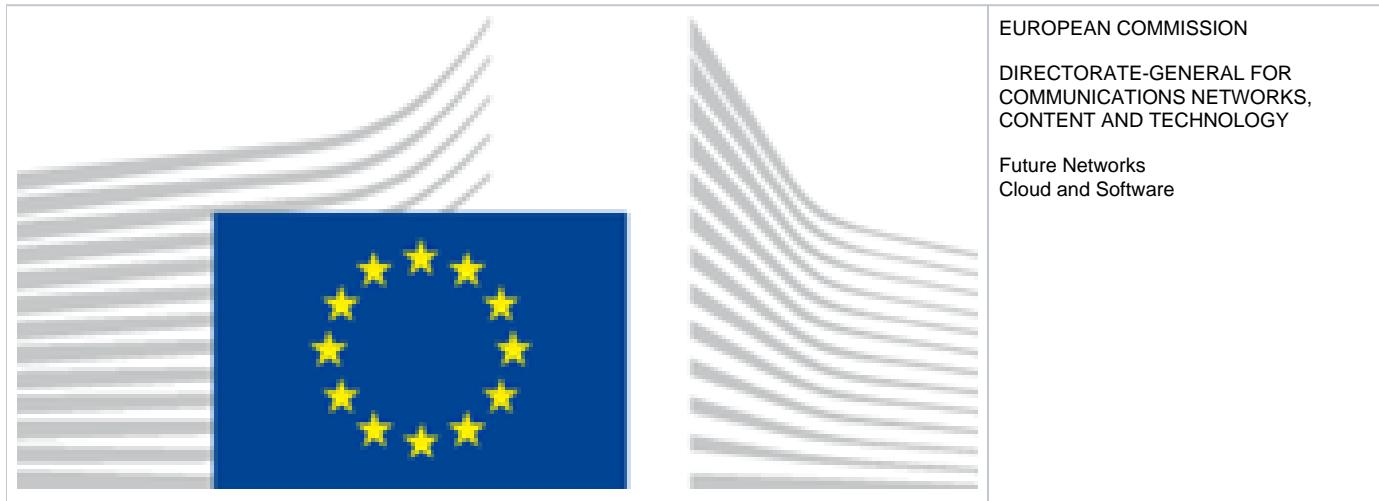


Functional and Technical Architecture Specifications



EUROPEAN COMMISSION

DIRECTORATE-GENERAL FOR
COMMUNICATIONS NETWORKS,
CONTENT AND TECHNOLOGYFuture Networks
Cloud and Software

D1.3.1.2 Functional and Technical Architecture Specifications

SC1

Document Control Information

Settings	Value
Document Title:	D1.3.1.2 Functional and Technical Architecture Specifications
Project Title:	SC1
Document Author:	Sovereign-X
Doc. Version:	0.2
Sensitivity:	Limited
Date:	13 déc. 2024

Document history:

The Document Author is authorised to make the following types of changes to the document without requiring that the document be re-approved:

- Editorial, formatting, and spelling
- Clarification

To request a change to this document, contact the Document Author or Owner.

Changes to this document are summarised in the following table in reverse chronological order (latest version first).

Revision	Date	Created by	Short Description of Changes
0.2	13 déc. 2024	Sovereign-X	SfR M9 second iteration
0.1	30 sept. 2024	Sovereign-X	SfR M9 first Iteration

Configuration Management: Document Location

The latest version of this controlled document is stored in [Functional and Technical Architecture Specifications](#).

Table of Contents

Introduction

- Scope of this document
- Target Audience
- Changes with respect to the previous version

Simpl-Open High Level Overview

- Simpl-Open Description
- Data Space Concepts
- High Level Architecture
- Architecture Framework

Simpl-Open Business Architecture

- Actors
- Simpl-Open Functional Architecture
- List of Business Processes and Functional Requirements

Simpl-Open Application Architecture

- Application Components Views
- Interfaces
- Traceability from the functional architecture

Simpl-Open Data Architecture

- Open-Source Components Data Model
- Custom Components Data Model

Simpl-Open Technology Architecture

- Technology Components Views
- Technology Deployment View
- Technology Open-Source Products
- Detailed Technical Specifications

DevSecOps Approach

- Overview
- Planning and Design of Clusters
- Cluster Provisioning and Setup
- Security and Access Control
- Continuous Deployment and GitOps
- Testing
- Monitoring and Logging
- Backup and restore
- Deprovision of Environments

Annexes

- Annex 1 - Glossary of Terms
- Annex 2 - Mapping between functional requirements and components
- Annex 3 - Non-Functional Requirements
- Annex 4 - Architecture Patterns
- Annex 5 - Architecture building blocks

Introduction

Scope of this document

The purpose of this document is to describe the functional and technical architecture of Simpl-Open, following the approach further described in the Architecture Approach section. It includes the following content (non-exhaustive list):

- A high-level overview of Simpl-Open architecture vision;
- A description of Architecture Principles, Assumptions and Decisions that drive the Simpl-Open architecture;
- A description of the Simpl-Open architecture from a business, application, data and technology perspective, each of them described using appropriate diagrams (BPMN, ArchiMate, UML);
- A description of the Simpl-Open security architecture.

Target Audience

The intended audience of this document comprises people involved in the architecture, design, integration, testing and maintenance of Simpl-Open.

It mainly targets architects, but can also be helpful for developers, testers and other stakeholders involved in Simpl-Open, as well as stakeholders involved in Simpl-Live or other data spaces interested in integration of Simpl-Open.

Changes with respect to the previous version

The current version is the first version of this document.

Simpl-Open High Level Overview

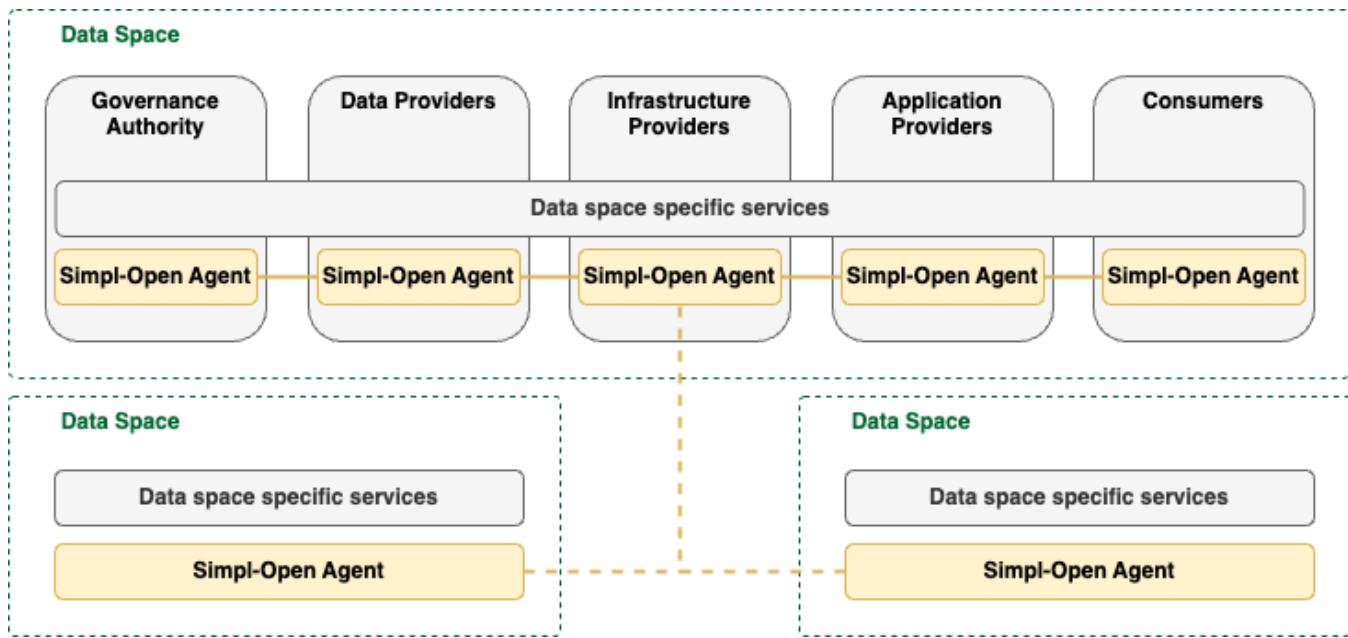
Simpl-Open Description

With the ongoing exponential growth of data, there is a pressing need within the European Union to provide access to resilient and competitive data storage and processing capacities for both the private and the public sector. In particular, the European Commission aims to address the need for more data sharing and decentralised data processing closer to the user (at the edge). It is also critical to deploy EU data services in the public and private sectors to grant Europe a leading status as a data-driven society and improve data usage within the European Union. The data services of various organisations within the same industry sector should be abstracted into sector-specific data spaces. This could bring several benefits, such as greater productivity, improvements in health and well-being, adaptation to environment and climate change, transparent governance and convenient public services.

To support the above-mentioned objectives, the European Commission is creating an open source, multi-vendor, large-scale, modular and interoperable middleware called "Simpl-Open". Simpl-Open will be the basis for a European Cloud Federation enabling the operation and interconnection within and in between various European data spaces and the safe migration of the users to the cloud.

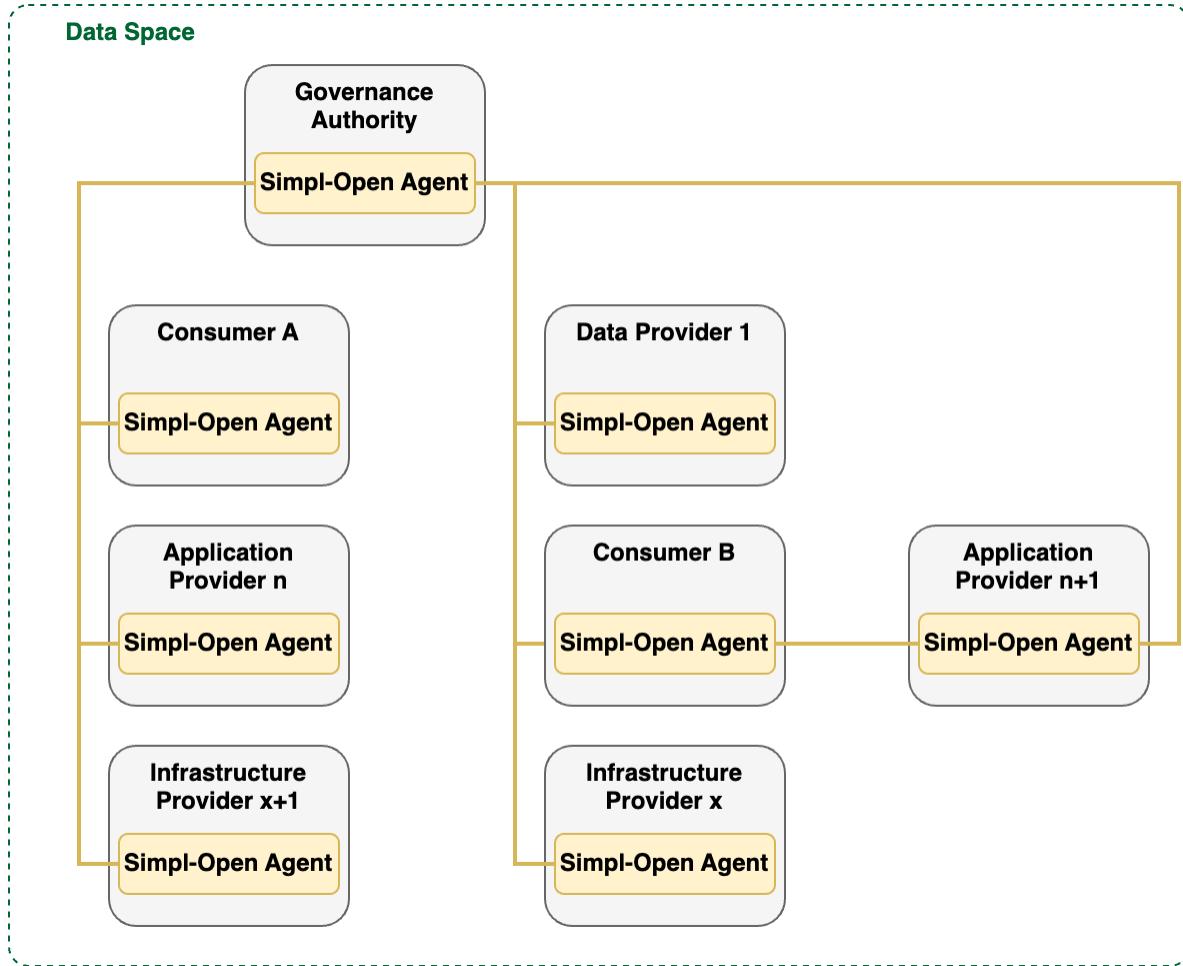
Simpl-Open will federate data, application, and infrastructure across the European Union with secure, resilient, energy efficient, and accessible cloud-to-edge capabilities. It will allow EU stakeholders to pool together resources to create more business value, increase resource usage efficiency, and reduce costs and duplication of efforts. Simpl-Open considers both the public sector as well as EU business as core stakeholders. Using the features provided by Simpl-Open, an open marketplace for EU resources will be created that enables energy-efficient reuse of efforts achieved by other EU participants.

The following figure displays how the architecture vision of Simpl-Open maps the five actor groups (see definition in Actors section) and different data spaces:



At the core of data spaces lie the five types of actors that Simpl-Open considers. These actors are a symbolic representation of a distributed network of cooperating parties in an open ecosystem. Simpl-Open, represented by the Simpl-Open Agent, spans across these actors, enabling asset sharing between them. It provides common services on which data spaces can be built. Simpl-Open stays agnostic to the specifics of a particular data space, allowing additional data space specific services to be added on top of Simpl-Open. This added layer can, for example, contain standards on data representation, enforce common quality certifications, or define peer review rules to assess data quality. The data space specific services tailor the ecosystem beyond simple sharing of assets; they make sure those assets become valuable to participants.

Simpl-Open does not only aim to be used to build data spaces, it also creates interoperability between data spaces. As multiple data spaces incorporate Simpl-Open, data spaces become more connected. This enables services to cross the boundaries of specific data spaces. Such services will initially be more limited, as Simpl-Open cannot capture the details of all different data spaces. It will be up to the user to deal with the specifics of each data space in interpreting the assets that it obtains. To make this illustrative view more tangible, the following figure presents an example of how a set of distributed actors might interconnect to form a data space. It is important to note that this figure displays one possible scenario of many possible ways different participants might interact. The number of participants in a data space or the number of stakeholders behind a single actor is only limited by its technical feasibility. This implies that large numbers of participants and stakeholders can interact simultaneously. The Simpl-Open Agent in the figure serves as an abstract component that actors need to deploy to become part of the data space.



It is important to note that each of these displayed actors are an abstraction to the internal systems of one or more stakeholders. The deployment of Simpl-Open in a data space can have various degrees of granularity. The stakeholder behind an actor can be an individual user that has the capabilities to deploy a Simpl-Open Agent, or can be an entire data sharing initiative on its own. It is up to the data space governance authority to decide how Simpl-Open best provides value, and what level of granularity of the deployment fits best.

Data Space Concepts

This section defines general concepts that are necessary for a good understanding of the Simpl-Open documentation and ecosystem in general.

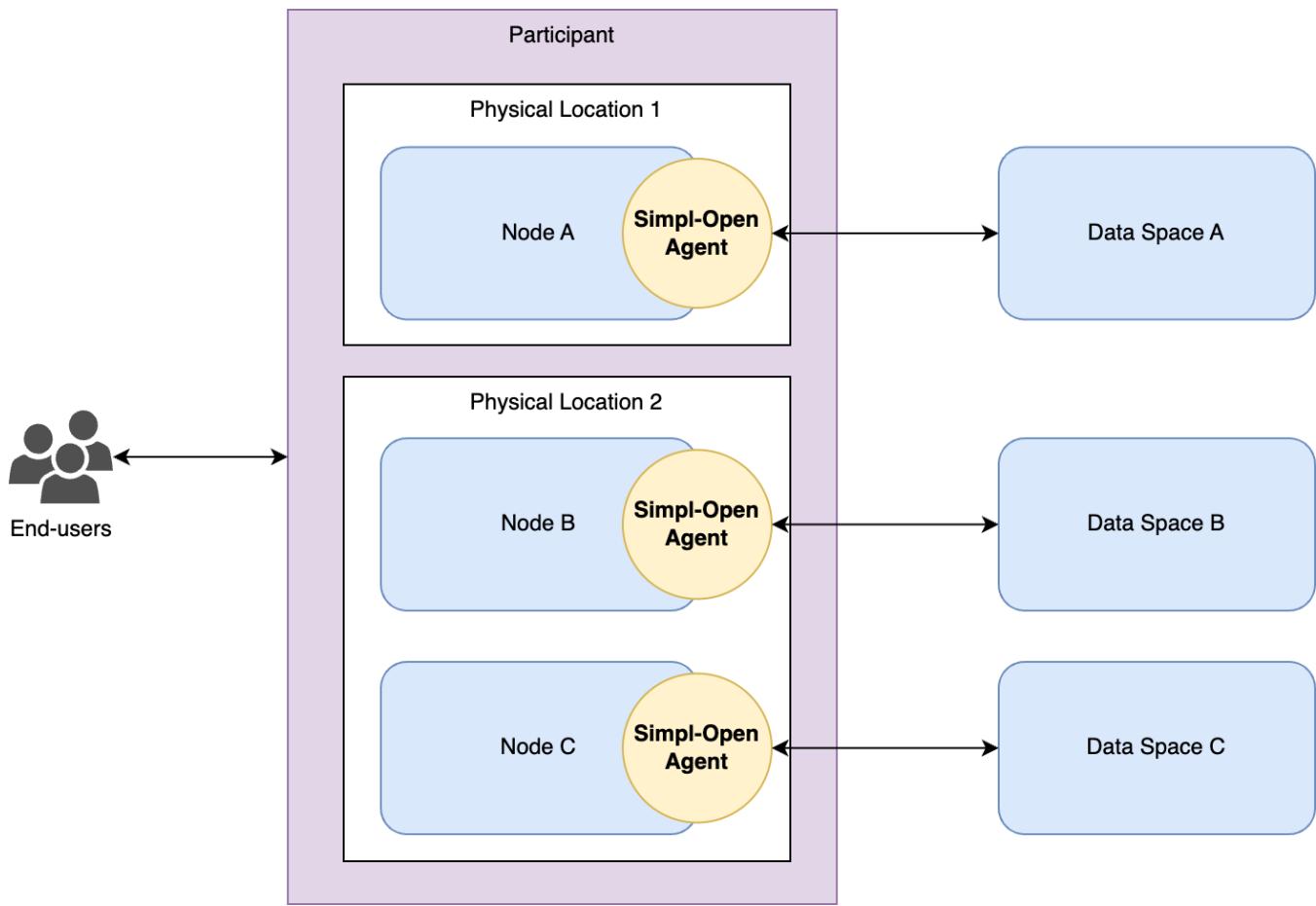
Actors and Data Space Deployment

As described above, a data space consists of **actors** (individuals or entities) who need to interact with each other. In Simpl-Open, it is assumed that individuals (called **end-users**) are always part of an entity (called **participant**). A participant can operate one or multiple **node(s)** which represent a distinct and/or isolated set of IT resources and can participate to different data spaces with various participant **roles** (including Data Provider, Application Provider, Consumer, Infrastructure Provider and - exactly one - Governance Authority). Nodes can also be spread across physical locations.

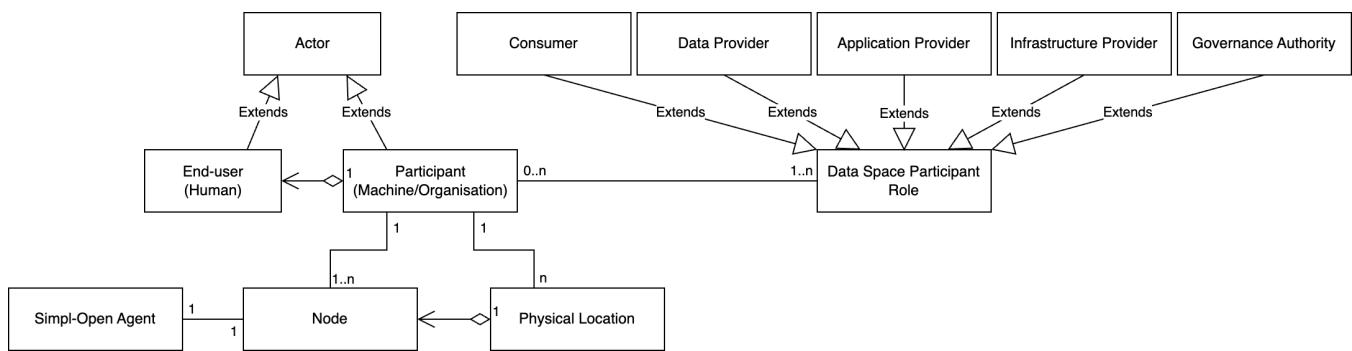
Example: a university (= **participant**) which embodies students, researchers or accountants (= **end-users**). The university has a dedicated network (with connected IT resources) for its sciences department (= the sciences **node**) hosted in Paris (= **physical location**) and a dedicated network for its economy department (= the economy **node**) hosted in Rome (= **physical location**). The sciences department might want to offer the results of its researches (= data provider **role**) to a science-related data space while the economy department might want to consume data (= consumer **role**) from an economy-related data space.

Simpl-Open Agent is a middleware, to be deployed on each node, acting as a local gateway for secure communication within a data space.

The following diagram illustrates this deployment view:



The following diagram translates the above deployment view into a data domain model to better understand the relationship and cardinality between the different entities.



Data space Participant: Tier I and Tier II

Only the HL concepts of Tier I and Tier II are presented here as it is required for a good understanding of the next sections. More details can be found in the Simpl-Open Application and Technology Architecture, especially related to Domain 1.

Identification, authentication and authorisation are of paramount importance within a data space.

The identification must be supported by the governance authority. This authority is in charge of reviewing the identity details of organisations that want to participate in the data space. If the authority approves the participation of an organisation in the data space, it provides a proof of identification that the organisation installs in its Simpl-Open Agent. With this proof, the participant authenticates itself to other data space participants and other participants define authorisation rules based on the verifiable identity.

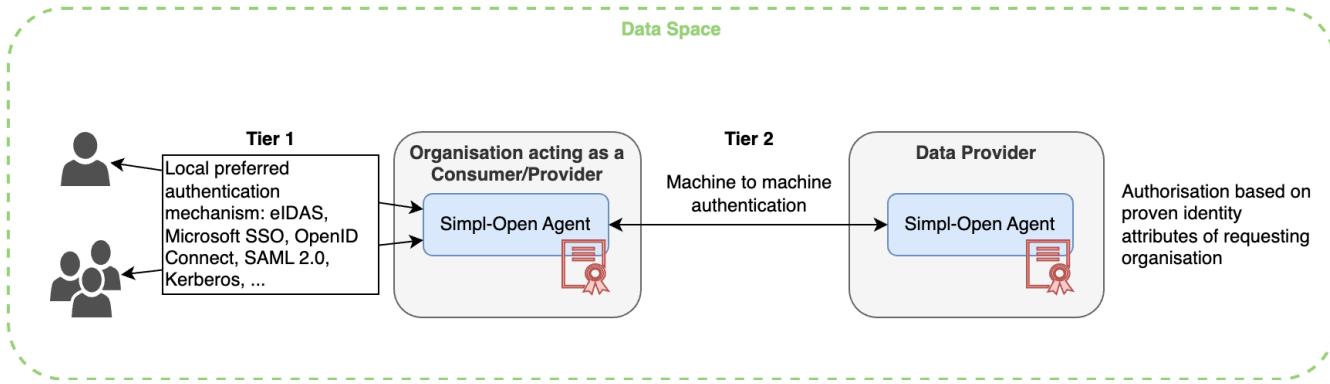
The identification system plays an important role in two functionalities of Simpl-Open:

1. Establish secure communication channels;
2. Provide the information on which participants can base themselves to define access and usage policies.

To keep the identification system manageable in a large-scale environment, the identification is split into two tiers:

1. The **first tier** manages the identification, authentication and authorisation of the organisation's members (humans or machines) to use the Simpl-Open Agent of their organisation;
2. The **second tier** identifies and authenticates the organisation as a whole in the Simpl network.

The figure below depicts this two-tier approach:



In the **first tier**, the Simpl-Open Agent connects to the preferred IAA system of the organisation: EU Login, eID, Microsoft AD, OpenID Connect, etc. This mechanism is already well established and not unique to Simpl.

The **second tier** involves the machine-to-machine authentication and identification of an organisation in the Simpl network. Each organisation holds an "Identity" file to support the identification, authentication and authorisation of the organisation in the data space. Recalling the two functionalities that the IAA supports, the necessary content of this Identity file becomes apparent:

- For the **establishment of a secure communication channel between participants (1)**, the Identity file should contain a proof of the organisation's public key. Each data space participant will create a cryptographic public/private keypair that is used in the asynchronous authentication mechanisms needed to establish a communication channel. An example of how such a secure communication channel can be established is the well-known TLS/SSL protocol. The Identity file associates the public key of an organisation to its identity. Proving the identity of the organisation then becomes proving the possession of the private key that belongs to the respective public key. This way, the organisation can be authenticated in the network and a secure communication channel can be established.
- **Access control and authorisation by providers (2)** can be performed based on custom identity attributes of an organisation. Examples of such attributes are the organisation name, geographical location, whether it is a private or public institution, etc. Based on these attributes, providers can define access and usage policies for their resources. For example, a provider can open a resource to all public institutions, or to all participants from a specific Member State. On the other hand, the access control policies can be more stringent and access is only allowed for a specific organisation. The Identity file proves the attributes of an organisation, and, as such, ensures the trust on which a provider can rely to enforce their access control.

Access Control & Trust

How IAA works at a high level:

1. **Roles** are used to enforce **RBAC** (role-based access control) to end users that access SIMPL Open functionalities in tier 1;
2. **Identity Attributes** are used to enforce **ABAC** (attributes-based access control) in the agent-to-agent (node-to-node) communication in tier 2;
3. **Assignable Identity Attributes** are used to be assigned to **Roles** enabling end users belonging to those roles to act on behalf of the Participant in a certain context.

Second Tier IAA - X.509 certificates with dynamic attribute provisioning

For clarification purposes, next an example is shown on how Tier II will work in practice:

1.1 - John Doe logs into the **Consumer** IAA Tier 1 System.

1.2 - IAA Tier 1 System retrieves user roles from Simpl-Open Agent **User roles** module and assign to **John Doe** the rights to access the data space functionality through **Consumer**'s Simpl-Open Agent, from now on all actions performed by **John Doe** are actually performed by the Simpl-Open Agent of **Consumer** which in turn interacts with the other Simpl-Open Agents (Provider and/or Data space built-in capabilities).

2.1 - **John Doe** makes the infrastructure request to the **Provider** Simpl-Open Agent that validates it against the **Access Control and Trust** capability.

2.2 - **Provider** and **Consumer** authenticate each other using the mutual x509 TLS Authentication.

2.3 - **Provider** and **Consumer** verify validity of the x509 certificate through the **Identity provider federation**.

2.4 - Provider enforces access control policy based on embedded identity attributes and authorise **Consumer** Simpl-Open Agent.

2.5 - Consumer requests his own identity attributes ephemeral proof to **Identity provider federation**.

2.6 - Identity provider federation responds to **Consumer** ephemeral proof with identity attributes.

2.7 - Consumer sends ephemeral proof with identity attributes to **Provider**.

2.8 - Provider checks and validates the ephemeral proof, then enforces access control policy based on embedded identity attributes and authorises **Consumer** Simpl-Open Agent.

3.1 - Once verification against **Access Control and Trust** is successfully passed, the **Provider** uses his own **Infrastructure/User data services** module to fulfill received requests:

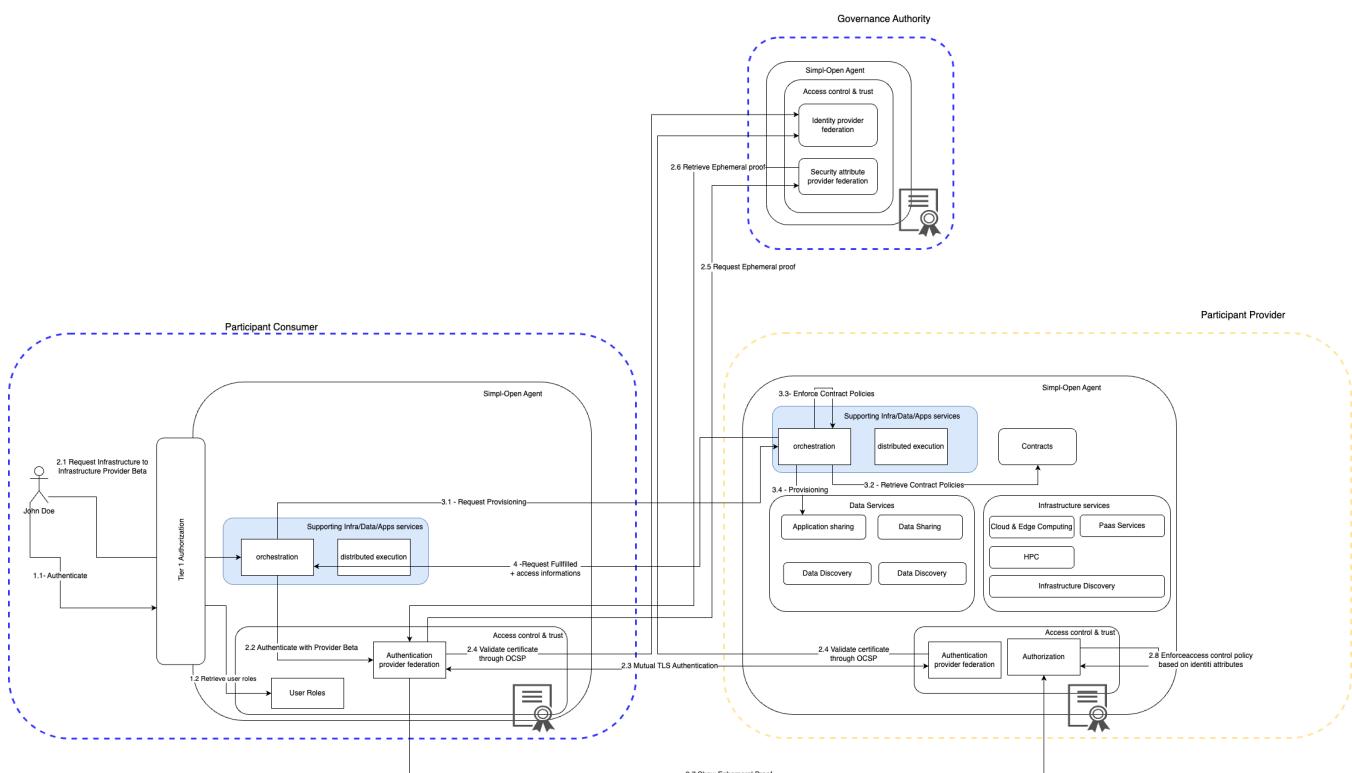
3.2 - Provider checks the policies querying **Contracts** module.

3.3 - Provider enforces retrieved contract policies.

3.4 - Activate the Provisioning module to fulfil the requested resource.

4 - Provider returns an affirmative response to **Consumer** request.

The process explained above is depicted below :



Connector

The [IDSA Reference Architecture Model](#) defines the connector as being the technical core component required for a participant to join a data space.

DSSC defines the connector as a technical software component that is run by (or on behalf of) a participant and that provides connectivity with similar components run by (or on behalf of) other participants, to enable the secure and trusted sharing of data.

A connector can provide more functionality than what is strictly related to connectivity. The connector can offer technical modules that implement data interoperability functions, authentication interfacing with trust services and authorization, resource description, contract negotiation, etc.

DSSC uses “participant agent services” as the broader term to define these services.

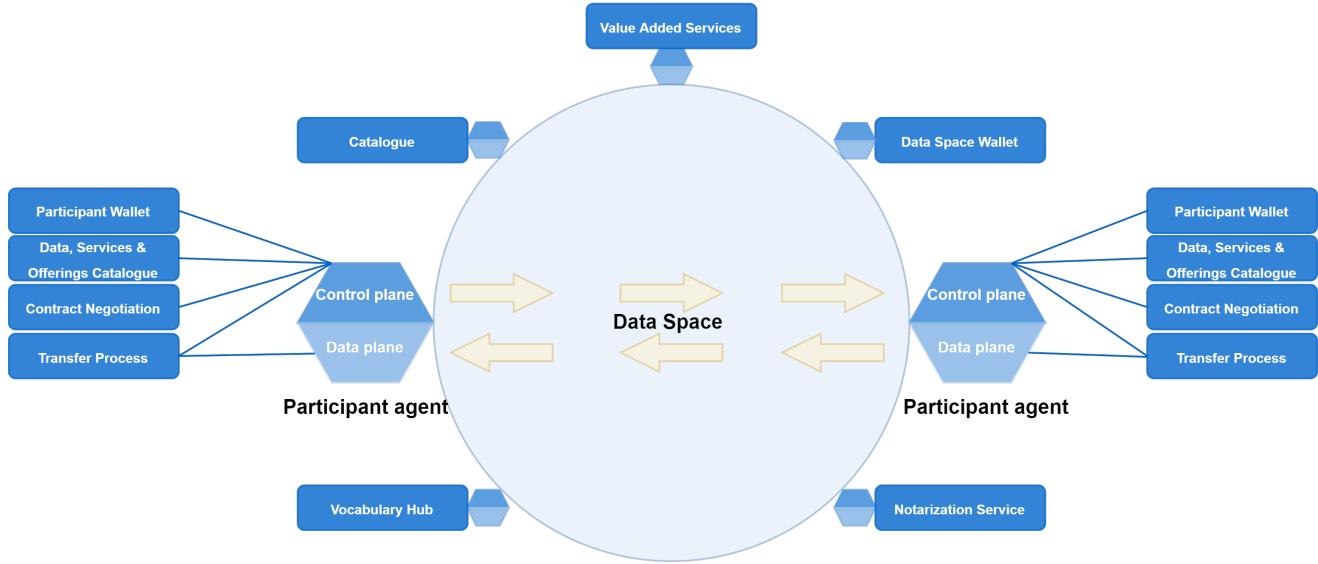
DSSC also distinguishes the 2 major components that make up a connector:

- The **control plane** is responsible for deciding how data is managed, routed, and processed. For example: the control plane handles the identification of users and the handling of access and usage policies.
- The **data plane** handles the actual exchange of data.

This implies that the control plane by its nature can be standardized to a high-level, while the data plane is likely to be different for each data space (as different types and sorts of data exchange take place in each data space).

The data plane needs to be integrated with the control plane to ensure that it can work with the necessary control mechanisms.

DSSC identifies the different categories of components within a data space, making a distinction between the (1) participant agent (= connector in DSSC vocabulary) and (2) shared services:



Within the control plane, several components can be identified:

- A Participant Wallet: providing participants with the ability to store and exchange identities and other attestations. For instance in the form of Verifiable Credentials.
- A Data, Services & Offerings Catalogue: providing participants with the ability to share (on a technical level) the data, services and offerings which are provided through the data plane.
- Components for Contract Negotiation: providing participants with the ability to share data access and usage policies with others in the data space and to enforce these on the data plane. For instance: to create an authorization registry, which - based on policies - can determine who gets access to a certain data set or service.

On the data plane there is the actual transfer process. As indicated before, the data plane is likely to be highly application specific. It should however work in conjunction with the control plane, e.g. to ensure that no data sharing can start before certain conditions are met (identification, contract negotiation, etc.).

Note that components of the connector can have different granularities. They can be conceived as an integrated component, but they can also consist of multiple (packaged) components (e.g. with a separated, but linked, component for Participant Wallet).

Concretely for Simpl-Open, a connector is used to implement the 3 parts of the IDSA Data Space Protocol :

1. Publication and request of catalogue items - mapping to Data, Services & Offerings Catalogue component of the control plane ;
2. Contract negotiation - mapping to Contract Negotiation component of the control plane;
3. Data transfer process - mapping to the Data Plane.

The control plane of the connector is also used as orchestrator between the 3 parts.

Current implementations of connectors do not cover all the needs envisioned in Simpl-Open and therefore extension points are planned, for instance, to cover the infrastructure provisioning.

High Level Architecture

This section elaborates on the High Level Architecture of Simpl-Open. It presents the capabilities of Simpl-Open and the building blocks that support these capabilities. It is important to remark that the high level architecture lays out the capabilities of Simpl-Open as a whole. How these capabilities are realised is then described in the following sections of the document.

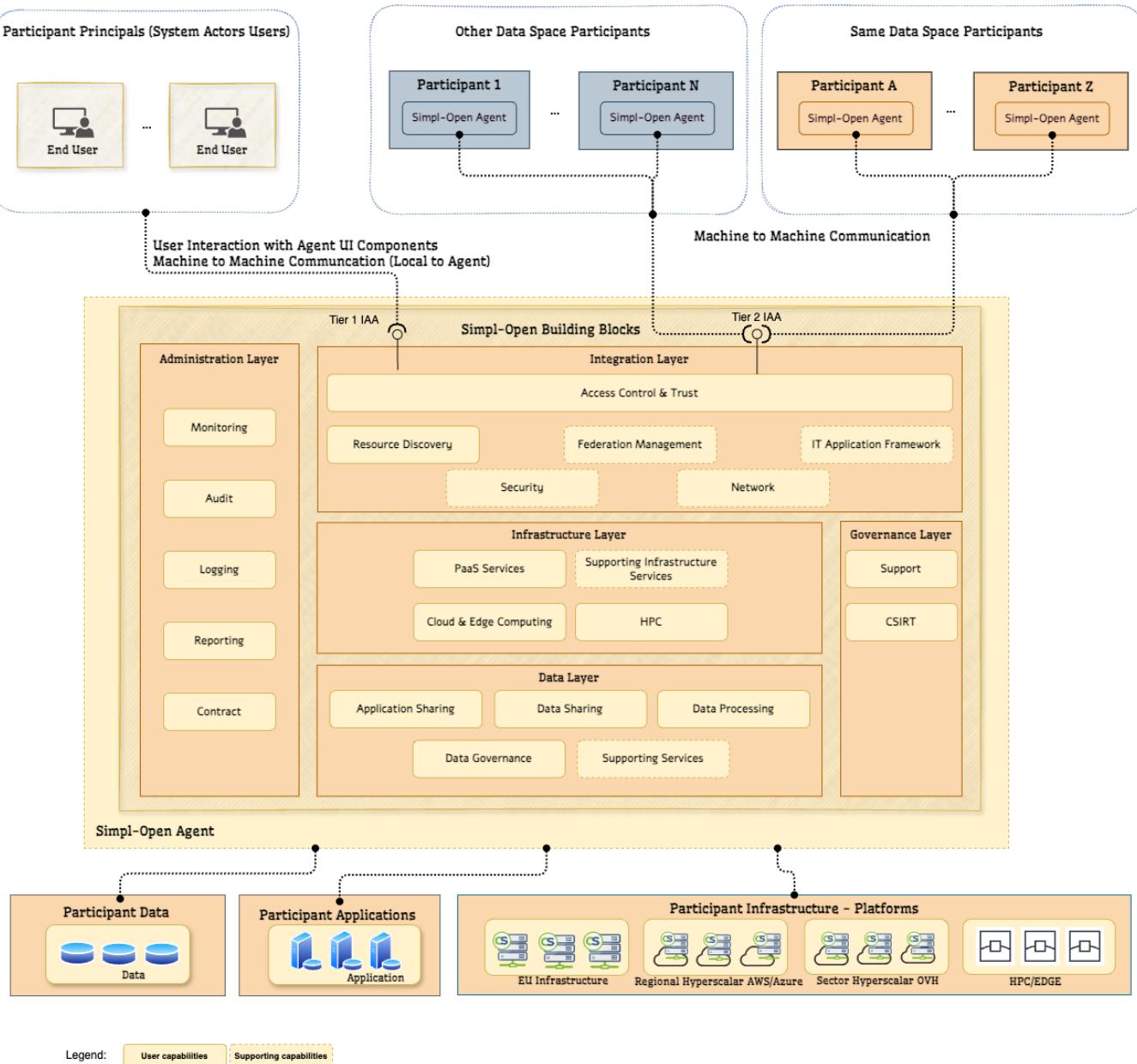
i The concepts described in this section have been, for a large part, already developed in the [Architecture Vision Document](#) of the Simpl Preparatory Study. They are taken over in this document and updated/complemented where needed to stay up-to-date with the current developments of Simpl-Open.

Five architectural layers describe Simpl-Open: the integration layer, the data layer, the infrastructure layer, the administration layer and the governance layer.

1. The **integration layer** contains the capabilities that enable participants to integrate with each other in a secured and trusted manner. This is required for the well-functioning of a data space integrating Simpl-Open. These capabilities regard security, access control and trust, resource discovery, federation management, network and IT application framework.
2. The **data layer** encompasses the capabilities to enable the exchange of data resources and applications. Simpl-Open offers data consumers the means to access different types of data from different providers, enabling interoperability between providers and consumers. The layer contains the capabilities to share as well as manage data and applications, and perform basic analysis.
3. The **infrastructure layer** has similar responsibilities for managing infrastructure resources. This layer allows Simpl-Open to connect to third-party infrastructure resources, such that end users can, for example, execute applications on them. Simpl-Open does not provide infrastructure itself, but allows infrastructure providers to open up internal infrastructure towards data space participants. Both elemental computing and storage resources (e.g. virtual machines, file system storage) as well as PaaS services (e.g. databases, AI hardware) can be provided. The infrastructure layer allows end users to utilize, and manage infrastructure resources offered by infrastructure providers.
4. The **administration layer** vertically spans the data, infrastructure and integration layers. It provides supporting capabilities for the well-functioning of the other layers, as well as for the administration of Simpl-Open. The administration layer allows actors to operate their components in the data space.
5. Last, the **governance layer** addresses transversal capabilities that apply to all the aforementioned layers as it provides contingency measures against issues the participants might find while using Simpl-Open. The two main capabilities in this layer are composed of human task forces that either support the participants in finding solutions to technical problems or trigger a response action against security threats in the form of a Computer Security Incident Response Team (CSIRT).

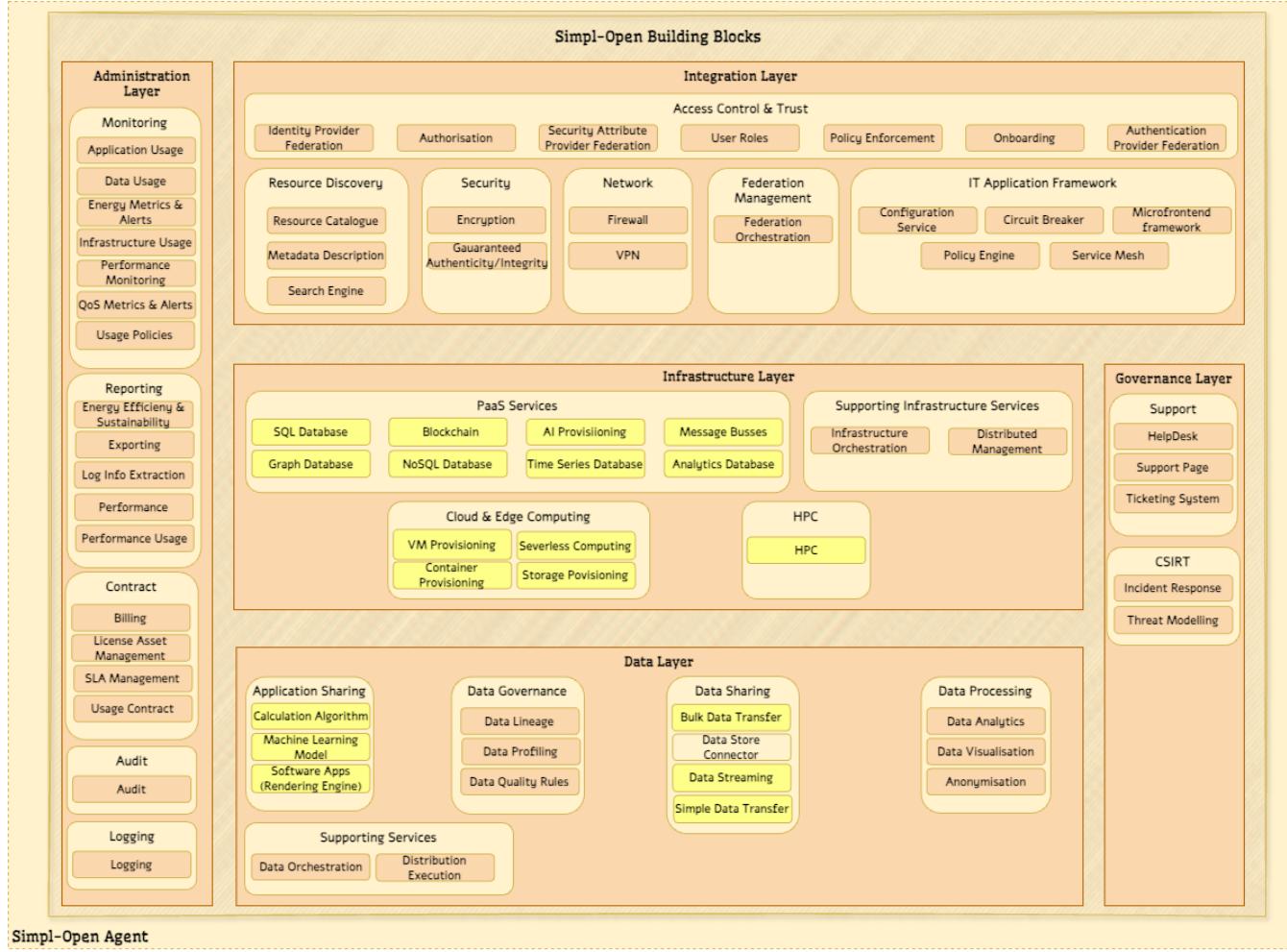
Each of these five layers is further detailed in the following sub-sections.

Two types of capabilities are distinguished: user capabilities and supporting capabilities. The user capabilities represent the capabilities that are offered to the users of Simpl-Open. Not all users will need the provided capabilities of Simpl-Open. Organisations acting as a data provider will, for example, not require the user capabilities of the infrastructure layer. The supporting capabilities enable a correct functioning of the user capabilities. They do not add immediate value to Simpl-Open actors, but run in the background to support the user capabilities. Simpl-Open actors also do not directly interact with the supporting capabilities.



The capabilities (depicted with light yellow boxes) group a set of building blocks (depicted with light orange or yellow boxes) that are required for Simpl-Open to provide that capability to users. This vocabulary is aligned to Togaf methodology.

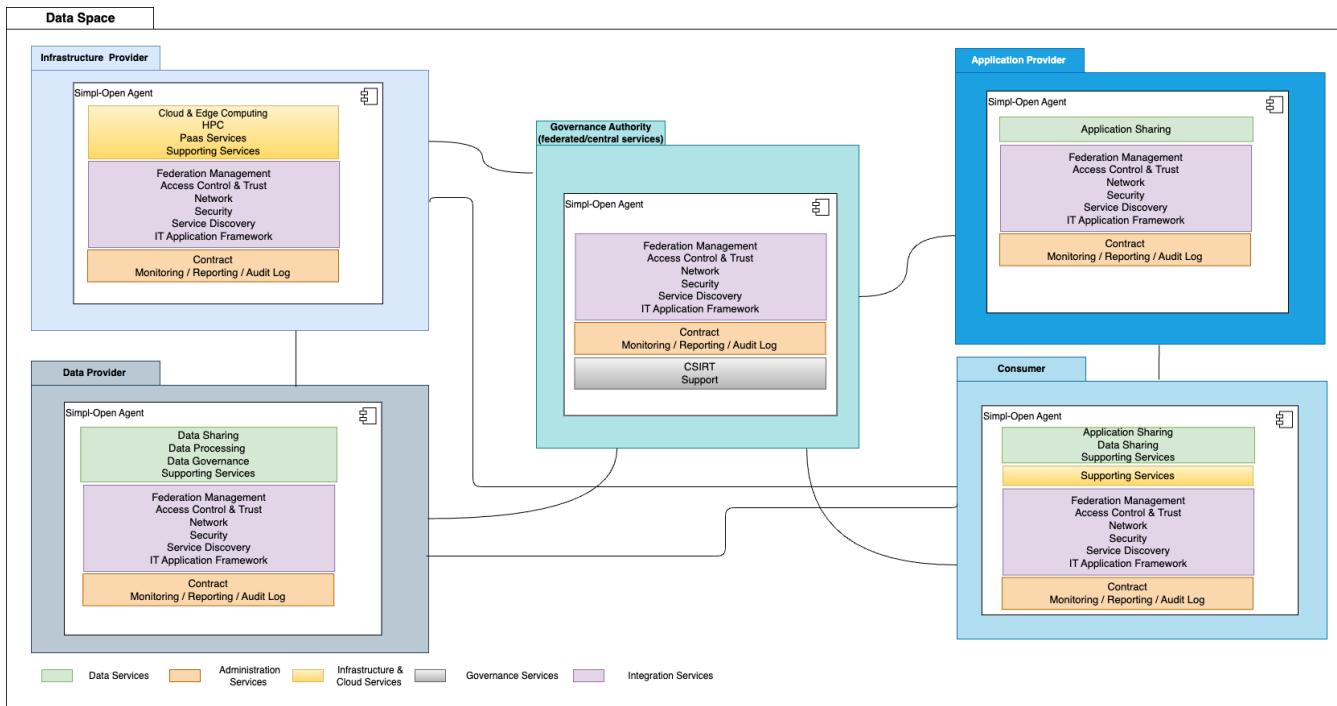
It must be noted too that another distinction is made in each of the layers between building blocks that can be accessed through Simpl-Open and building blocks that are built into Simpl-Open. This is an important distinction, as building blocks that are accessed through Simpl-Open translate to software components that are compatible with existing solutions through their APIs. Building blocks that are built into Simpl-Open require the absorption of possible existing solutions into the software stack of the platform.



Simpl-Open Agent

Legend: Built-in Access-Through

The architecture of Simpl-Open follows a loosely coupled self-contained architecture which groups components into building blocks, capability by capability. This approach permits the deployment Simpl-Open agent in different flavours depending on the type of participant, e.g. an Infrastructure Provider requires a different subset from the full Simpl-Open stack than a Data Provider. This modular architecture within a data space is presented on the following figure:



The following sub-sections describe how the capabilities interact and offer functionalities to participants.

Details on the smallest scale building blocks are explained in Annex 5 .

Integration layer architecture

The integration layer provides the capabilities for participants to securely and trustfully integrate with each other

Consumers can find resources of a data space through the tooling provided by the **Resource discovery** capability. First, providers make their resources (data, application or infrastructure) discoverable by submitting a well-structured metadata description of their resources – in a standardised format. Then consumers can query the catalogues to find suitable resources within the data space. These catalogues describe the content of the resources, how to consume them, and the policies that apply on this usage.

In the widespread context of Simpli-Open, **Security** is crucial to protect EU resources. As it will be described in Annex 5, security capabilities are constantly active. The agreed level of security encryption and integrity on every data transmission or resource deployment must be granted for the consumer at any time. For this reason, any provision of a resource will be inevitably deployed along with the security block, providing strong end-to-end security guarantees of all data that is handled by Simpli-Open.

Related to the security capabilities, the functions gathered by **Access control & Trust** capability will be constantly required whenever any participant (consumer, provider or governance authority) accesses Simpli-Open. Mapping end user roles with participant attributes (Role-Based Access Control - RBAC & Attribute-Based Access Control - ABAC), as well as authorizations to proceed with an action are addressed here. In this sense, every relation of the user with a data layer building block or infrastructure provisioning is closely screened by the Access Control & Trust. Simpli-Open will provide identification, authentication and authorization (IAA) building block for communication between data space participants, and integrate existing IAA systems of participating organisations for IAA of users within the organisation. To ensure that all access and usage policies are effectively enforced within the data spaces, Simpli-Open provides a policy enforcement capability. It also provides the necessary capabilities to submit/review/approve onboarding requests and deliver to the applicant the necessary security credentials to join a data space.

The **Network** capability contains building blocks for the establishment of secure network connections using technologies like virtual private networks. Additionally a firewall protects unwarranted access to Simpli-Open components and backend services. These network capabilities are relevant for connecting to infrastructure resources, as well as setting up the communication channels for data and application transfers.

The **Federation management** capability encompasses the general orchestration, as well as the supervision of the building blocks. It will oversee that the main principles of federation and interoperability are met by providing the means to connect resources. The federation management will encompass the needed configuration parameters for a well-functioning of Simpli-Open components. Such parameters may include the servers to connect to, rules concerning the lifecycle of recorded data, network configuration, and other parameters.

Finally, the **IT Application Framework** capability provides the necessary capabilities for the data spaces to deploy the building blocks in accordance with the proposed architecture (e.g. API Gateway, circuit breaker, service mesh, etc.).

Crucially, the Integration layer acts as an entry door towards the other layers. Below sections describe how the Data and Infrastructure layer are organised. The data layer provides blocks related to applications and data, whereas the infrastructure layer provisions computing, storage, and other infrastructure resources. When a consumer needs to access any of the building block belonging to the Data layer, the Access control & Trust capability is triggered to assure the right permissions are given. Additionally, authentication and authorization mechanisms are a prerequisite for executing the Data layer building blocks, such as the Application or Data sharing capabilities.

Once the Integration layer confirms the identity and the role of the end user, the contract capability of the administration layer is activated in order to verify the terms agreed and the Service Level Agreements. After this process, data building blocks can be executed. The explanation above concerning data building blocks is applicable to the Infrastructure layer building blocks in case the consumer is intending to access computing, storage, or other infrastructure resources. Nevertheless, it will be common to combine data and infrastructure resource usage by means of a distributed execution. It must be noted that beyond the usage of data and infrastructure resources in combination, independent use of infrastructure or data alone will be a possibility as well.

When considering the security and orchestration needs for the global usage of any of the Data and Infrastructure layers building blocks, the Integration Security and Federation management will be responsible of the correct encryption and verification deployments, assuring as well a correct allocation and interoperability of the resources. It must be noted however that both the Data and the Infrastructure layers will hold their own local orchestration blocks as explained in their respective sections.

Data layer architecture

Two prominent capabilities are the **Application sharing** and **Data sharing**. These contain the building blocks required for providers and consumers to exchange both data and applications. The capabilities create the connection between stakeholders to share data and application resources. The data sharing capability encompasses several functions including management of various types of data sharing, from transferring a single, few megabyte file, to transferring a terabyte-sized data dump. The Simpl-Open architecture foresees a simple data transfer mechanism and two special types of data transfer: bulk transfer and data streaming. A datastore connector handles the connection to the backend data store of the data provider, which can vary from a simple file storage to a relational database system. Additionally, this layer addresses a number of scenarios where **Data processing** tools will be desirable to process data as near as possible to the source. Among these tools, data anonymization tools support data providers and consumers to protect the privacy of data owners. On top, **Data governance** tools are offered by Simpl-Open for consumers and providers who can verify the integrity and quality of the required datasets.

Sharing applications is similar to sharing data. Indeed, at their core, applications are no more than a collection of data that is marked to be executable. However, specifics of application sharing come in terms of formats and the fact that usually multiple files need to be combined correctly to be able to run the application. It also adds additional considerations to handle the security of executable code and the trust that consumers have in the provided application. Simpl-Open will define the procedures to use for sharing applications. The application sharing capability considers three types of applications: full-fledged software packages, isolated algorithms, and machine learning models. Each type of tool comes with different specifics and runtime environment requirements that Simpl-Open should adhere to.

Additional building blocks of the data layer orchestrate data resources across Simpl-Open actors. The **Data orchestration** and **Distributed execution** capabilities allow actors to pool together data from different sources and manage partial sets of data across infrastructure providers when executing distributed applications. The combination of these capabilities allows consumers to gather data from different providers and spread it over distributed infrastructure where data is fed into an application.

Infrastructure layer architecture

The capabilities provided by this layer enable the consumers to easily provision the necessary computing and storage resources to execute their workloads in a secure and energy-efficient way.

The **infrastructure orchestration**, automates the provisioning of the infrastructure resources to enable the various infrastructure providers to interconnect and get exposed via a standard interface. The **distributed execution**, allows the consumer to deploy applications and execute computations close to the data.

The **cloud & edge computing** capability provides the opportunity to provision various resources to execute computations or store data in the environment of their choice. The **platform-as-a-service** building blocks provide several database engines and other platform-level resources. Finally, the **HPC** capability permits the consumer to perform complex calculations at high speed by providing a cluster of high-performance computers.

The infrastructure building blocks can be easily combined with each other to create even more value for the consumer. For instance, after successfully analysing certain data with the help of the provisioned *PaaS* analytical resources or the *HPC* capability, the consumer may want to store the used datasets and/or the results of their calculations. In this case, the *PaaS* storage building block can easily fulfil the storage needs of the end user. In case a consumer would like to develop a stand-alone application, they may also use various *PaaS* resources at the same time. They can leverage the different storage options to store each sort of data in the most efficient manner (e.g. the transactional data in a transactional database, while the sensor data in a NoSQL database). Besides, they can use the *cloud & edge computing* capabilities to deploy and run their applications, and the *distributed execution* capability even enables them to run the code close to the edge.

Administration layer architecture

The administration layer addresses the main capabilities that will assure the correct delivery of capabilities within Simpl-Open. Despite being depicted as isolated bubbles, capabilities and building blocks are strongly related and further discussed (see Annex 5). The function of this layer can be described as a set of supervision and management functions that will lead to a better control and interoperability of the rest of Simpl-Open building blocks. Along with the later described governance layer, it will play an executive role within Simpl-Open.

Whenever a consumer accesses a resource through Simpl-Open, the **Contracts** capability will determine if the licenses are correctly delivered. It will also register the billing terms and will manage the service level agreements as specified by the providers, as well as managing the permissions related to data sharing. The usage contracts will be accessible through this Administration layer every time a consumer requests a data or computing resource.

The **Logging, Monitoring and Reporting** capabilities are strongly interconnected. In the case of Logging, it is regarded as the real time information collection, while Monitoring is about screening the collected information and registering alerts and usage information concerning other layer's building blocks, as well as energy and quality of service optimization. On the other hand, the Reporting capability will handle the historical record of such information, as well as the general platform usage, allowing the relevant stakeholders to export and log the extraction of the information obtained. These two capabilities could be visualised as a supervision of the processes taking part at every layer level within Simpl-Open. Should additional action be taken, the Governance Authority will resolve the situation with the assistance of other Administration layer blocks such as Security or Contracts.

In order to analyse if the resources given by Simpl-Open are meeting contracts, access or security requirements, an auditing tool in the **Audit** capability will be capable of receiving information inputs from the logging, reporting and monitoring building blocks. By comparing what is expected from Simpl-Open resources and what is actually happening at a resource delivery level, the audit capability will interact with the Logging, Reporting and Monitoring capabilities.

The Administration layer will perform consistent and continuous analysis of the usage with the help of the Reporting and Monitoring capabilities. Along with the Auditing capability, the Administration capabilities will extract valuable information about the resources exchanged between the consumer and the data, application, and infrastructure provider(s). This will include analysis of the usage kept in metrics logging and real-time monitoring, which will be transferred to the reporting capability for a wider history data recording. In this reporting function, performance of deployed resources, the level of efficiency or the usage of the platform and the sustainable utilization will be supervised by the administration.

Governance layer architecture

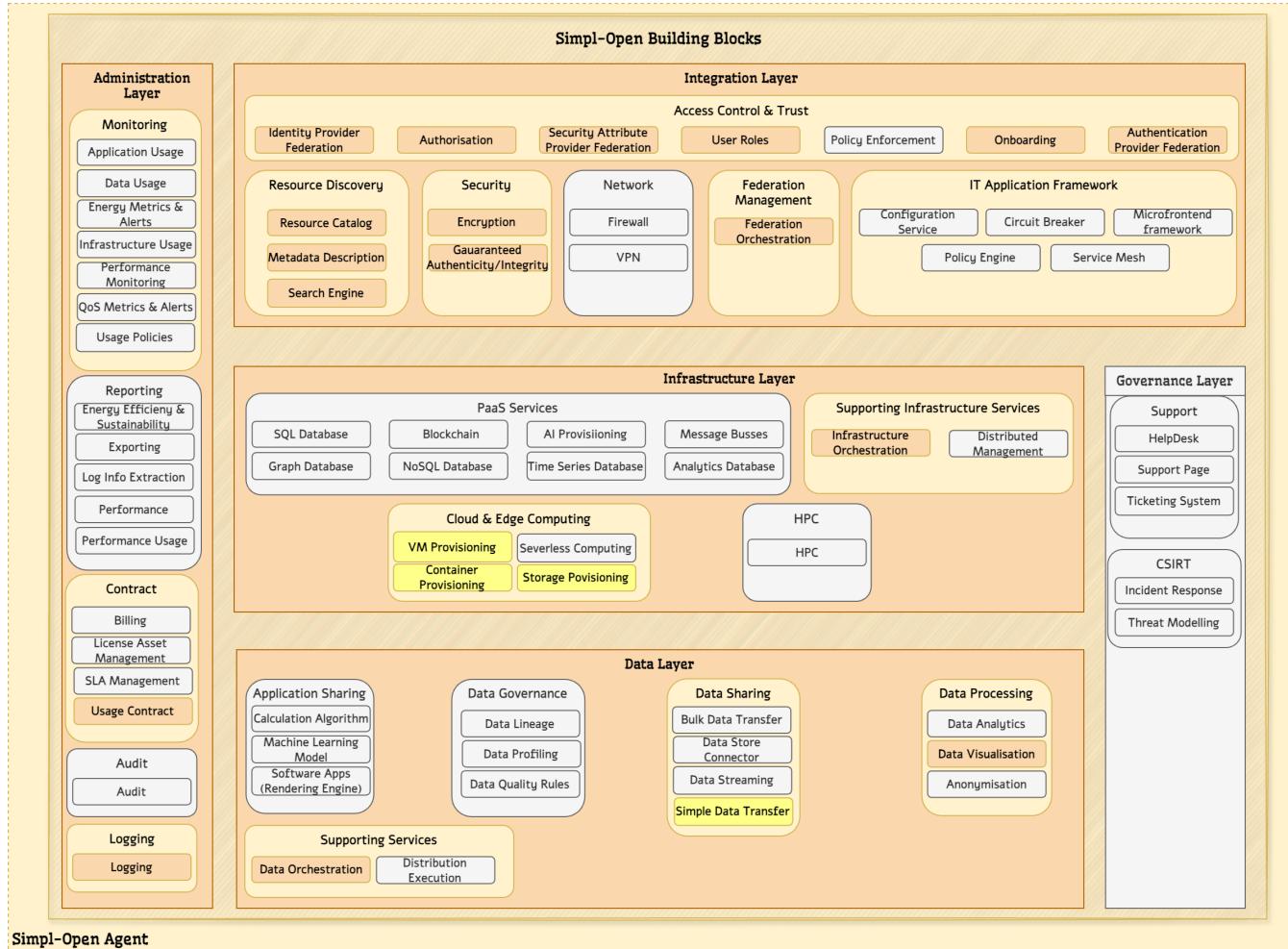
The capabilities provided in the governance layer enable the support and management of Simpl-Open. The governance layer can be divided in two categories of governance capabilities: Support and CSIRT (Computer Security Incident Response Team).

The **Support** capability is associated with the administration layer of the architecture and will assist Simpl-Open end users when issues arise during installation of Simpl-Open and during the use of other building blocks from Simpl-Open. Three building blocks are currently foreseen in the support category, the first building block is a Support webpage for Simpl-Open users. The second support building block is a Ticketing system where Simpl-Open end users will be able to log and keep track of issues regarding Simpl-Open and the third building block is a Helpdesk where Simpl-Open end users can connect to in case issues remain unsolved.

The **Computer Security Incident Response Team** capability is the second category of governance capabilities which includes the incident response and threat monitoring building blocks. The Incident response building block will have procedures to respond to security incidents to restore the compromised Simpl-Open functionality as soon as possible. The second building block, Threat monitoring, will proactively monitor and follow-up on possible malicious activities that could affect Simpl-Open to avoid potential security breaches.

Scope covered by the MVP

Below figure depicts the capabilities that will be (partially) implemented as part of the MVP (December 2024):



i The current version of this document covers the architecture of the MVP only and as such, following sections only focusses on components implementing the capabilities mentioned above as being in scope of the MVP.

Placeholders have also been added for content that will be made available after the MVP, with clear disclaimers at the beginning of the respective sections.

Architecture Framework

Architecture Approach

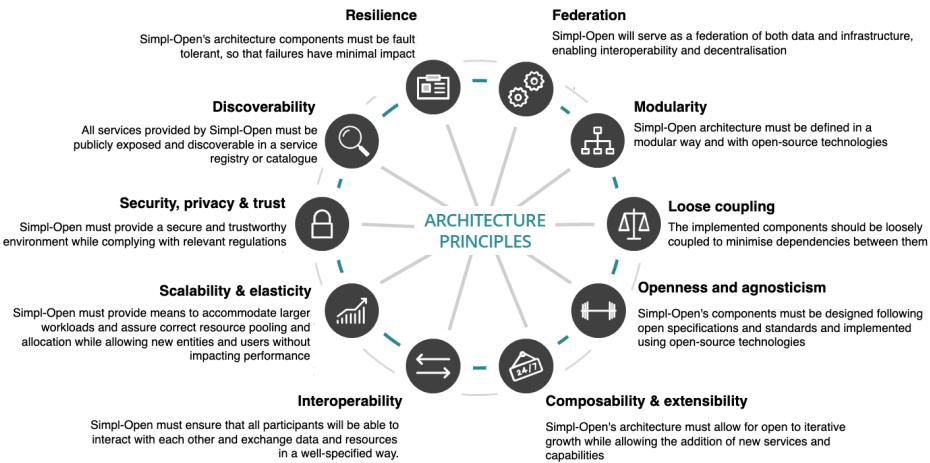
The architecture of Simpl-Open is created using a layered approach, inspired by the TOGAF methodology, which is reflected in the structure of this document:

- Business Architecture - describes how Simpl-Open should achieve its business goals and respond to the strategic drivers set out in the Architecture Vision. This layer was already defined in the preparatory study and this document only provides an update on the functional capabilities (which have evolved since then) and revisited concepts of business processes.
- Application Architecture - develops the target application architecture of Simpl-Open that enables the business architecture and the architecture vision, in a way that addresses the requirements. It identifies architecture components through Solution Views (business process-based approach, both static and dynamic) and Deployment Views (agent type-based approach, static only).
- Data Architecture - presents data entities and/or collections and how they are structured within the system.
- Technology Architecture - develops the target technology architecture that enables the application architecture to be delivered through technology components and technology services. Each application building block is mapped to a technology implementing the capabilities. Just like for the application architecture, both Solution and Deployment Views are defined.
- Security Architecture - covers the security aspects of the architecture.

The list of architecture principles to which we adhere is presented in the next section. Some of the architecture patterns used are also described in Annex 3 - Architecture Patterns.

Architecture Principles

Simpl-Open is designed upon ten architectural principles. Each of these principles is applied throughout Simpl-Open's design. They are all equally important to the design. The following figure provides an overview of these principles:



- **Federation**: Federated systems describe autonomous entities, tied together by a specified set of standards, frameworks, and legal rules. Simpl-Open should federate data, infrastructure and applications. This principle is key to enable interoperability and information sharing among the different entities that will be part of Simpl, while giving maximum autonomy to service owners.
- **Modularity**: The architecture of Simpl-Open needs to be defined in a modular way which allows the replacement or addition of components without affecting the rest of the system. This also provides the possibility to implement every component with a different open-source technology. Through modularity, Simpl-Open users are able to deploy a specific subset of components that are tailored for their purposes.
- **Loose coupling**: Components and services should have minimal dependencies on each other. Standardised, business-oriented APIs make sure consumers are not impacted by changes to services. This allows service owners to change implementation, switch out components, or modify data records behind the APIs without downstream impact to end users. This principle ties in with the *modularity* and *resilience* principle.
- **Resilience**: Components of the architecture must be fault tolerant, such that failures in one of them will have minimal impact on other components. Single points of failure need to be avoided to the maximum extent possible as the main objective is achieving a distributed architecture.
- **Openness & agnosticism**: The open specification allows insights into all parts of the architecture without any proprietary claims. It makes adding, updating or changing components easy for all users. Services should be provided irrespective of specific technologies and should be executable in all environments.
- **Composability & extensibility**: Simpl-Open's architecture should allow services to deliver value to the business in different contexts, providing the necessary tools to facilitate their composition together with other services to form new aggregated services. Simpl-Open remains open to iterative growth allowing the addition of new services and capabilities that fit future use cases to the platform. An open development community should be promoted in order to enable the contribution of new features that extend Simpl-Open's functionalities by its members.
- **Interoperability**: Simpl-Open enables interoperability between its participants to share resources in a well-specified manner. The architecture should describe the technical means to achieve this and be agnostic to the specific implementation details of each participant.
- **Scalability & elasticity**: Simpl-Open provides the means to accommodate larger workloads and allow new entities and users on the platform without affecting the performance. Both vertical scaling – i.e. the practice of adding more resources to a single node – and horizontal scaling – i.e. the process of duplicating nodes – should be possible. Simpl-Open's performance should be able to follow user demand without deteriorating.
- **Security, privacy & trust**: Users of Simpl-Open must be confident that when they interact with other entities they are doing so in a secure and trustworthy environment and in full compliance with relevant regulations. Data confidentiality, availability and integrity must be guaranteed. Privacy of data subjects, Simpl-Open users, or individuals must be assured.
- **Discoverability**: All services that are deployed in Simpl-Open will be 'publicly' exposed and discoverable in a service registry or catalogue. In this context, 'public' is seen as visible by all approved participants of a data space, not the public internet. Services will adhere to a service description, providing interested parties with a clear understanding of their business purpose and technical interface.

These architecture principles are completed with coding principles which can be found in the Development Handbook

Assumptions and Architecture Decisions



This information is based on currently available information tailored for MVP (December 2024 release) only.

ID	Topic	Assumptions / Decisions
----	-------	-------------------------

AS M-01	Data space data management (downloading data vs using it)	<ol style="list-style-type: none"> 1. Infrastructure provider is a mandatory intermediary to enable security of data processing. 2. It is the responsibility of the infrastructure provider to setup access control on the provisioned infrastructure tenant for the consumer. 3. It is the responsibility of the data provider to setup policy enforcement measures (e.g. restricting download) on the infrastructure tenant for the consumer. 4. In the case where data from the data provider is downloaded directly by the consumer (without an infrastructure provider involved), then the "technical enforcement" is replaced by a "legal enforcement".
AS M-02	Possible data sharing scenarios	<p>The following scenarios to share data exist:</p> <ol style="list-style-type: none"> 1. Simple Data Download: Data Provider is willing to offer the possibility of downloading the dataset to the consumer. The contract will create some legally binding usage policies. in scope for the MVP. 2. The data/app provider offers one, or a bundle of infrastructure instances that host both the data, and the application that can process the data. The consumer still has the possibility of downloading the data, but the contract may prevent it or put limitations/usage policies on it. not in scope for the MVP (but could be implemented in the future). 3. The data/app provider offers a bundle of infrastructure instances that host the data and the application that process the data separately. The access will be provided only to the application (or the infra instance that hosts the application). The consumer cannot access the data, and as part of the contract they shouldn't even try. could potentially be in scope for the MVP. 4. Compute to Data or loading the data in confidential memory enclaves (such as Intel SGX). An advanced version of Scenario 3, with more technical complexities. not in scope for the MVP (but could be implemented in the future).
AS M-03	Actors with multiple participant roles	<ol style="list-style-type: none"> 1. One agent per participant role (i.e. multiple agents required if a participant plays multiple roles in the data space). 2. One standard deployment script per type of participant will be provided.
AS M-04	Distinction between Certificate /Credentials	<p>There is a clear distinction between credentials for securing the data space (Tier 1 and 2 IAA) and the credentials for signing SDs and contracts (legally binding signature).</p>
AS M-05	Data sharing connector	<ol style="list-style-type: none"> 1. A connector agnostic "Asset Manager" will be developed, which can access different storage types and handle the data transfer. For the MVP, existing plugins of the EDC connector will be used (such as the S3 object storage extension, that can handle access management and data transfer, in case of contracting). The asset manager will be a module of the agents. 2. The combination of the connector (any) and the asset manager will be a part of the agents, for example the consumer and the provider agent.
AS M-06	Contract signature	<p>For the MVP, only a dummy signature will be used (not a legally valid one).</p>
AS M-07	Usage of a data space connector	<p>Any communication/transfer between agents will be done via data space connectors. They are responsible to implement the 3 aspects of the Data Space Protocol (DSP):</p> <ol style="list-style-type: none"> 1. registering and requesting service offerings in/from the catalogue; 2. negotiation of a contract; 3. enabling consumption of service offerings.
AS M-08	Storage attached to VMs and containers	<p>It is assumed that VMs and containers always have an attached storage.</p>
AS M-09	Type of storage supported	<p>It is assumed that Simpl-Open only supports natively S3-compliant storage but is extensible to support other storages (offering an API).</p>
AS M-10	Deployment and termination of built-in applications	<p>It is assumed that the application is always deployed and terminated together with the infrastructure resource as part of deployment script.</p>
AS M-11	Type of built-in application deployment supported	<p>It is assumed that Simpl-Open only supports natively applications deployed on Kubernetes but is extensible to support other platforms (offering an API).</p>

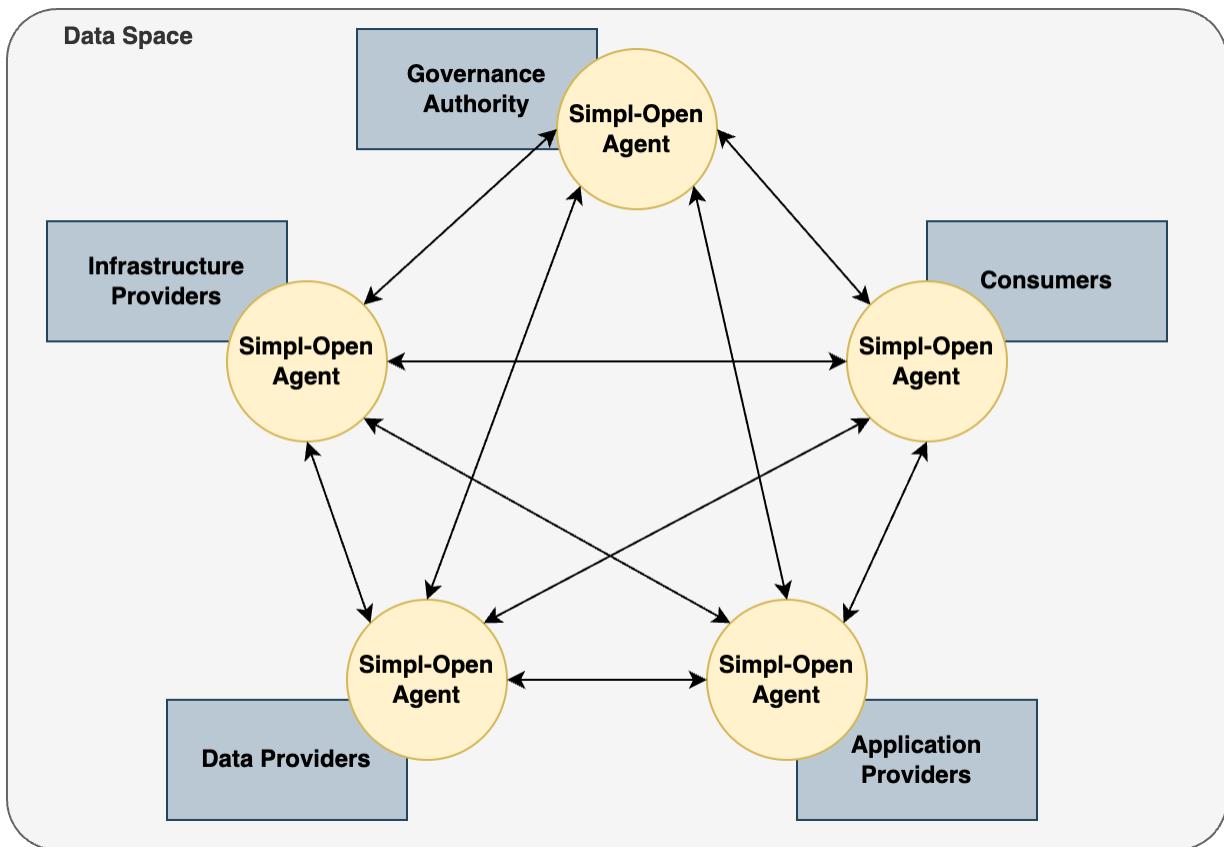
AS M-12	Supported infrastructure resources	<ul style="list-style-type: none"> It is assumed that Simpl-Open only supports natively: <ul style="list-style-type: none"> S3-compliant storage Kubernetes containers platform VMWare virtual machines <p>but is extensible to support other platforms (offering an API).</p>
---------	------------------------------------	---

Simpl-Open Business Architecture

Actors

An actor refers to an entity or participant that interacts with the system. Actors can be users, applications, Simpl-agent, etc. They play specific roles and have distinct permissions within the data space ecosystem.

The following context diagram introduces the main actors that will interact with each other using Simpl-Open and their interactions.



These actors are defined as follow:

Application Provider	The application providers cover all the data space actors offering applications to the consumers or any other type of participant. The term "application" is used in a rather broad sense in this document, and it covers any sort of executables including applications, as well as algorithms, such as a trained AI model that users can leverage to analyse their data. Application providers can also define the access control policies regarding their resources and bill the users for their usage.
Data Provider	This category covers all the data space actors <u>offering data</u> to the consumers. They can share one or more data sets and regulate the access and usage over the data with the help of policies. In order to compensate the data usage, the data providers can also bill the data space consumers. An example of a data provider can be an energy network operator sharing data on the energy grid load towards energy production facilities (who act as consumers) for production optimisation application.

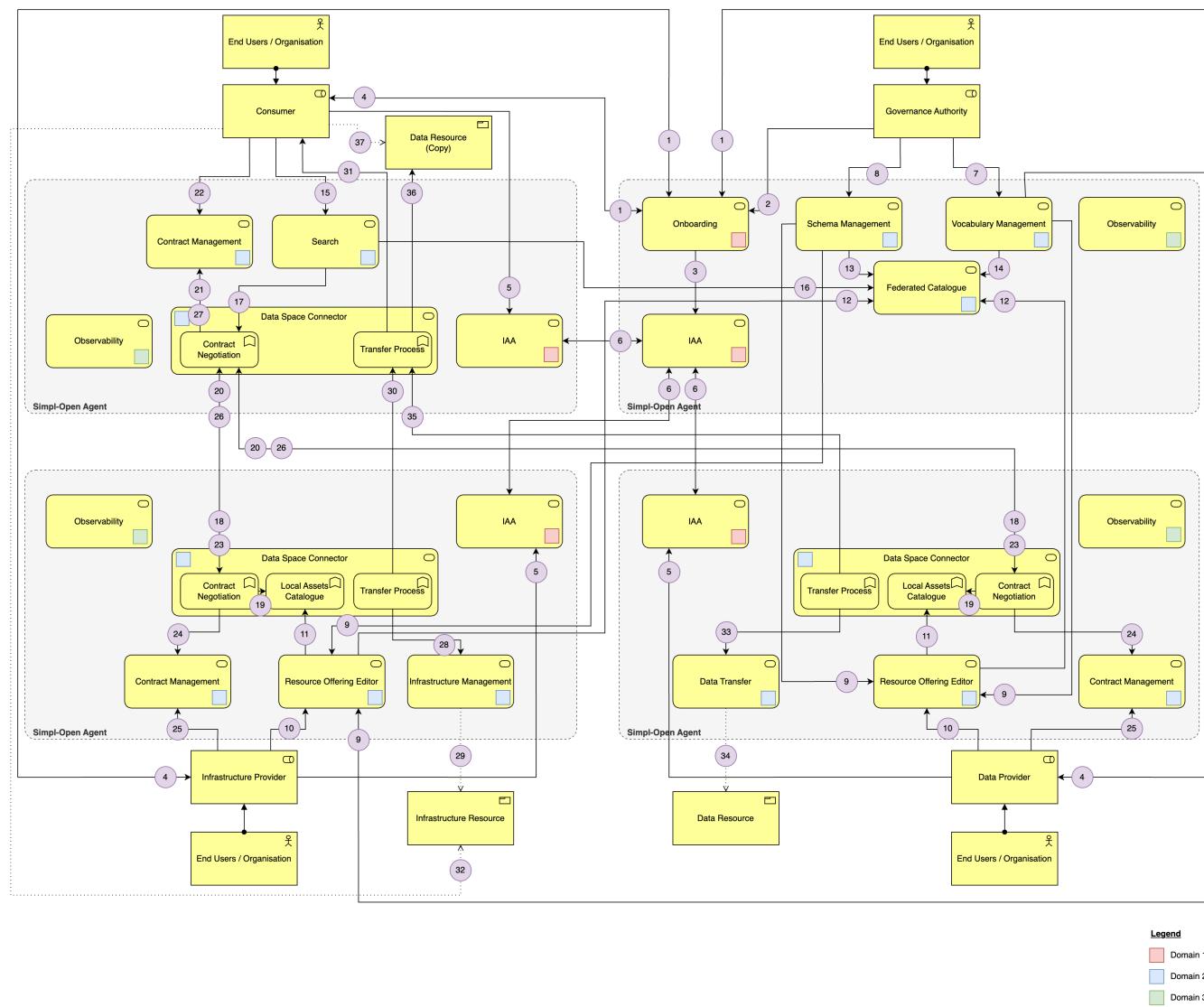
Infrastructure Provider	The infrastructure providers offer infrastructure resources and services to the consumers (or possibly to any other type of participant) to enable them to process the data provided by the data providers. They can, for example, launch virtual machines or containers and run applications, algorithms, or other executables on top of the underlying infrastructure. Similarly to the data providers, the infrastructure providers can define access control policies for the infrastructure resources and bill the middleware users for their usage.
Consumer	A consumer aims at using data, applications and infrastructure shared by providers. They can search for these and use them as allowed by the policies. For data, this means typically either using them online by utilising the infrastructure and applications provided by application and infrastructure providers, or if policy allows, download them for local usage.
Governance Authority	The data space participant that is accountable for creating, developing, operating, maintaining and enforcing the governance framework for a particular data space. ¹

¹<https://dssc.eu/space/Glossary/176553985/DSSC+Glossary+%7C+Version+2.0+%7C+September+2023>

Simpl-Open Functional Architecture

The following diagram presents Simpl-Open functional architecture.

An agent per type of participant is represented and the functional components are represented as ArchiMate services.



Below are described all the functional components presented on the diagram, how they implement the building blocks from the high level architecture, and how they interact between them. These interactions are highlighted with purple numbers on the diagram, which are linked to the below description through the purple numbers between brackets.

The **Onboarding** component implements the Onboarding building block: it provides the functionalities to submit, review and approve onboarding requests and deliver to the applicant the necessary security credentials to join a data space.

Both consumers and providers (data/application/infrastructure) can request to join a data space through the **Onboarding** component (1). This component allows the governance authority to control the required onboarding documents and approve/reject the onboarding request (2). If approved, the onboarding component sets up accesses and rights into the **IAA** component of the Governance authority (3) and delivers security credentials to the applicant (4).

The **IAA** component implements the Identity Provider Federation, Authorisation, Security Attribute Provider Federation and User Roles building blocks: it serves as a security intermediary for all communications between actors and components of Simpl-Open.

Once a participant has received security credentials from the **Onboarding** component, they install the credentials into their own **IAA** component (5). Once installed, the **IAA** component of all participants are federated, using the **IAA** component of the governance authority as trust anchor (6). In reality, the **IAA** component is connected to any single component of Simpl-Open as any interaction with the agent must be authorised and authenticated. For the sake of keeping the diagram readable, the relations between the **IAA** component and all the other components are not represented on the diagram.

The **Vocabulary Management** component implements part of the Metadata Description building block: it serves to harmonize the vocabularies in the data space, by providing the definition of metadata representation and, if required, the data representation standards.

The governance authority defines the vocabularies through the **Vocabulary Management** component (7).

The governance authority defines the schemas through the **Schema Management** component (8).

The **Schema Management** component implements another part of the Metadata Description building block: it provides the functionalities to define the ontologies and schema of the resource description (i.e. what properties can/should be part of it, what are their types, constraints and vocabulary).

The **Resource Offering Editor** component implements the last part of the Metadata Description building block. It provides the functionalities to create and sign resource descriptions (in the form of Self-Descriptions). It remains up to date with current metadata description standards by fetching schemas and vocabularies from the **Schema Management** and **Vocabulary Management** components (9).

The **Federated Catalogue** component implements the Resource Catalogue building block and part of the Search Engine building block. It provides the functionalities for providers to publish their resources and for consumer to discover these resources.

The **Search** component implements the remaining part of the Search Engine building block. It provides the functionalities for consumers to query and filter catalogue items to find the most suitable resources.

The **Data Space Connector** implements part of the Resource Catalogue, Usage Contract, Data Orchestration and Simple Data Transfer building blocks. It provides an implementation of the Data Space Protocol and acts as an orchestrator between its 3 parts:

1. Local Assets Catalogue in which the providers register the information, related to their own published resources, that is required for supporting the contract negotiation and transfer process
2. Contract Negotiation provides the electronic contract negotiation required for consuming any type of resources
3. Transfer Process supports the triggering of the data transfer or deployment of other types of resources.

Providers (data/application/infrastructure) create and sign resource description in the **Resource Offering Editor** component (10) and can then register it in the Local Assets Catalogue of their respective **Data Space Connector** component (11). The data registered in the Local Assets Catalogue of the **Data Space Connector** is the minimal subset of metadata required to enable the 2 next parts of the DSP: contract negotiation and transfer process.

Once registered locally, the Resource Offering Editor can publish the entire resource description (in the form of a Self-Description) to the **Federated Catalogue** component (12).

The **Federated Catalogue** validates the submitted resource descriptions against the schemas and ontologies provided by the **Schema Management** component (13) and against the vocabularies provided by the **Vocabulary Management** component (14).

Consumers can browse the resource offerings published in the **Federated Catalogue** through the **Search** component (15). Instead of having a search functionality embedded in the **Federated Catalogue**, the **Search** component is represented as a distinct component of the consumer agent, connecting to the **Federated Catalogue** in the governance authority agent (16), to enable the 2 tiers approach for IAA (the consumer end-user connects to the Search component via tier 1 and the Search component connects to the Federated Catalogue via tier 2).

Once consumers have found a resource offering that they would like to consume, they can request the consumption in the **Search** component which initiates a contract negotiation with the provider through the **Data Space Connector** component (17). The **Search** component has obtained from the **Federated Catalogue** the address of the provider's **Data Space Connector** and the identifier to the resource offering, and provides these elements to the **Data Space Connector**. Based on these 2 elements, the consumer's **Data Space Connector** initiates a contract negotiation with the provider's **Data Space Connector** (18).

Based on the received resource offering identifier, the provider's **Data Space Connector** can query its Local Assets Catalogue to obtain the necessary metadata to create a contract (19).

The provider's **Data Space Connector** provides the contract to the consumer's **Data Space Connector** for signature by the consumer (20).

As signing a contract is not explicitly part of the data space protocol, the signature process is not implemented within the Data Space Connector. Instead, it is externalised to the **Contract Management** component (21).

The **Contract Management** component implements the last part of the Usage Contract building block. It provides the functionalities to create, sign and persist usage contracts.

The consumer signs contracts through the **Contract Management** component (22).

Once signed by the consumer, its **Data Space Connector** provides the contract back to the provider's **Data Space Connector** for the provider to sign it (23). As for the consumer, the signature is delegated to the **Contract Management** component (24) through which the provider can counter-sign the contract (25). The **Contract Management** component persists the signed contract and provides a copy to the consumer via their **Data Space Connectors** (26).

The **Contract Management** component of the consumer persists the signed contract (27).

Once a usage contract agreement is established, the **Data Space Connector** of the provider can start data and/or infrastructure consumption.

For standalone infrastructure consumption (see BP 08), the **Data Space Connector** of the infrastructure provider triggers the deployment of the Infrastructure Resource through the **Infrastructure Management** component (28).

The **Infrastructure Management** component implements the Infrastructure Orchestration, VM Provisioning, Container Provisioning and Storage Provisioning building blocks. It provides the necessary features to deploy and configure (incl. policies) infrastructure resources. It also partly implements the Data Visualisation building block by providing the functionality to deploy a built-in data visualisation application on the infrastructure resources. The remaining part of the Data Visualisation building block is implemented by the built-in application itself.

The **Infrastructure Management** component deploys and configures the requested **Infrastructure Resource** (29) and provides access details back to the consumer via the **Data Space Connector** (30).

The consumer gets access details from their **Data Space Connector** (31) and can access the **Infrastructure Resource** using these details (outside of Simpl-Open) (32).

For direct data download (see BP 09A), the **Data Space Connector** of the data provider accesses the **Data Resource** through the **Data Transfer** component (33).

The **Data Transfer** component provides the functionalities to access various types of data resources and transfer them between participants. It implements the Data Orchestration and Simple Data Transfer building blocks.

The **Data Transfer** component accesses the **Data Resource** (34) and transfers a copy of it to the consumer via the **Data Space Connector** (35). The consumer's **Data Space Connector** stores the copy of the **Data Resource** on the consumer side (36), which can be accessed by the consumer (37).

For access to data over an application deployed on an infrastructure, for the MVP, both the data and application resources are already available in the infrastructure provider and are deployed together with the infrastructure resource. After the MVP, a solution involving the **Data Space Connectors** of both infrastructure and data providers will be envisaged.

The **Observability** component implements the Logging building block and part of the Monitoring building block. It provides the functionalities to collect and monitor logs and metrics from the other components of the agent.

In reality, the **Observability** component is connected to any single component of Simpl-Open as all of them produce logs and are monitored. For the sake of keeping the diagram readable, the relations between the **Observability** component and all the other components are not represented on the diagram.

From the above architecture, we can distinguish 3 functional domains:

1. **Onboarding & IAA** - This domain provides the means to join a data space and establish trust between participants.
2. **Publish and consumer resources** - This domain is about the essence of a data space: allow to share resources (datasets, infrastructure, applications) between the participants.
3. **Management/Operation of data space** - This domain provides the functionalities that are necessary to manage and operate a data space.

Below table summarises how the functional components implement the building blocks from the high level architecture, and how they map to the functional domains.

Functional architecture component	High level architecture building block implemented	Functional domain
Onboarding	Onboarding	Onboarding & IAA
IAA	Identity Provider Federation Authorisation Security Attribute Provider Federation Authentication Provider User Roles	Onboarding & IAA
Vocabulary Management	Metadata Description (partly)	Publish and consumer resources
Schema Management	Metadata Description (partly)	Publish and consumer resources
Resource Offering Editor	Metadata Description (partly)	Publish and consumer resources
Federated Catalogue	Resource Catalogue Search Engine building block (partly)	Publish and consumer resources
Search	Search Engine	Publish and consumer resources

Data Space Connector	Resource Catalogue (partly) Usage Contract (partly) Data Orchestration (partly) Simple Data Transfer (partly)	Publish and consumer resources
Contract Management	Usage Contract (partly)	Publish and consumer resources
Infrastructure Management	Infrastructure Orchestration VM Provisioning Container Provisioning Storage Provisioning	Publish and consumer resources
Data Transfer	Data Orchestration Simple Data Transfer	Publish and consumer resources
Observability	Logging Monitoring (partly)	Management/Operation of data space

A mapping between the functional requirements level 2 and the functional components presented above is provided in annex.

List of Business Processes and Functional Requirements

The following table summarises for each of the business domain, the list of business processes that are (at least partially) covered by the architecture described in this document. These business processes and the underlying functional requirements are available from the [Simpl Programme website](#).

Domain	Label ID (for Jira)	Business Process	BP ID	Sub-process	Published on website
	1_Setup_of_Dataspace	1 - Setup of Dataspace: Role of Governance Authority	BP 01		No
	2_Setup_of_ID_Trust_Catalogues_and_Vocabulary	2 - Setup of ID/Trust, Catalogues and Vocabulary	BP 02		No
Domain 1 - Onboarding & IAA	3_Onboarding_of_a_new_Dataspace_Participant_(T1,TII)	3 - Onboarding of a new Dataspace Participant: Providers of data, application or infrastructure and Consumers	BP 03A	BP 03A – Onboarding of a New Dataspace Participant - Providers (data - application - infrastructure) & Consumers	Yes
			BP 03B	BP 03B – Onboarding of a New Dataspace - Participant End-User	Yes
	4_Registration_of_Individual_Participants	4 - Registration of "Individual" Participants	BP 04	BP 04	On hold until further notice
Domain 2 - Publish and consumer resources	5_A_provider_adds_a_new_Resource_to_Dataspace	5 - A provider adds or updates a new Resource on data, application or infrastructure, on the Dataspace's Catalogue	BP 05A	BP 05 – A Provider adds or updates a new data-, application-, or infrastructure resource on the Dataspace's Catalogue	Yes
	6_A_Consumers_search_Resources_on_the_Dataspace	6 - Consumers search Resources on Dataspace's Catalogues	BP 06	BP 06 – Consumers search Resources on Dataspace's Catalogues	Yes
	7_Consumers_establish_a_Contract_with_a_Provider	7 - The Consumer and the Provider establish a Usage Contract for selected Catalogue items	BP 07	BP 07 – The Consumer and the Provider establish a Usage Contract for selected Catalogue items	No
	8_Consumers_select_and_use_an_Infrastructure_Resource	8 - Consumers select and use an Infrastructure Catalogue Resource from the Infrastructure Provider	BP 08	BP 08 – Consumer consumes an infrastructure resource from the provider	No
	9_Consumers_select_and_use_data_application_Resources	9 - Consumers select and use data/application Catalogue Resources from the Data or Application provider on the secure processing environment (established by 8)	BP 09A	BP 09A – Consumer consumes a data resource from the provider	No
			BP 09B	BP 09B – Consumer receives data processing service over a dataset via an Application	No

			BP 9C	BP 09C - Consumers selects and uses a Catalogue Resource from the Provider (Application)	No
		10 - Placeholder -	BP 10		No
		11 - Placeholder	BP 11		No
Domain 3 - Management /Operation of data space	12_Management /Operations_of_the_Dataspace	12 - Management / operations of dataspace business (catalogues, vocabulary, clean up, etc.): • audit of the dataspace usage • management of participants • helpdesk (to be seen how/if/what is in scope) • CSIRT • billing should be a detailed use case	WF 12B	WF 12B - Local Node Logging, Monitoring (collection and visualisation of events)* *Technical Workflow only, as this is not a BP	No
			BP 12D	BP 12D Audit by Governance Authority	No
			BP 12E	BP 12E Billing of Catalogue Resources Utilisations	No
	13_IT_Administration	13 - IT Administration		No Business Process, only workflows/architecture views.	No

Simpl-Open Application Architecture

Simpl-Open Application Architecture develops the target application architecture of Simpl-Open that enables the business architecture and the architecture vision, in a way that addresses the requirements.

It identifies architecture components through following views:

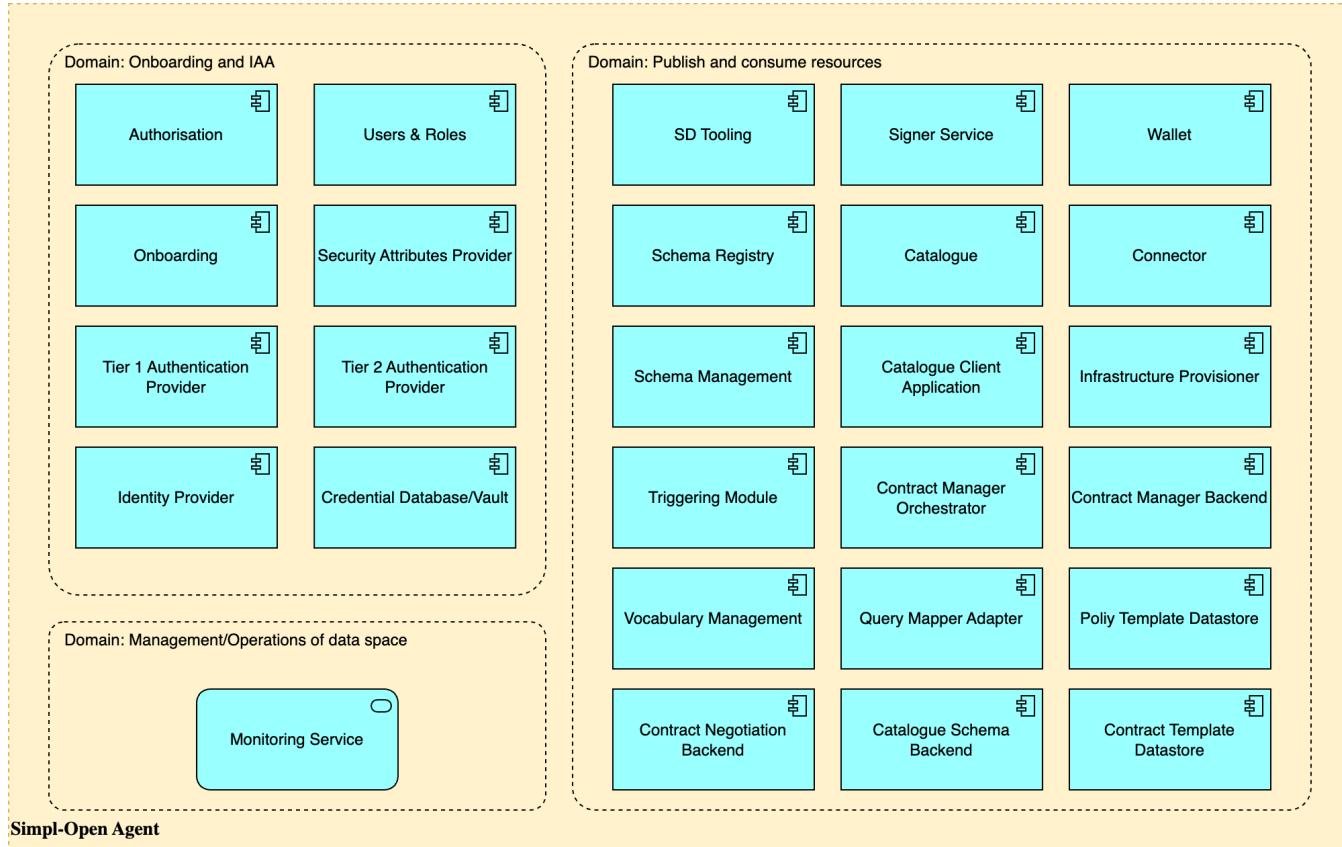
View	Description
Application Components Static View	Provide a view per business domain of the application "Solution", with all the main components and interactions.
Application Components Dynamic View	Provide a dynamic view per business process (or sub-process) on how application components are used to satisfy different workflows.

Next to these architecture views, are provided:

- Workflows - technical information flows, not identified as business processes, which explain how a specific sequence/composition of application components could satisfy a business process implementation, with clear distinction of human and machine actors;
- Interfaces - describes APIs and/or UIs for each relevant architecture component presented in above views.

An initial list of requirements stemming from the Simpl-Open tender specifications and which could drive elicitation of Simpl-Open non-functional requirements is provided in Annex 4. Once these non-functional requirements will be elicited, they will be published on the Simpl website.

The following figure presents an overview of the application architecture components, grouped by Functional Domain of the business architecture:



Application Components Views

Application components views are presented per functional domain in following sub-sections.

For each functional domain, are presented:

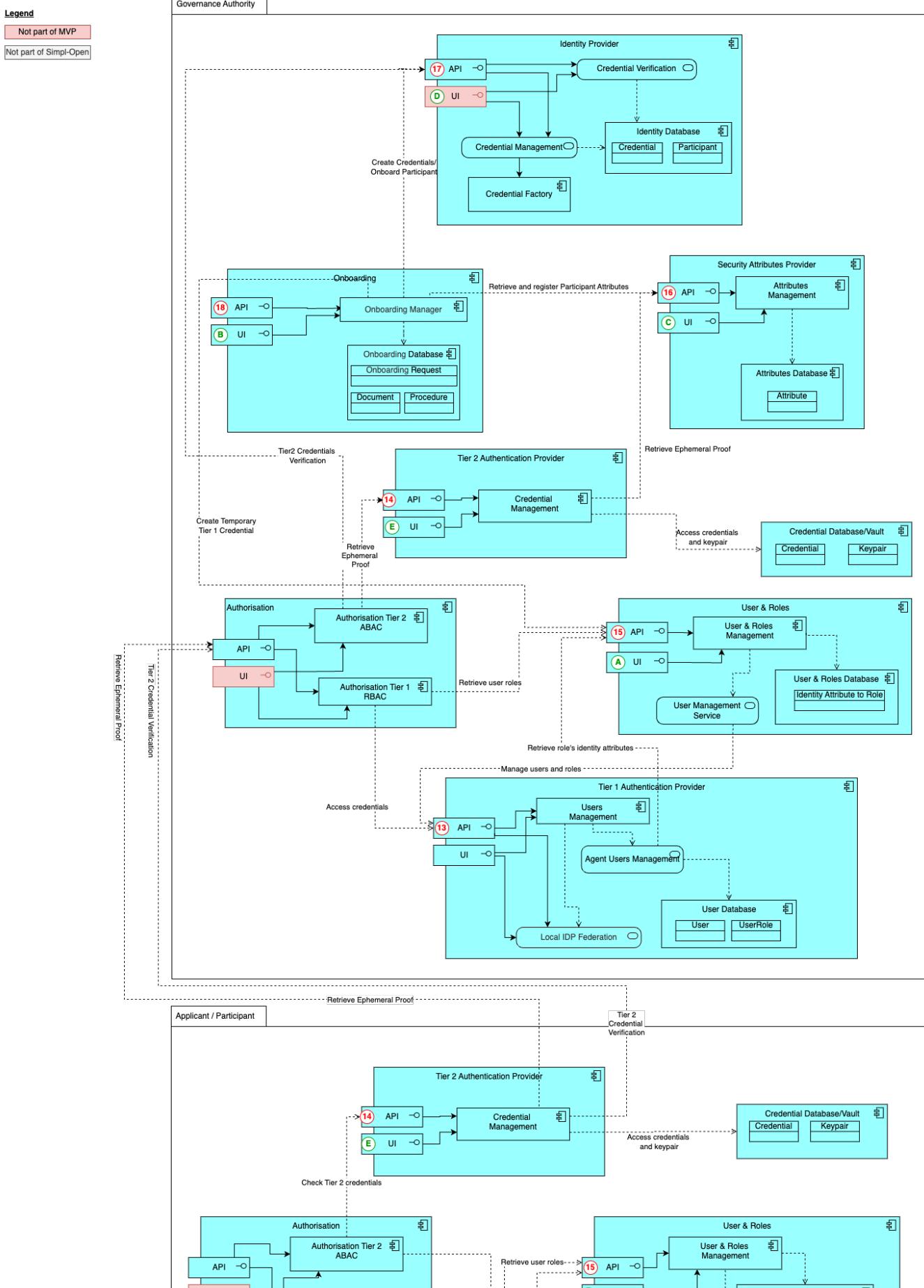
1. a static view of the entire domain which presents all the application components that are necessary to implement the functionalities of the domain and how they interact with each other;
2. a set of dynamic views that present how a subset of the application components are used to satisfy different (parts of) business processes.

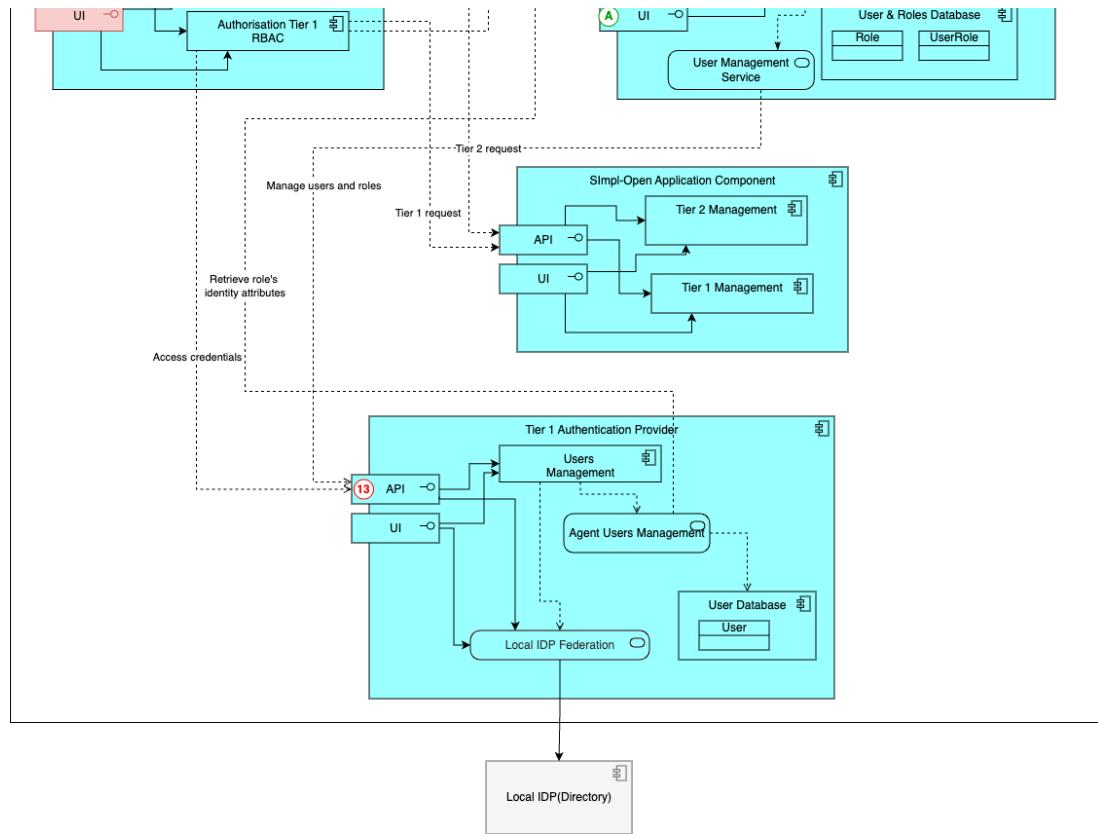
ACV - Domain 1 - Onboarding & IAA

The static view diagram illustrates the structural organisation of components involved in the domain, segmented into two types of agents (Governance Authority and a generic applicant/participant which can represent Consumer, Data Provider or Infrastructure Provider), showcasing the roles each plays in the domain.

As per the legend, components highlighted in red are foreseen to be part of Simpl-Open but are not part of the MVP, while components highlighted in grey are external to Simpl-Open.

The red numbers on the diagram help to correlate APIs with their definition which can be found in the Interfaces section, while the green letters makes the link with the User Interfaces which can be found in the same Interfaces section.





Onboarding

The **Onboarding** component is deployed inside the Governance Authority Agent and it's the core for managing onboarding requests by applicants (applicants can be both providers and consumers). This is where the applicant requests new Tier 1 credentials and initializes its onboarding request. The Governance Authority Tier 2 authorisation operator can approve, reject, or require new documents to fulfil the request. After the request has been approved, the applicant must create its keypair to be associated with the credential and can submit the public key to the governance authority which triggers the creation of a Tier 2 credential by the Identity Provider component.

Refer to ACV Dynamic - BP 03A - Onboard a Participant for a full description of a Participant Onboarding.

Identity Provider

This component is deployed inside the Governance Authority Agent. It generates the credentials for a newly onboarded participant and stores them along with the participant's information. This component also allows the applicant participant to download the generated security credentials that can then be installed in the Tier 2 Authentication provider of the participant agent.

Security Attributes Provider

The Security Attributes Provider component is deployed in the Governance Authority Agent and registers the participant's security identity attributes. Upon approval of an onboarding request, the onboarding component calls the Security Attributes Provider to associate the security identity attributes to the participant.

User & Roles

This component works as an interface in front of the tier 1 authentication provider. Its responsibilities are:

- reading and writing users and roles in the tier 1 authentication provider
- map the Tier 1 Roles to assignable security identity attributes
- create an applicant user along with temporary credentials in the tier 1 authentication provider at the beginning of the onboarding process

Tier 1 Authentication Provider

The Tier 1 authentication provider contains the participant users, roles and allows IdP Federation.

Refer to ACV Dynamic - BP 03B - Connect/map Organisation Local IDP (Directory) for a full description of IdP federation.

Tier 2 Authentication Provider

The tier 2 authentication provider is the component that

- manages the storage and update of the security credentials inside the Credentials Database/Vault component.

- inside a participant, is involved in 2 steps after the onboarding request has been approved:
 - when the applicant representative creates/uploads a keypair into a participant agent
 - when the applicant representative installs the security credentials previously generated by the governance authority
- in the communication between participants, helps the Authorization Tier 2 components to validate Tier 2 credentials (Ephemeral Proof and Security Credentials)

Credentials Database/Vault

Component that handles the physical storage of the participant credentials

Authorization

The authorization component processes all Tier 1 and Tier 2 inbound traffic originating from external sources and enforces RBAC and ABAC rules.

ACV Dynamic - BP 03A - Onboard a Participant

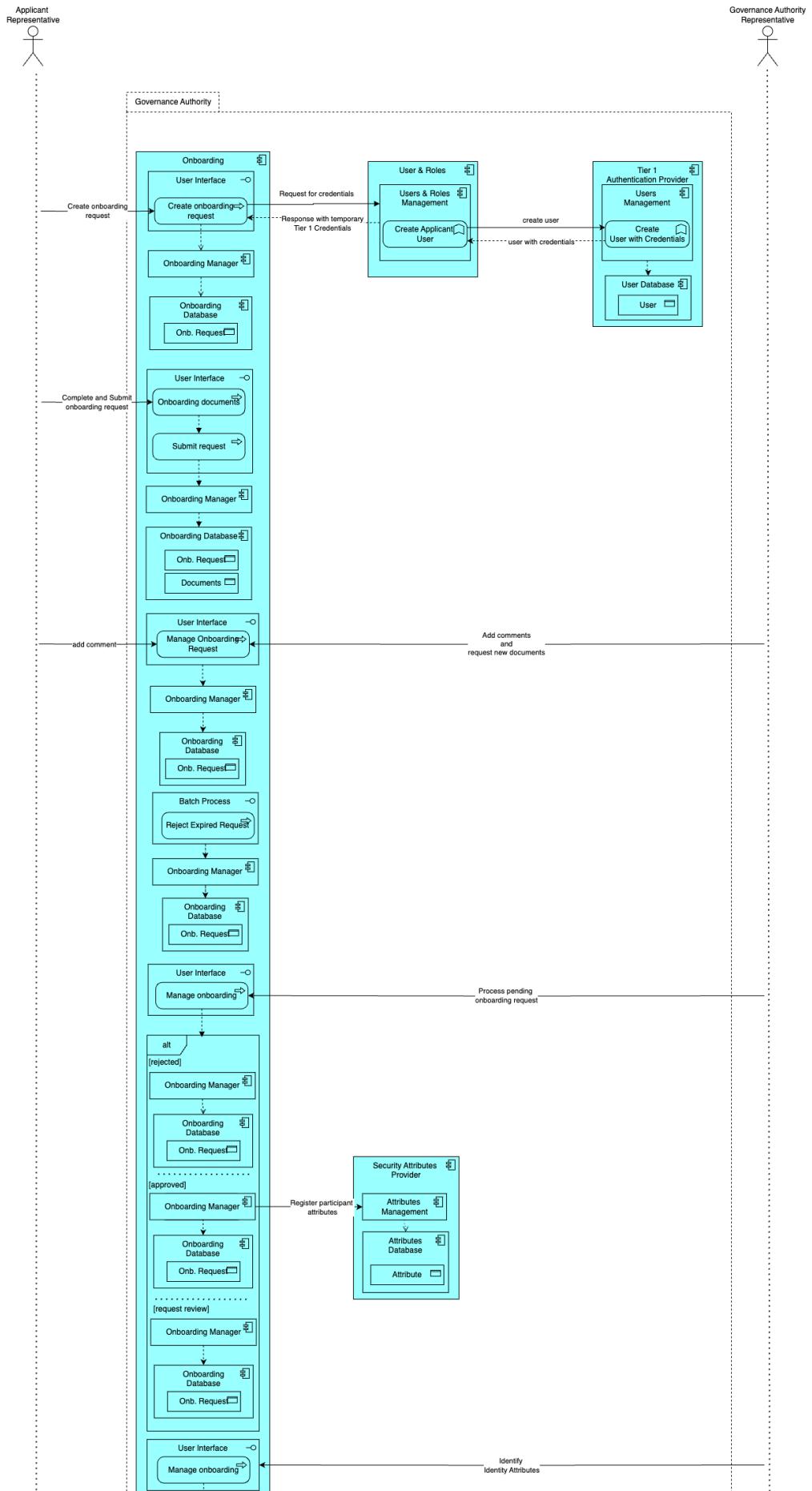
A new participant in a data space – whether a data provider, application provider, infrastructure provider, or consumer – begins by registering itself and obtaining a temporary Tier 1 credential.

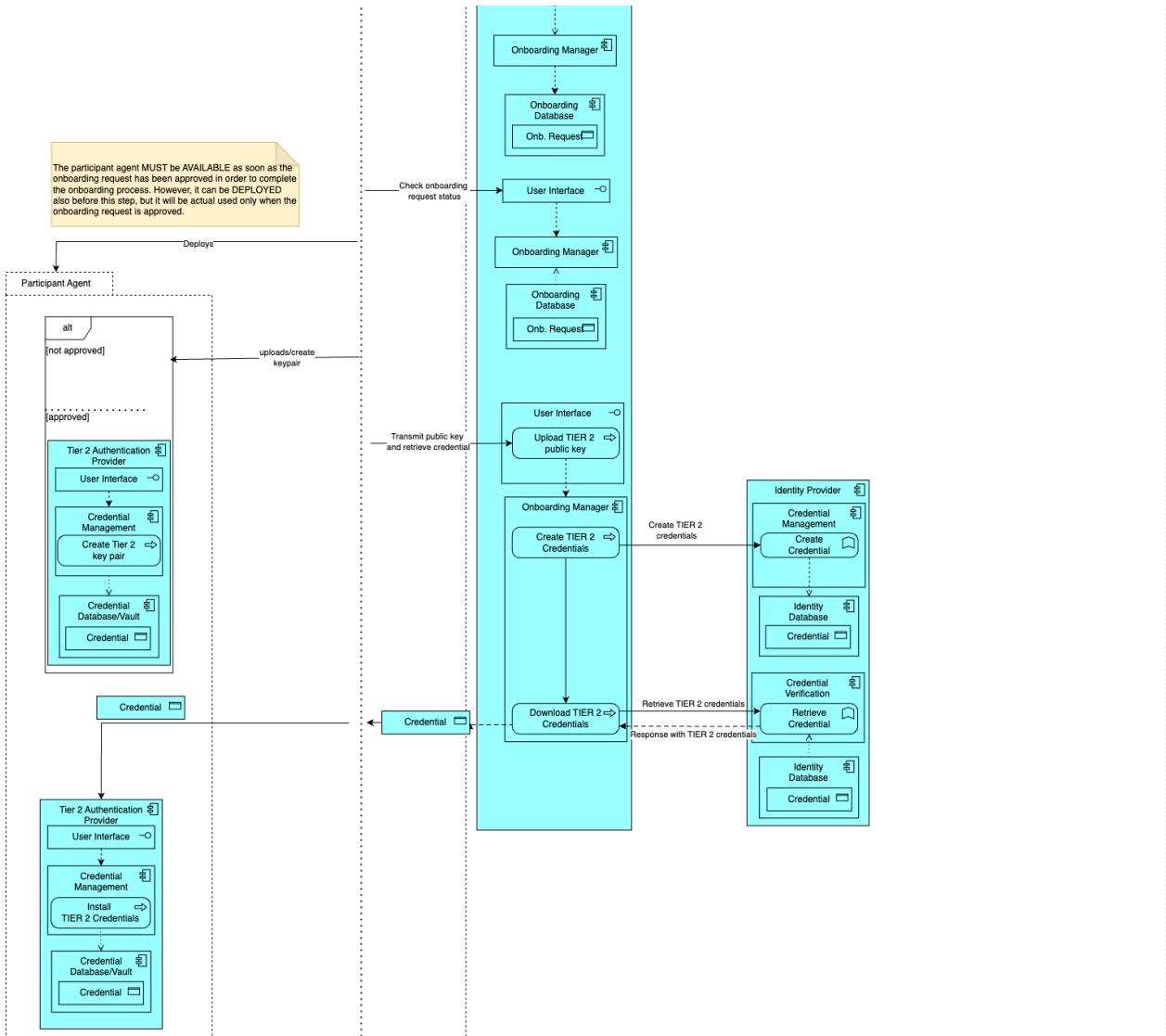
Using the Tier 1 temporary credential, the new participant submits an onboarding request by completing the required information forms.

The onboarding request is then processed by the Governance Authority and either approved or rejected. During the review process, the Governance Authority can provide comments on the onboarding request and submit requests for additional documents to the applicant participant.

Assuming the onboarding request gets approved, the new participant creates a Tier 2 key pair and begins the process of obtaining a valid identity credential for its Simpl-Open Agent. This credential proves the 'identity' of the installed Simpl-Open Agent and enables secure communication with other data space participants. Simpl-Open Agents will only permit communication with other network participants who hold a valid identity credential.

After the participant has successfully acquired a valid identity credential, they proceed to install this credential within their Simpl-Open Agent. This installation process involves integrating the credential into the agent's system, ensuring that it is properly recognised and authenticated.





- Applicant Participant creates an onboarding request:** the applicant participant requests credentials in the Governance Authority providing information about the organization and the participant's role in the data space (consumer or data/infrastructure/application provider). Credentials are created in the Governance Authority Tier1 Authentication provider through the Users&Roles component. After the credentials have been created and stored in Tier1 User Database, the onboarding component creates an onboarding request with the status IN PROGRESS.
- Applicant participant completes the onboarding request⁽¹⁾:** the applicant participant logs in to the onboarding Frontend using the temporary credentials and fills the onboarding request providing the required documents and adding comments, if needed. Once all the required documents have been uploaded, the applicant can submit the request for review to the Governance Authority representatives.
- Governance Authority representative reviews the onboarding request⁽¹⁾:** when the onboarding request has been submitted, the governance authority representative reviews it and decides:
 - to APPROVE the onboarding request and proceed to the credentials creation step.
 - to REQUEST A REVIEW to the participant applicant, possibly requiring additional documents (step 2)
 - to REJECT the onboarding request. In this case the onboarding process stops.
- Security Attributes Registration:** as soon as the onboarding request has been approved, the onboarding component saves the participant identity attributes in the Security Attributes Provider component.
- Keypair creation:** once the onboarding request has been created, the applicant representative can start the credentials creation process inside the participant agent. The applicant representatives generates a keypair and stores it in the participant agent.
- Credential creation:** the public key (whose keypair is safely stored inside the participant agent) is sent by the applicant representative to the governance authority. Using the public key, the onboarding component triggers a credential creation through the identity provider component. After the creation, the applicant representative can download the credential.
- Credential installation:** the participant applicant can install the generated credential inside the participant agent along with the previously generated keypair (step 5).

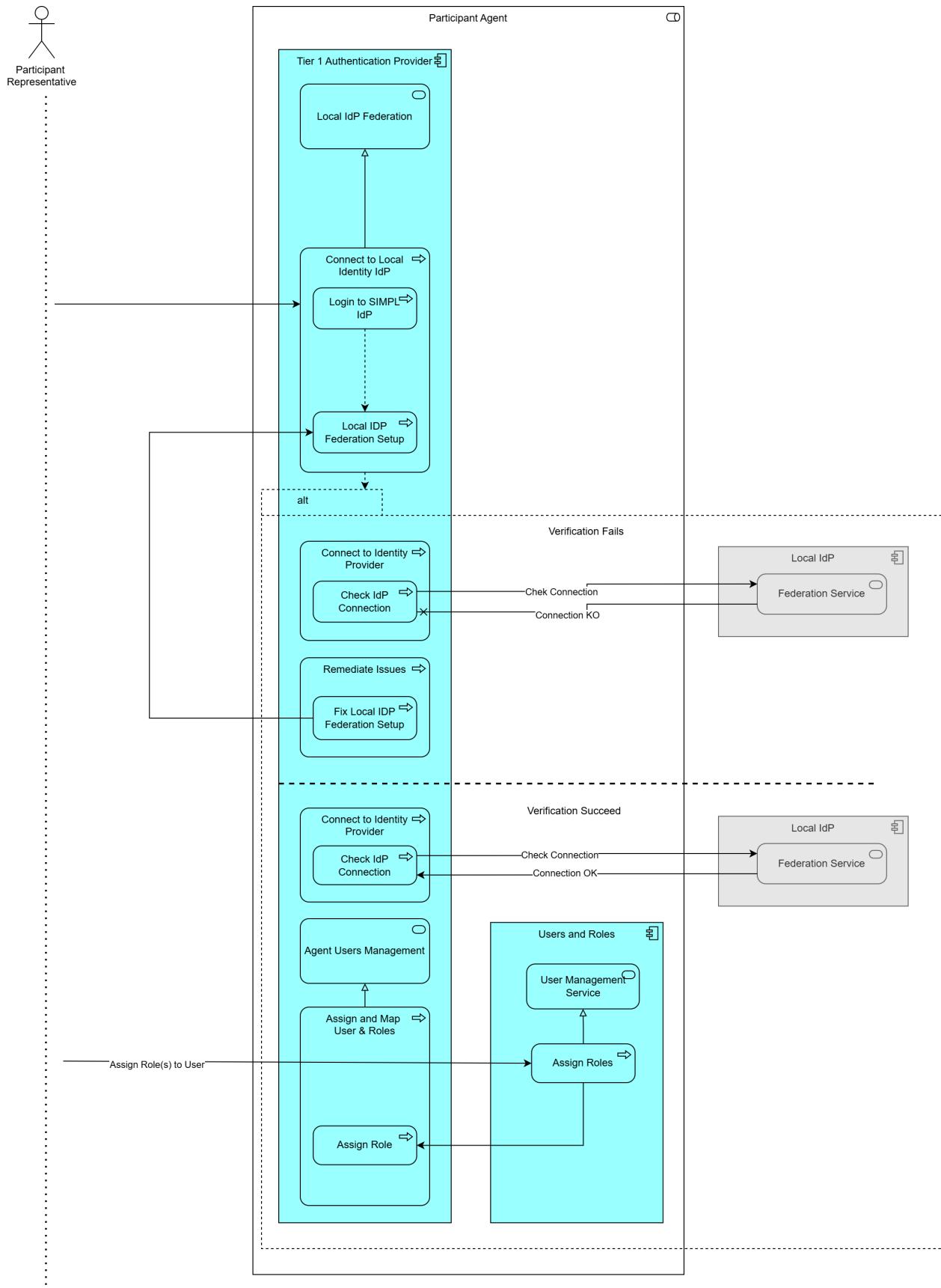
⁽¹⁾ Note that implementing a notification system is necessary to inform relevant users when comments are added or the status of an onboarding request changes, prompting them to take appropriate action. However, the development of such a notification system, including the specification of notification channels (e.g., Email, Frontend Notifications, SMS, etc.), is beyond the scope of the MVP.

ACV Dynamic - BP 03B - Connect/map Organisation Local IDP (Directory)

The participant must federate the local identity provider to the Authentication provider module of the Simpl-Open Agent.

This step is crucial as it ensures that the Simpl-Open Agent can accurately verify and manage existing users' identities. The new participant is responsible for assigning users to their respective roles within the Simpl-Open Agent for their organisation. The federation can be configured immediately after the participant agent is installed, without waiting for the full participant onboarding process to complete. Configuring the local IdP beforehand allows the participant to define and manage its own roles, enabling their use as soon as the onboarding process lands in the participant agent for the keypar generation/upload (see BP 03A).

This assignment ensures that each user has the appropriate access and permissions to perform their tasks effectively.



1. **IdP Login and Federation Setup:** the participant representative with administrative roles logs in to the SIMPL Identity Provider and uses the provided UIs to configure the federation with the Local IdP. The Local IdP serves as the identity provider with a pre-configured authentication mechanism, enabling organization users to log in to the organization's existing applications. In the federation setup the representative configures all the endpoints, credentials, configuration and mappers that are needed to connect to the Local IdP and federate it.
 2. **Connect to identity provider:** the participant representative checks that the federation has been configured correctly
 3. **Remediate issues:** the participant representative fixes some configuration problems that may arise during the setup phase
 4. **Assign and Map User & Roles:** the user with administrative roles can now list the list of federated users (*) and assign roles to users according to the role they have to play inside the participant agent

ACV - Domain 2 - Publish and consume resources

To share the provider of the data, application or infrastructure offering needs to make its offering available and findable for the interested consumers. To this end, the provider needs to describe its offering in the form of metadata (called Self-Description) and make it available in a central catalogue. This catalogue needs to provide appropriate functionality for the consumer to find his desired data, application or infrastructure offerings. For the consumption of the offerings are provided functionalities to negotiate a binding contract with validation of the access policy (control plane) as well as the technical consumption, e.g., the file transfer for data or the triggering of infrastructure deployment in the case of infrastructure.

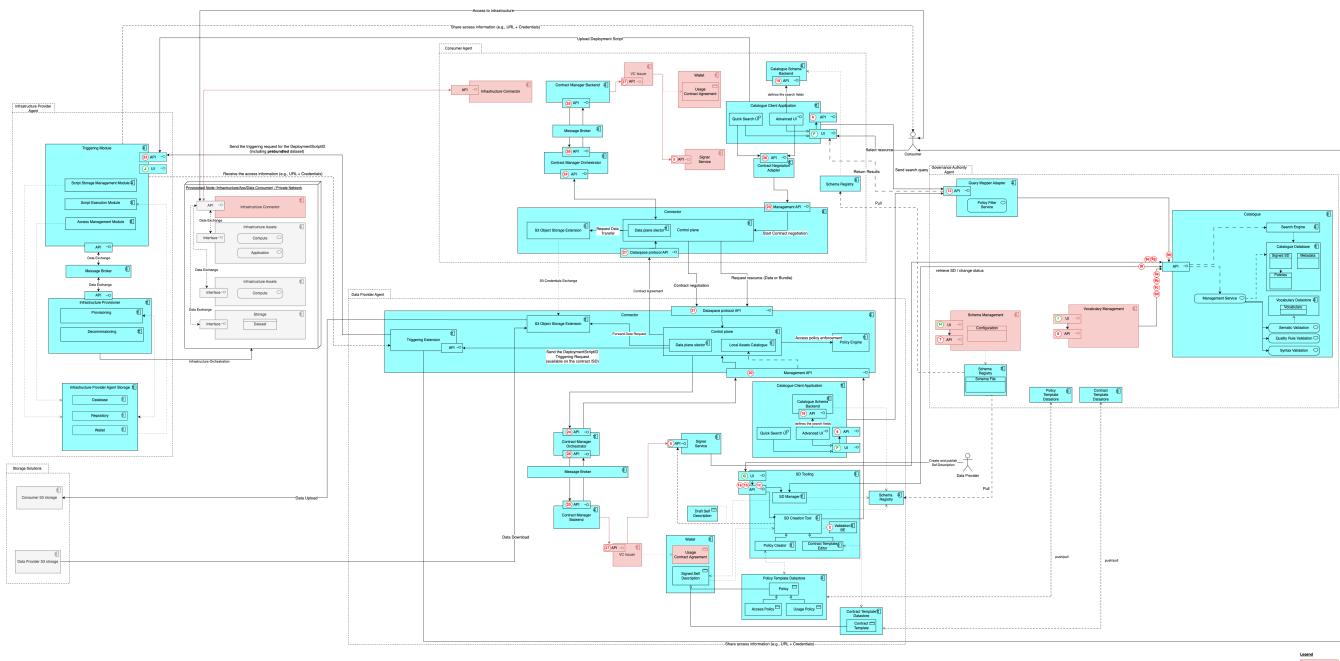
The static view diagram illustrates the structural organisation of components involved in the domain, segmented into four types of agents (Governance Authority, Consumer, Data Provider and Infrastructure Provider), showcasing the roles each plays in the domain.

As per the legend, components highlighted in red are foreseen to be part of Simpl-Open but are not part of the MVP, while components highlighted in grey are external to Simpl-Open.

The red numbers on the diagram help to correlate APIs with their definition which can be found in the Interfaces section, while the green letters makes the link with the User Interfaces which can be found in the same Interfaces section.

To keep the diagram lighter, following components (and their relations) have only been represented within the Data Provider agent but are in reality also part of the typical deployment of an Infrastructure Provider agent:

- Connector
 - Contract Manager Orchestrator
 - Contract Manager Backend
 - Signer Orchestrator
 - Signer Async Adapter
 - Signer Backed
 - VC Issuer
 - Wallet
 - SD Tooling
 - Catalogue Client Application
 - Schema Registry
 - Policy Template Datastore
 - Contract Template Datastore



Schema Management

- The **Schema Management** component represents the Metadata Description building block, enabling the Governance Authority to define the structure of self-descriptions. Using a UI or API, the Governance Authority can establish properties, data types, constraints, and controlled vocabularies that apply across resources (datasets, applications, infrastructure). The resulting schema configurations are automatically transformed into semantic files and managed within the Schema Registry, ensuring the Provider Node has access to the most current schema standards for generating self-descriptions in compliance with governance protocols.

Schema Registry

- The **Schema Registry** functions as a central repository and management interface for schemas created by the Governance Authority. These schemas, represented as ontologies and structured schema definitions, are actively managed to provide consistent standards across resource descriptions. Serving as an application component rather than a simple data storage element, the Schema Registry facilitates regular synchronisation with the Provider Node, ensuring that providers always have access to the latest schema standards needed for creating compliant self-descriptions.
- The Schema Registry is used by the catalogue client application to enable semantic consistency by defining and validating the terms used in self-descriptions and search fields. The Search Client uses the schema to define the search fields for the advanced search. This automatic form generation helps prevent ambiguous searches and ensures users can only search for terms recognised within the data space.

Catalogue

- Operating on the Governance Authority node, the **Catalogue** component functions as the central publication point for signed self-descriptions. It includes secure API functionalities for publishing, querying, and managing self-descriptions. After publication, the self-description becomes accessible to potential consumers via the Catalogue's API. The Catalogue also manages the status of self-descriptions and facilitates seamless access to information and metadata stored in the system's databases.
- When a search request is made via the Catalogue Client Application, the Catalogue's Search Engine processes the request, taking into account the filters and parameters provided by the Policy Filter Service and Adapter Component. This ensures that the search results returned to users are both relevant and compliant with defined policies.
- The Catalogue component also works closely with the Schema Registry to ensure semantic consistency across searches.
- The Catalogue component contains:
 - Catalogue Database** - The catalogue database is one or multiple databases that persist the published Self-Descriptions.
 - Search Engine** - The search engine indexes the entries in the catalogue database and allow for an performant search
 - Vocabulary Datastore** - The vocabulary datastore contains the loaded ontologies and schemas of the catalogue used for the semantic validation
 - Management Service** - The management service allows to perform several operation on the self-description, for instance the revocation of a Self-Description.
 - Syntax Validation Service** - The Syntax Validation Service checks the syntax of the Self-Description before publication
 - Semantic Validation Service** - The Semantic Validation Service checks the semantic of the Self-Description before publication. In detail it performs both an validation of SHACL Constraints and checks if the Self-Description complies with the ontologies in the catalogue.
 - Quality Rule Validation Service** - The Quality Rule Validation Service checks the quality of the Self-Description before publication. It checks if all mandatory quality rules are fulfilled and uses the recommended quality rules to calculate the quality score for the Self-Description.



Remark on Catalogue Deployments

In the current architecture view the catalogue is depicted as a single component, but yet a different schema is used for each type of resource (data, application and infrastructure). The catalogue might thus be deployed multiple times (e.g.) for testing purposes. The way this is deployed is subject to change. In the future the catalogues (data, infrastructure and application) may be kept in a single component deployment and can be separated by the different schemas.

According to DataSpace Protocol (DSP) specification each implementation of a connector has to provide a local assets catalogue instance providing all registered service offerings (asset) and usage contract offerings of this provider. Hence as a prerequisite to adding/updating a resource this service offerings (assets) has first to be registered at the connector. There the contract negotiation id to start contract negotiation will be created. This id is crucial for self description to provide any customer the link to start contract negotiation.

Query Mapper Adapter

- The **Query Mapper Adapter** component functions as an intermediary, translating user-defined search parameters into a format compatible with the Catalogue's database query language. These translation capabilities allow users to perform complex searches without needing to know the technical specifics of the database's query language, making it easier for users to interact with the Catalogue in a secure and user-friendly manner.

Policy Filter Service

- The **Policy Filter service** dynamically enforces access policies on search queries. It applies the access control rules defined within each self-description, filtering search results based on the user's permissions.
- This service is integrated in the Query Mapper Adapter component to embed policy-based filters into search queries before they are sent to the Catalogue. This integration ensures that all queries reflect the necessary governance controls, restricting access to authorised users and ensuring that sensitive information remains protected. In this way, the Policy Filter Service works as an invisible layer of security that ensures compliance while providing authorised access to the appropriate search results.

Message Broker

- Certain processes (e.g. publishing to catalogue, provisioning the infrastructure resources) are designed to be asynchronous. The role of the **Message Broker** is to facilitate these asynchronous processes.

SD Tooling

- Located on the Provider Node, the **SD Tooling** component enables providers to define self-descriptions for their resources by leveraging schemas from the Schema Registry. This ensures each self-description adheres to predefined properties and constraints. The SD Tooling Component supports both UI and API methods, providing flexibility to providers. It works in tandem with the Policy Creator and Contract Template components, allowing providers to incorporate policy and contract terms directly into self-descriptions.
- The SD Tooling component contains:
 - **SD Manager** - The SD Manager allows the user to manage his published Self-Description, for instance triggers the revocation
 - **SD Creation Tool** - The SD Creation Tool supports the provider in the creation of the Self-Description of their resources, by providing a generated frontend from the schema with the correct property fields.
 - **Policy Creator** - The **Policy Creator** component enables the creation and management of Access and Usage Policies for resources. Access Policies determine the accessibility of a resource, while Usage Policies outline the permissible uses and monitor the extent of usage to support billing based on consumption. These policies are serialised into a standardised format to ensure consistent application and interpretation across components. Integrated into the Self-Description, they contribute to a governed, comprehensive resource description.
 - **Contract Template Editor** - The Contract Template Editor enables the creation and the customisation of contract templates linked to resources in self-descriptions. These templates, once created, are stored in the Contract Template Datastore.
 - **Validation BE** - The Validation BE performs syntax validation for the Self-Description on the provider side before they are published to the catalogue

Policy Template Datastore

- The datastore contains templates of the policies that can be used as a blueprint to describe the access and usage policies for a resource

Contract Template Datastore

- The **Contract Template Datastore** stores Contract Templates to ensure consistent application of contract terms, which are later accessible to consumers during resource negotiation and access stages.

Signer Service

- The **Signer Service** component manages the digital signing of self-descriptions, ensuring their authenticity and integrity. Upon completion, the self-description is signed using the provider's private key to verify identity and prevent tampering. Once signed, the self-description is ready for distribution and is published to the relevant Catalogue component for broader access. This service is crucial for establishing trust between providers and consumers.
- The **Signer Service** component provides cryptographic signing capabilities for contracts, ensuring non-repudiation and authenticity. This component validates the identity and integrity of each contract, instilling confidence in the security of agreements.

Catalogue Client Application

- The **Catalogue Client Application** is the primary interface through which users interact with the Catalogue. It presents search fields and options to users, which in case of advanced search are defined by the schema.
- The **Catalogue Client Application** then sends the policy-filtered queries to the Catalogue Component via the Adapter Component. After receiving results from the Catalogue, it presents them in a structured format, ensuring that users can easily navigate and interpret the returned self-descriptions and metadata.
- The **Catalogue Client Application** component contains:
 - **Catalogue Schema Backend** - This Component transforms the schema definition automatically to front end files that are used to generate a custom made frontend to define the Self-Description
 - **Quick Search UI** - This UI allows the consumer/provider to perform a Quick Search on the respective Catalogue
 - **Advanced UI** - This UI allows the consumer/provider to perform a Advanced Search on the respective Catalogue

Contract Negotiation Adapter

- The **Contract Negotiation Adapter** component is requesting an Offering from the Provider. This Offering is returned with the Offering ID and respective usage & access policies.
- Once the user accepts the conditions (usage & access polices) the **Contract Negotiation Adapter** is building the request to start the Contract Negotiation on the **Connector** and retrieve the Status of the Contract Negotiation.

Connector

- The **Connector** component registers each resource (dataset, application, or infrastructure) as an asset within the data space, associating policies and contracts with each asset. It also provides controlled endpoints for each resource, playing an intermediary role in the contract negotiation process by leveraging the policies and contract templates associated with the resource. This enables the management of contractual relationships between providers and consumers. The connector functions also as a gateway for secure data exchange and ensures that policies are enforced during data consumption. It is responsible for enforcing security protocols and managing policies that govern access to the data (simple dataset or bundle).
 - The **Connector** component is implementing the data space protocol and contains the following sub-components:
 - **Control Plane** - The control plane of the connector acts as a state machine, overseeing the various states and transitions specified in the Contract Negotiation Protocol. It ensures that all agreements between the data provider and consumer are finalised before any data transactions take place. The Control Plane at the Provider side includes the Local Assets Catalogue component. An Asset is the primary building block for resource sharing, it represents any data or API endpoint that can be shared. Assets are descriptors that are loaded into EDC via its Management API during the registration phase performed before uploading a resource to the catalogue. In case of a bundle, it is the URL that triggers the deployment script which will deploy the requested infrastructure and application.
- The control plane to perform its functionalities interacts with the Management API, the Protocol API and the Policy Engine.

- **Data Plane** - The Data Plane enables the data exchange based on the transfer protocol which will only take place in case the contract negotiation protocol has successfully established a contract. This second part is controlled by the control plane and performed by the data plane. The data plane component, which consists of an extension of the connector, manages the actual data exchange, ensuring that data flows securely from the provider's source to the consumer's specified destination, aligning with the agreed-upon terms of the contract. In the scenario of a bundled infrastructure, data and application, the role of the data plane component is performed by the infrastructure orchestrator which is responsible for retrieving the deployment script ID from the Asset of the resource to be used and triggering the execution of the script on the infrastructure provider. Once the provider completes the deployment, it will return the access details for the newly created environment. These details will then be forwarded to the user, who will use the provided information to access the infrastructure directly.
- **Management API** - The Management API is a RESTful interface for client applications to interact with the control plane.
- **Dataspace protocol API** - The Dataspace protocol API is a RESTful API interface that is used for the contract negotiation protocol.
- **Policy Engine** - The Policy Engine is crucial in making decisions based on the policies tied to the requested resource. The policy engine is able to perform this operation because the policies are registered and linked to the registered assets (Assets component in the Control Plane). This allows the policies to be retrieved at this moment and the necessary checks to be carried out. This component evaluates whether all policy requirements are met, and if they are not, they can halt the process to prevent unauthorized access.
- **Triggering Extension** - The triggering extension will send the DeploymentScriptID and the email address of the consumer, to the Infrastructure Triggering Module, at the time of finalising a contract agreement. This will result into provisioning of the infrastructure resources and deployment of the applications on that resource.
- **S3 Object Storage Extension** - Can transfer datasets from the S3 Object Storage of the Data Provider, to the S3 Object Storage of the Data Consumer, at the time of finalising a Data Transfer Contract.

Contract Manager (Orchestrator and Backend)

- The **Contract Manager** coordinates with the **Verifiable Credentials Issuer (VC Issuer) Component**, **Signer Component**, and **Wallet Component** to integrate contract validation, issuance, and storage functionalities. It also stores contracts for billing and record-keeping purposes, centralising key contract-related data.
- **MVP Note:** For the Minimum Viable Product (MVP), interactions with the VC Issuer Component, Signer Component, and Wallet Component are streamlined through a single stub interface. Additionally, contract storage and Wallet emulation are consolidated into a single database, simplifying the initial implementation.

Verifiable Credentials Issuer (VC Issuer)

- The **VC Issuer** component securely issues and manages verifiable credentials, providing transparent and reliable validation of usage contracts. It relies on the **Signer** component to apply cryptographic signatures to contracts, ensuring data integrity.
- Signed usage contracts are stored in the **Wallet** component for secure access, facilitating a robust and trustworthy credential management process.

Wallet

- The **Wallet** component serves as a secure digital repository for storing, managing, and presenting verifiable credentials (VCs). This "digital wallet" enables providers to securely manage and share their credentials, ensuring compliance with contractual requirements while facilitating efficient access to validated information.

Triggering Module

- The **Triggering Module** component is responsible for adding, managing and executing the deployment scripts, and finally sharing the access data. The triggering module is made of three submodules:
 - **Script Storage Management submodule (Accessible via the API and the Infrastructure Deployment Script Management UI)** : That is responsible for adding and managing the deployment scripts. It contains the following functions:
 - **Add Script**: Enables users to add deployment scripts to the local repository and database, ensuring the scripts are accessible for future provisioning tasks. This function also performs security checks to prevent uploading of malicious scripts and files.
 - **Remove/Invalidate Script**: Manages the removal or invalidation of outdated scripts from the repository and database.
 - **Script Execution submodule**: When an API call requests the triggering of the deployment script, this module initiates the execution process. Its functions are:
 - **Retrieve Deployment Script**: Retrieves deployment scripts from the repository, allowing the Infrastructure Provisioner to execute the necessary steps for the resource provisioning and software deployment.
 - **Validate Deployment Script**: Generates and compares a hash of the retrieved script from the repository to the hash that was stored in the database at the time of storing the script, to check for integrity and authenticity, confirming the script is secure and unaltered.
 - **Trigger Execution**: Communicates with the Infrastructure Provisioner via a message broker to initiate the provisioning process.
 - **Access Management submodule**: When the provisioning is done, shares the access information such as endpoints and credentials with the consumer:
 - **Retrieve and Share Access Data**: Obtains access credentials and details from the Infrastructure Provisioner, making them available for distribution to the necessary stakeholders.
- The **Triggering Module** exposes its functionality via an API, enabling other SIMPL-Open Agent modules to interact with it as needed. After triggering the execution of the deployment script, the triggering module listens for provisioning completion events from the Infrastructure Provisioner to confirm successful deployment and share the access data.

Infrastructure Provisioner

- The **Infrastructure Provisioner** component is an asynchronous service that orchestrates the actual provisioning of infrastructure resources and potential deployment of the applications and datasets (in case they are a part of the deployment script). Upon receiving a deployment trigger from the Triggering Module, this component follows several steps to ensure resources are provisioned, configured, and made accessible. This module contains two submodules for provisioning and decommissioning:

- **Provisioning** sub-component: Provisions the infrastructure resources, creates/grants access to them, and runs post-configuration processes to set policies and to deploy applications.
 - **Execute Deployment Script:** Runs the deployment script received from the Triggering Module, provisioning resources such as compute instances, storage, or other assets.
 - **Set Policies:** Defines infrastructure specific usage and access policies to govern resource usage, aligning with predefined rules on the deployment script to control who can access the provisioned infrastructure resource.
 - **Create Access Information:** Generates and provides access credentials and endpoints, allowing authorized users to interact with the infrastructure.
 - **Post Configuration:** Can deploy applications and load datasets on the provisioned infrastructure resource.
 - **Share Access Data:** Returns the generated access information back to the Triggering Module / Access Management, so the information can be shared with the consumer.
- **Decommissioning** sub-component: It will decommission the infrastructure asset based on the criteria set by the business (e.g., end date of the contract.) The two main functions are:
 - **Pre-decommissioning:** Initiates the pre-set decommissioning configurations such as notifying the consumer and making snapshots/backups.
 - **Access Revocation:** Revokes user access, if applicable, and triggers the final termination process.
- This **infrastructure provisioner** is not directly exposed via the public API but through the processes of the triggering module.

Infrastructure Provider Storage

- The **Infrastructure Provider Storage** component houses both a **Database** and a **Repository** to store deployment scripts. The Storage component can supports versioning, audit trails, and controlled access, thus facilitating compliance and security in deployment operations.

Following sub-sections contain dynamic views that each present how a subset of above-described application components are used to satisfy different (parts of) business processes:

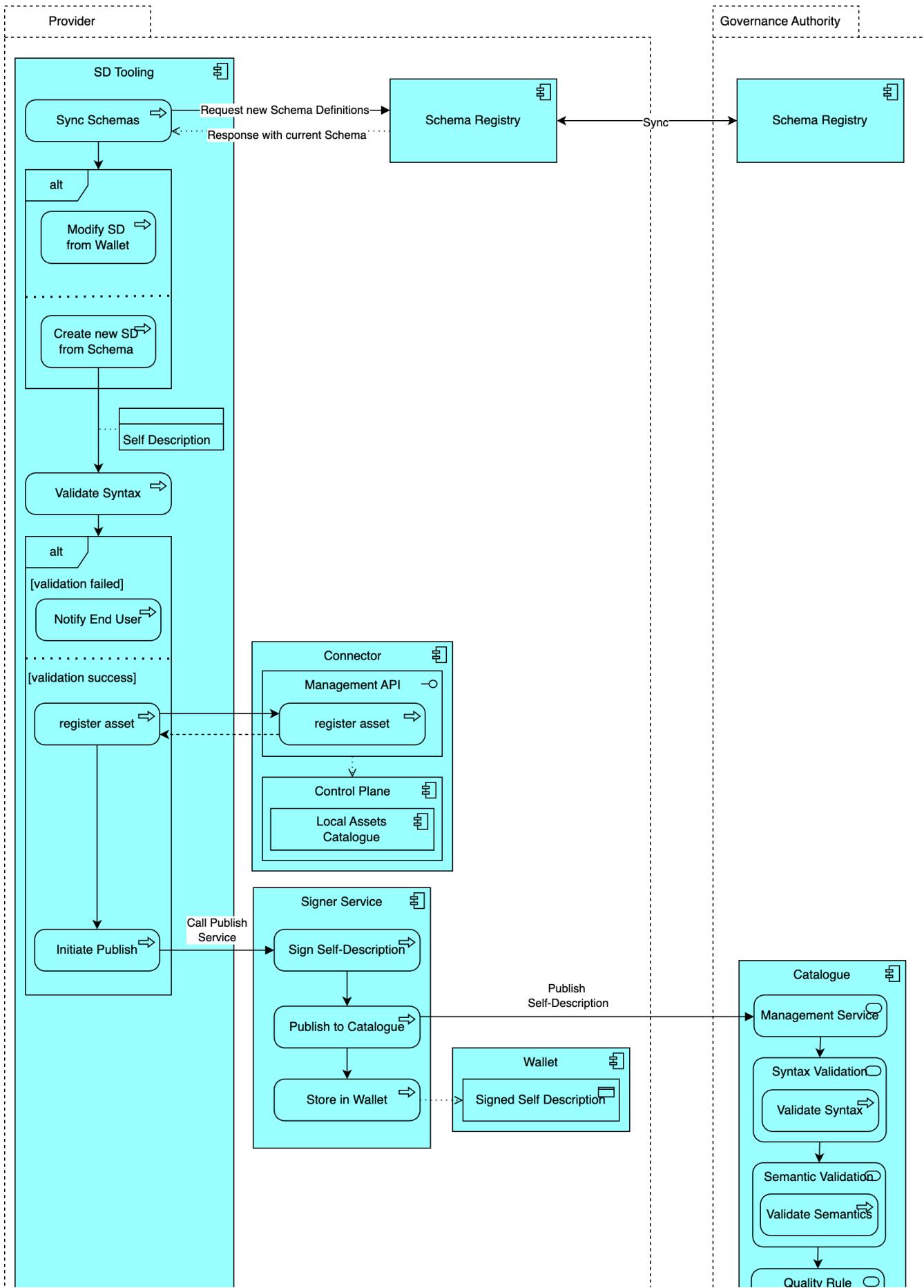
1. The first part is that the provider needs to describe his resource using a predefined schema that when tailored to the resource at hand becomes a self-description. Next the provider need to make this self-description (SD) available for potential search. This process is described in detail in ACV Dynamic - BP 05A - Add or Update Resource (Publish) on Catalogue. In simple terms, this publication process consists of:
 - a. The provider describes his offering using the SD Tooling, how the description should look like is defined in the schema of the self-description
 - b. The provider registers the SD as an asset in the connector. The asset is composed by a subset of the metadata present in the SD, only the one that will be necessary afterwards during the consumption
 - c. The provider signs the Self-Description with his credentials to proof that he is the owner and to make the Self-Description tamper-proof
 - d. Finally, the provider publishes the Self-Description to the central catalogue on the Governance Authority Node so the consumer can search for it. The Governance Authority checks automatically if the Self-Description is correct according to the syntax, semantics and quality.
2. To update a Self-Description consist of at first revoking the old version of the Self-Description and publishing a new version, for detail see ACV Dynamic - BP 05A - Add or Update Resource (Publish) on Catalogue.
3. The third process is that the consumer searches the catalogue for dataset, application or infrastructure offering. The consumer defines the search terms in the search client app and the catalogue on the governance authority agent executes the search in the catalogue. This is described in detail in ACV Dynamic - BP 06 - Search on Catalogue (Infrastructure, Data, Application). The process consists of:
 - a. The Consumer (or provider) uses the search client app to write his search terms. We allow for two different ways of searching quick search or advanced search.
 - b. The consumer calls a service in the Query Mapper Adapter on the Governance Authority. This service maps the search terms onto executable queries for the catalogue and also ensures that the consumer can only see the offerings that allow it by enforcing the policy.
 - c. The query is executed by the catalogue itself and the results are returned to the consumer (provider).
4. The fourth part consists of the consumption which is declined in 3 different resource consumptions: data direct download, Infrastructure consumption, and data consumption through an application (also called bundle for the MVP scope):
 - 3a. The first type of consumption is the direct consumption of Dataset ACV Dynamic - BP 09A - Consumer consumes a data resource from the provider. The consumer has found the offering that he wants from the central catalogue and next he wants to consume the data. this process in simple terms consists of three sub-processes:
 - a. The consumer uses the connector to establish a contract with the provider (described in detail in ACV Dynamic - BP 07A - Establish a usage contract agreement)
 - b. The control planes perform the contract negotiation between the connector of the consumer and provider (also includes the enforcement of the policies)
 - c. The data plane is used to transfer the data from provider to consumer
 - 3b. The second consumption is the infrastructure consumption ACV Dynamic - BP 08 - Consumers select and use an Infrastructure Catalogue Resource from the Infrastructure Provider. The consumer finds the offering in the central catalogue and then performs the request for its consumption.
 - a. The consumer uses the connector to establish a contract with the provider (described in detail in ACV Dynamic - BP 07A - Establish a usage contract agreement)
 - b. The control planes perform the contract negotiation between the connectors of the consumer and provider. The control plan also includes the enforcement of the policies.
 - c. The infrastructure provider retrieves, validates and triggers the deployment script of the infrastructure offering
 - d. The infrastructure provider retrieves the access data for the infrastructural resource and shares them with the consumer
 - 3c. Besides the direct consumption of a dataset we also want to support the data consumption through a processing service over an application ACV Dynamic - BP 09B - Consumer receives data processing service over a dataset via an Application. The steps are a bit more complex due to the need for infrastructure to host the application and dataset:

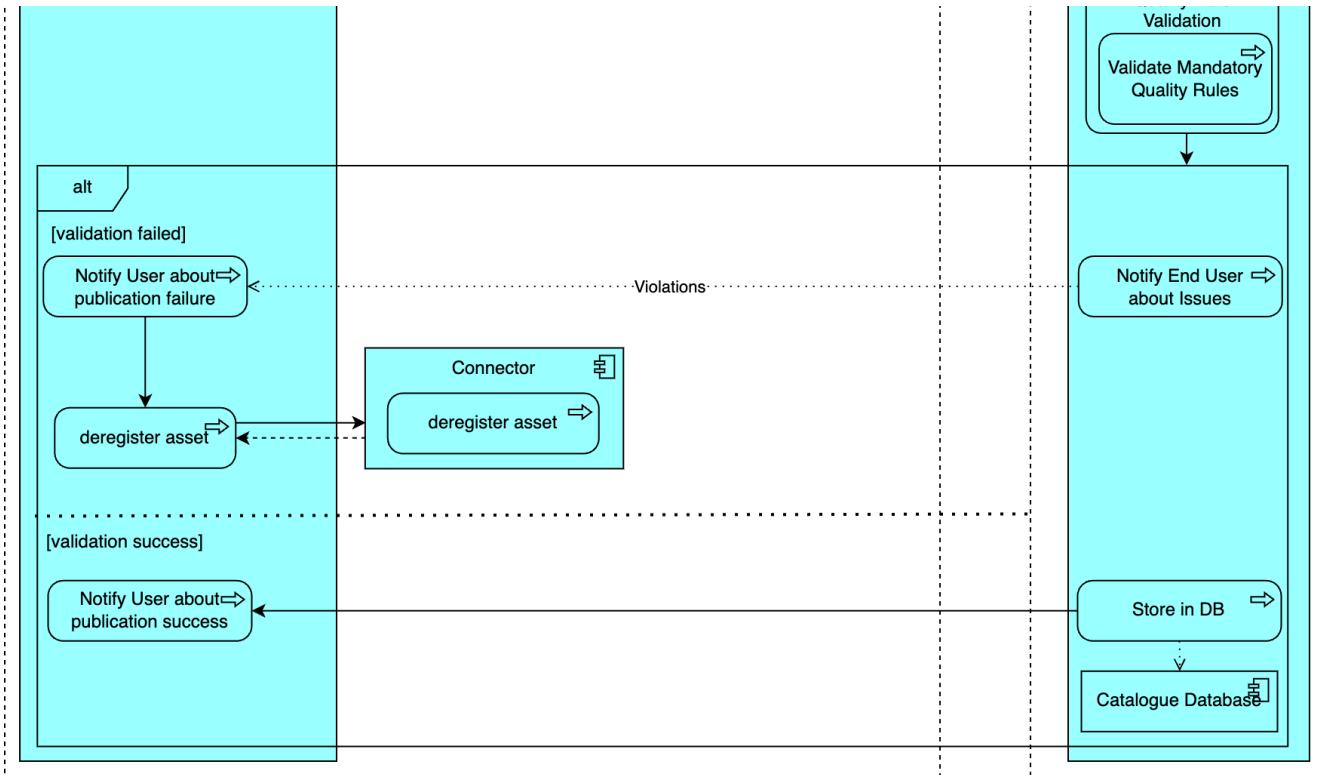
- a. The uses the connector to establish a contract with the provider (described in detail in ACV Dynamic - BP 07A - Establish a usage contract agreement)
- b. The control plane is negotiated between the connector of the consumer and provider (also includes the enforcement of the policies)
- c. An infrastructure is provisioned for the consumption (ACV Dynamic - BP 08 - Consumers select and use an Infrastructure Catalogue Resource from the Infrastructure Provider), and the dataset and application are installed on that infrastructure
- d. The consumer gets (restricted) access to this infrastructure

ACV Dynamic - BP 05A - Add or Update Resource (Publish) on Catalogue

Define and Publish Self-Description

This process outlines how a self-description can be defined and subsequently published in the Catalogue. Certain fields within the self-description link to other resources, which therefore need to be created beforehand. For instance, an infrastructure offering requires a deployment script to be added in advance so that it can be referenced within the self-description.

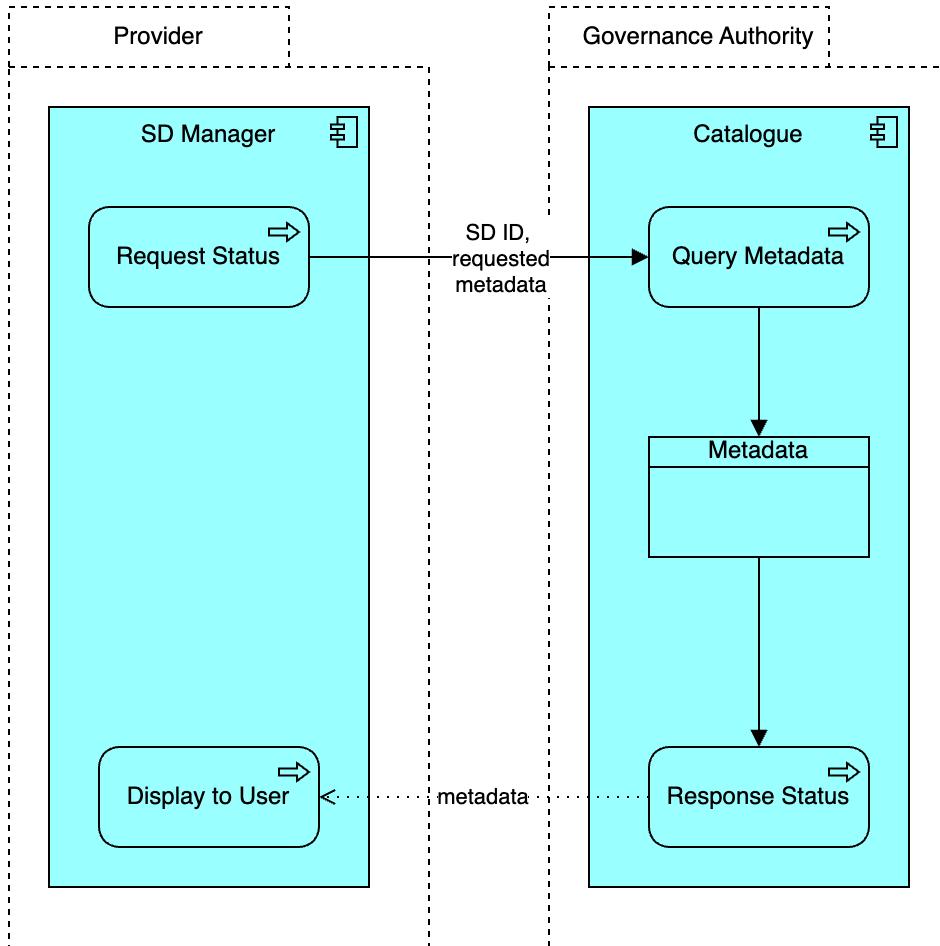




- Schema Synchronisation :** The SD Tooling component on the Provider side initiates a request for schema definitions from its local Schema Registry, which is kept in sync with the one on the Governance Authority node. This ensures consistent schema access and alignment across participants, supporting unified self-description formats in the data space. The retrieved schema definitions are stored in a local Schema Datastore on the Provider's end, ensuring quick access and version control.
- Create Self-Description :** Providers can create a new self-description or modify an existing one through the User Interface. This interface allows them to fill in necessary fields or use a previously stored self-description template.
- Syntax Validation :** The Syntax Validation component within SD Tooling checks the initial structure of the self-description to confirm it meets the required format. While primarily focusing on the form of the self-description, this step also checks for basic schema compliance. If any issues are found, the provider is prompted to make corrections before proceeding.
- Registering Self-Description :** Following syntax validation, the self-description is directed to the Connector component, where it is registered as an asset. This registration is critical for linking the self-description to a specific connector instance, enabling controlled access for consumption by the consumer.
- Signing and Publication :** The Signing/Publication Service manages the integrity and authenticity of the self-description. It signs the document using the Provider's private key to prevent tampering, and then publishes the signed self-description to the Catalogue. A copy of the signed self-description is also stored locally in the Provider's Wallet for record-keeping purposes. The Wallet maintains a history of signed copies, with any necessary purge or retention policies applied to manage storage effectively. These policies should specify when older records are archived or deleted to optimise space and meet governance standards.
- Semantic Validation :** After publication, the Catalogue on the governance node initiates Semantic Validation. This step checks that the self-description adheres to the data space's vocabularies and ontology standards, ensuring semantic consistency.
- Quality Check :** The Catalogue also performs a Quality Rules check to verify that the self-description meets all mandatory quality standards. If any semantic issues are identified, the End User is notified to address specific violations. If the self-description passes all checks successfully, the End User receives a confirmation notification, indicating that the resource is now ready for publication within the data space.
- Database Storage :** Upon passing all validations, the self-description is stored in the Catalogue's Database along with its associated metadata, making it discoverable and accessible to other participants in the data space.

Retrieve SD Metadata

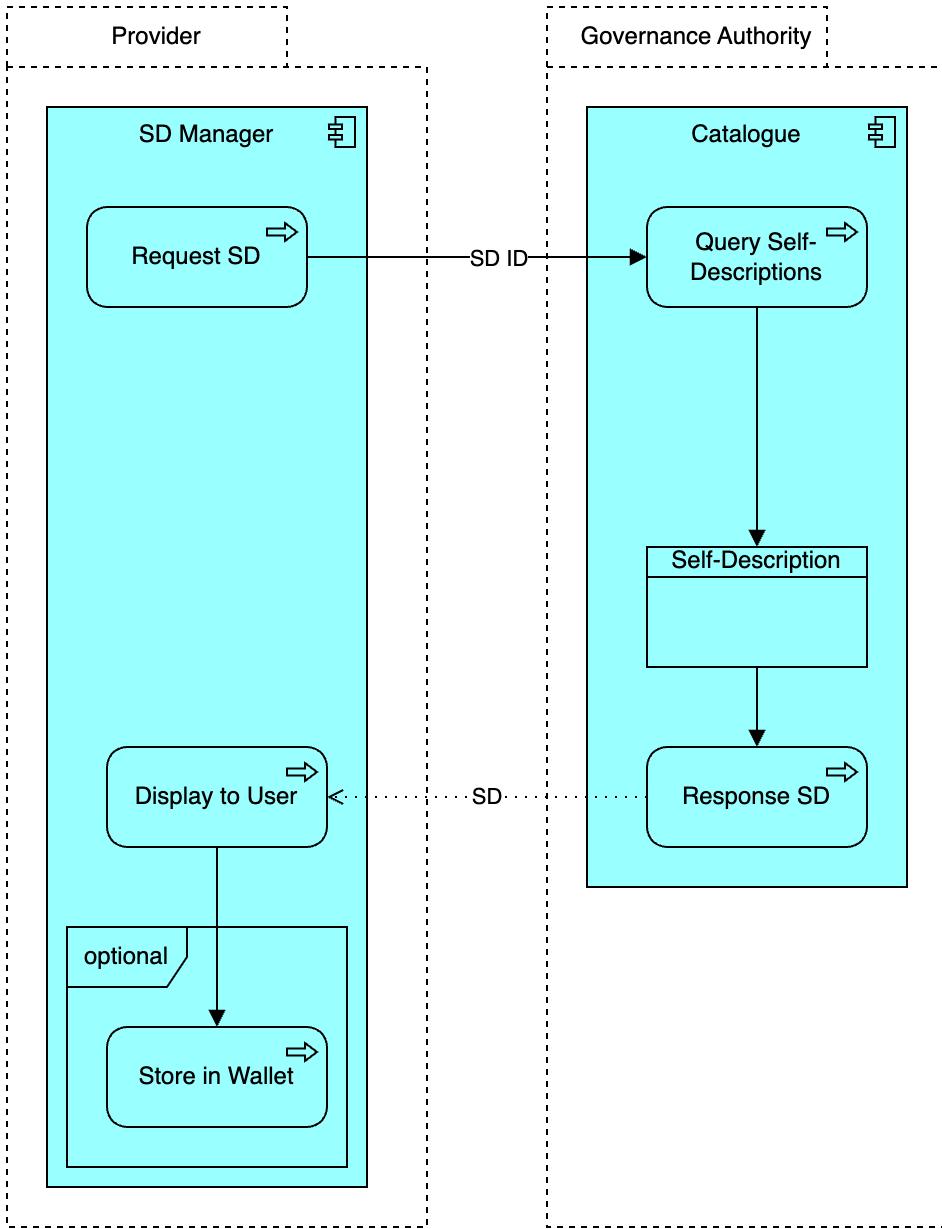
This process illustrates how the metadata (such as status) of a self-description (SD) can be retrieved by a Provider. The different possible statuses for a self-description are outlined in the Gaia-X Federation Services documentation (40.3 Product Constraints) [here](#).



1. **Initiate Metadata Request:** The Provider initiates a request to retrieve metadata associated with a specific self-description. This request includes the unique identifier of the desired self-description and is sent to the **Catalogue** on the **Governance Authority Node**.
2. **Query Metadata:** Upon receiving the request, the Governance Authority Node processes it by querying its **Metadata Database** to locate the requested metadata. This step ensures that the metadata aligns with the unique identifier provided.
3. **Return Metadata:** After locating the requested metadata, the **Catalogue** prepares a response containing the metadata details. This ensures the requested information is available for consumption.
4. **Display to User:** The **SD Manager** receives the metadata and presents it to the Provider's end user, allowing them to view details like the status of the self-description.

Retrieve Full Self-Description

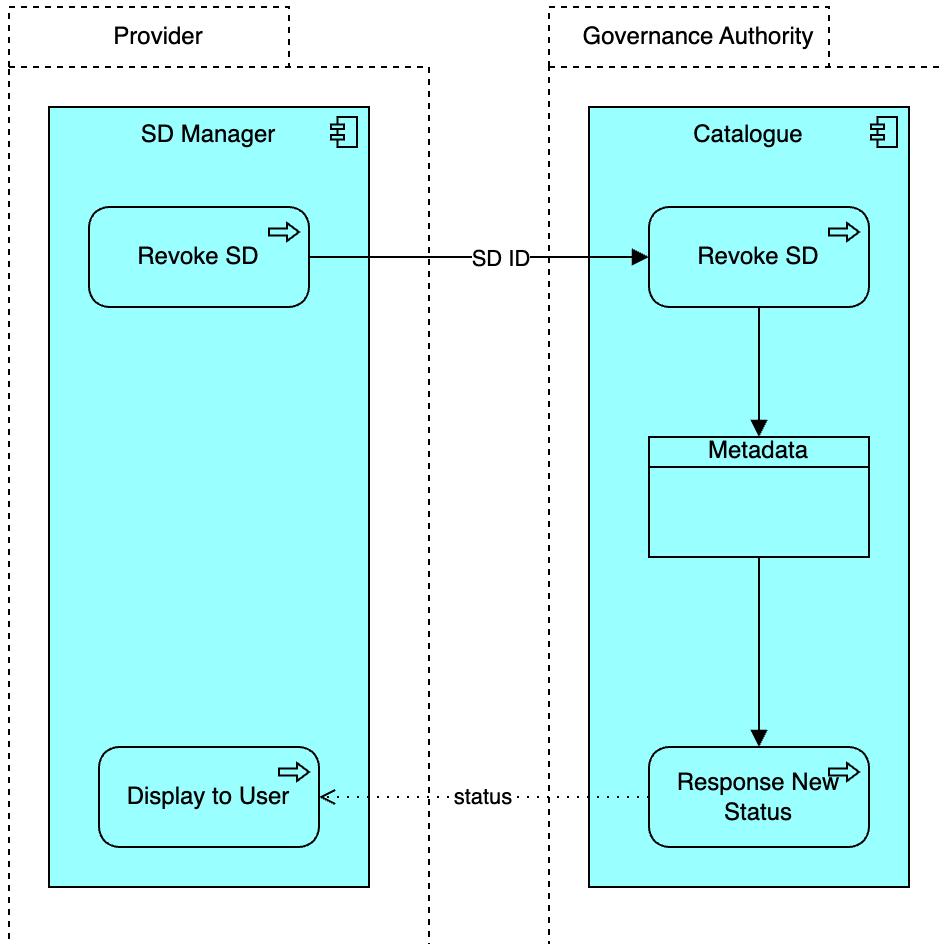
This sequence describes the steps taken by a provider to retrieve a complete self-description (SD) for a resource.



- Initiate SD Request:** The process begins with the provider using the **SD Manager** to initiate a request for a specific self-description by sending its unique identifier (SD ID) to the **Management Service** hosted on the **Governance Authority Node**.
- Query Self-Descriptions:** After processing, the **Management Service** queries the **Self-Description Database** to retrieve the detailed self-description associated with the given SD ID.
- Respond with Self-Description:** Once the full self-description is retrieved, it is sent back to the provider through the **Response SD** component. This self-description includes all necessary metadata and resource information.
- Display to User:** The SD Manager on the provider node displays the retrieved self-description to the provider's end-user through its User Interface.
- Optional Storage in Wallet:** Optionally, the retrieved self-description can be stored in the provider's **Wallet** for local record-keeping or offline access.

Revoke SD

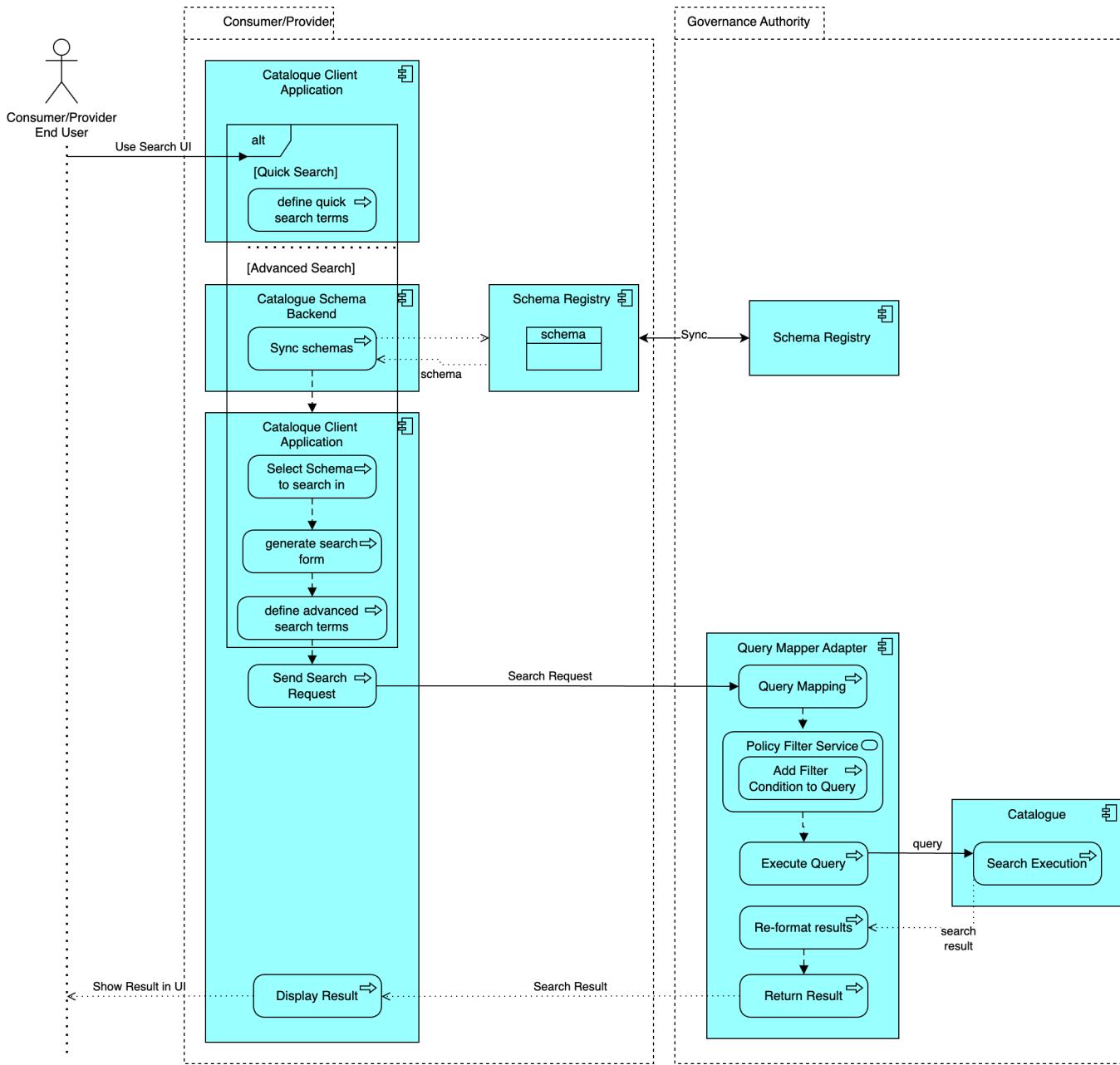
This process outlines how a provider can revoke a Self-Description (SD) in the system, with possible statuses detailed in the [Gaia-X documentation](#) (40.3 Product Constraints).



- Initiate Status Change:** The provider, through the **SD Manager** on the Provider Node, initiates a request to revoke a specific SD by sending the SD ID to the **Catalogue** on the **Governance Authority Node**.
- Revoke SD:** The system then revokes the SD in the **Catalogue** database to reflect the new status for the Self-Description.
- Response and Display:** The **Management Service** confirms the status update by returning the new status to the **SD Manager**. If the user interface (UI) is used, the updated status is displayed to the provider end user.

ACV Dynamic - BP 06 - Search on Catalogue (Infrastructure, Data, Application)

The process describes the end user searching for a resource in the catalogue. The end user can either use the quick search or the advanced search. For the advanced search, it is a prerequisite that the local schema registry of the provider/consumer is synced manually with the central schema registry of the governance authority. The search request is sent to the catalogue. The query mapper translates the query input to the related database query language and adds the filters based on the access policies related to the user performing the request. The search engine executes the search queries and returns the results. The result is then displayed in the end User's search Client.



1. User Search Request Initiation

The user initiates a search request through the **Search Client**. Here, the user enters the search criteria, which could include keywords, filters, or other parameters relevant to the desired resources (e.g., datasets or applications).

For the Advanced Search the form of the search is defined by the schema in the **Schema Registry**.

2. Policy Filter Service

The validated search request is forwarded to the **Policy Filter Service**. This service checks the user's access rights based on the policies defined in the **Policy Creator Component**.

By applying the relevant , the Policy Filter Service modifies the search query to restrict results to only those resources the user is authorised to view.

3. Query Translation by Adapter Component

The **Query Mapper Adapter** Component receives the policy-filtered search request and translates it into a query language that aligns with the Catalogue's database structure.

This step includes mapping the search parameters to the Catalogue's internal query schema and embedding any access restrictions set by the Policy Filter Service directly into the query.

4. Catalogue Component Query Execution

The **Catalogue Component** receives the translated and filtered query from the Adapter. Within the Catalogue, the **Search Engine** processes the query by scanning its database, which houses all signed self-descriptions, metadata, and associated policies.

The Catalogue ensures that each self-description or metadata entry returned aligns with the access policies, ensuring compliance with data

governance standards.

5. Result Return to Search Client

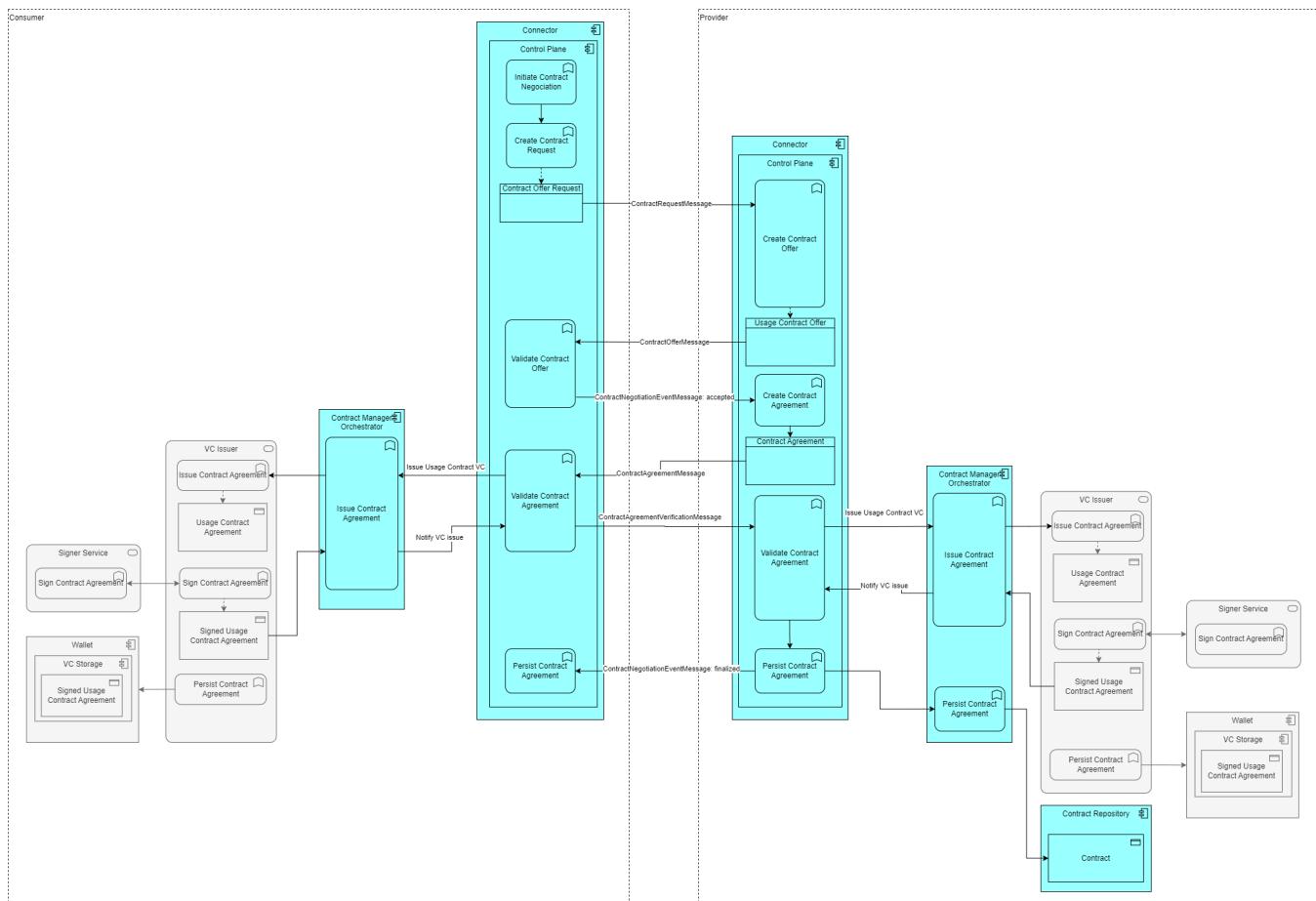
After processing the query, the **Catalogue Component** sends the authorised results back through the **Adapter Component**, which re-formats them for the Search Client's display needs. The Search Client then presents these results to the user in a structured format, along with relevant metadata to provide a comprehensive view of each item.

ACV Dynamic - BP 07A - Establish a usage contract agreement

This dynamic view for the "Establish Usage Contract Agreement" process captures the flow of interactions between various components involved in initiating, negotiating, validating, and finalising a contract agreement between a Consumer and Provider. The view is structured into four primary sections representing different roles: Consumer, Connector, Provider, and Governance Authority.

Preconditions:

- Consumer Discovery and Decision:** The Consumer must have discovered and selected the desired resource from the Dataspace's Catalogue, reviewed the associated terms and conditions within the Usage Contract template (Business Process - 06), and made the decision to consume the resource (Business Processes - 08, 09A, and 09B).
- No Existing Contract:** There must not be an existing Usage Contract in place that covers the specific resource and terms of the current consumption request.



- Initiating Contract Negotiation (Consumer to Connector):** The Consumer initiates a contract negotiation through the Connector's control plane by creating a "Contract Offer Request." This request is sent to the Provider's Connector, initiating the contract establishment process.
- Contract Offer Creation and Validation (Provider):** Upon receiving the Contract Offer Request, the Provider's Connector generates a "Contract Offer" and sends it back to the Consumer Connector for validation. The Consumer then reviews and validates this offer to ensure it meets their requirements.
- Agreement Formation and Validation (Bidirectional Communication):** If the Consumer accepts the offer, the Consumer Connector initiates the creation of a "Contract Agreement." This agreement is validated by both the Consumer and Provider's Connectors to ensure mutual

compliance. Once validated, both parties confirm the contract through Verifiable Credentials.

4. **Verification and Issue of Usage Contract VC:** The Provider invokes the VC Issuer to issue Verifiable Credential (VC) for the Usage Contract Agreement. This credential is signed by a signer service subsequently returned and stored securely within the VC storage of the Wallet for regulated access to usage terms. This is then repeated on the Consumers side to issue, sign, and securely store VC for the Usage Contract Agreement on the customer's side.
5. **Persisting Agreement (Wallet & Storage):** After the VC Usage Contract is signed and securely stored in the digital wallets of both the consumer and the provider, a copy of the contract (in a format to be determined, potentially a third VC or a traditional record) will be stored by the provider for future reference, such as billing and auditing purposes.

ACV Dynamic - BP 08 - Consumers select and use an Infrastructure Catalogue Resource from the Infrastructure Provider

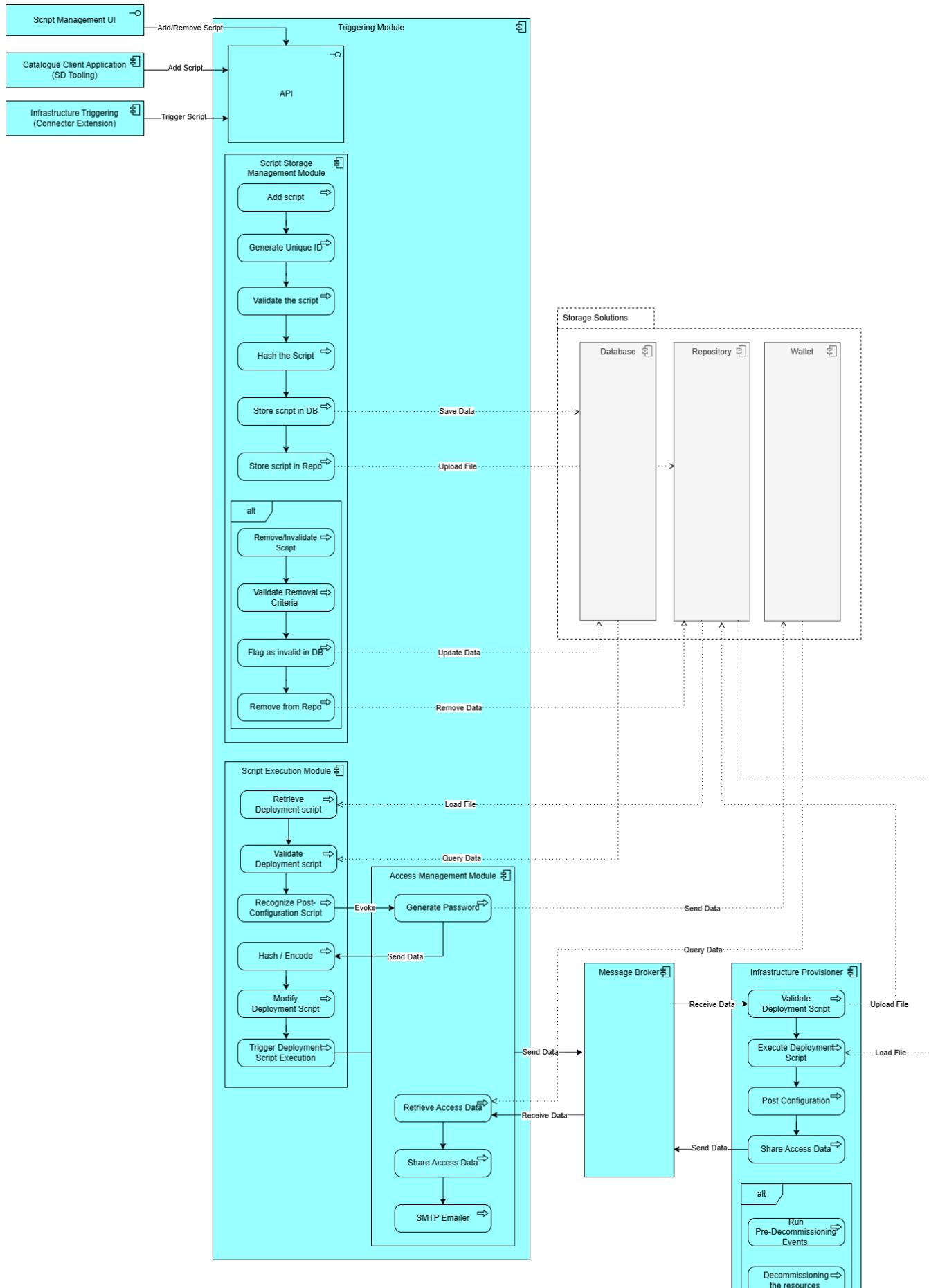
The dynamic view diagram illustrates the orchestrated interactions required to provision infrastructure resources, and to deploy applications on the provisioned infrastructure asset. This view focuses on the coordinated roles of the **Triggering Module**, **Broker**, **Storage** and **Infrastructure Provisioner**.

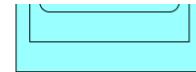
Preconditions:

- Infrastructure of the data space governance authority had been set up (agent deployed);
- Infrastructure of the infrastructure provider had been set up (agent deployed);
- Infrastructure service offering(s) had been listed on the catalogue (and therefore registered as connector assets), as per BP 05A;
- Infrastructure Consumer has been onboarded to the data space (as per BPs 03A and 03B);
- Infrastructure Consumer is authenticated and has been authorised;
- Main Infrastructure instance of the infrastructure consumer had been set up (agent deployed).

Triggering and Infrastructure Provisioner Modules

This process outlines how the deployment script can be added, removed, invalidated and triggered, using the Triggering and Infrastructure Provisioner Modules.



Legend

Storage Solutions on the SIMPL Agent

Triggering Module

1. **API:** The request to the triggering module API would be received either from the "script management UI" when a deployment script is being added or is being modified, or from other components of SIMPL such as connector extensions (at the time of contracting between two connectors, to send the Deployment Script ID, and other relevant information such as the Consumer Email, and trigger the execution of deployment script, which provisions the infrastructure resources and deploys apps asynchronously).
2. **Script Storage Management Module:** is the functionality of the backend, accessible via the API which also is available via the UI that relies on the API. Using this functionality, service providers (infrastructure, app or data) can store Deployment Scripts and receive a unique identifier (DeploymentScriptID) assigned to that specific script. At the time of adding the scripts, they are being validated to not contain malicious code. Scripts are added to a repository and a database at the same time, to have a mechanism to check their integrity in the future and at the time of retrieval.

Add Script

- **Generate Unique ID:** Assigns a unique identifier (DeploymentScriptID) to each script for tracking purposes.
- **Validate Script:** Ensures the script is free of malicious code. Scripts failing this check are rejected.
- **Hash the Script:** Generates a hash value for the script, which is stored in the database to enable future integrity checks.
- **Store Script in DB:** Saves the script's metadata and hash securely in the database, with protections against SQL injection attacks.
- **Store Script in Repo:** Stores the actual script file in a local repository for retrieval during execution.

Remove Script

- **Validate Removal Criteria:** Enforces predefined business rules, such as backup creation, before invalidating a script.
- **Flag as Invalid in DB:** Updates the script's validity status in the database to indicate it is no longer active.
- **Remove from Repo:** Deletes the script file from the repository, though its metadata remains in the database for audit or business purposes.

3. **Script Execution Module:** The module is responsible for handling deployment script retrieval, validation, and execution requests.

- **Retrieve Deployment Script:** When a request containing the DeploymentScriptID is received, the module retrieves the script from the repository.
- **Validate Deployment Script:** Checks the integrity of the retrieved script by generating a new hash and comparing it with the hash stored in the database. Integrity failures trigger errors, preventing execution.
- **Recognize Post-Configuration Script:** If the Crossplane configuration file contains a Cloud-init configuration section containing post provisioning configurations, it will be recognized, to be encoded to base64 (as described in the next steps, since it's required by Crossplane), after proper modifications (e.g., adding a public key or password that's generated by the access management module, as described in step 4. Access Management Module).
- **Hash / Encode:** Encodes the Cloud-init configuration (if exists) using Base 64. Hashes the randomly generated password by the Access Management Module (if exists) using SHA 256.
- **Modify Deployment Script:** Replaces the simple-text Cloud-init configuration by the base64 encoded version of it, which contains the added information such as the encrypted password or the public key.
- **Trigger Deployment Script Execution:** Sends the deployment script to the **Infrastructure Provisioner Module** via the **Message Broker**, ensuring asynchronous communication for scalability.

4. **Access Management Module:**

- **Generate Password:** When a request containing the DeploymentScriptID is received, the module retrieves the script from the repository.
- **Retrieve Access Data:** Checks the integrity of the retrieved script by generating a new hash and comparing it with the hash stored in the database. Integrity failures trigger errors, preventing execution.
- **Share Access Data:** Shares the endpoints, credentials and any information relevant for the provisioned instance (or deployed applications). For the MVP it relies on the SMTP emailer, and will be replaced by wallet solutions after the MVP.
- **SMTP Emailer:** Shares the access information with the Consumer, using the email address which was received during the triggering process.

Message Broker: The **Message Broker** facilitates communication between the **Script Execution Module** and the **Infrastructure Provisioner Module**.

Infrastructure Provisioner Module: The module is in charge of provisioning and decommissioning of the infrastructure resources and completing post-provisioning configuration tasks. Key steps include:

Provisioning

- **Validate Deployment Script:** Checks the script for syntax correctness and interpretability to avoid execution errors.
- **Execute Deployment Script:** Provisions infrastructure resources based on the script's configuration.
- **Post Configuration:** Completes additional tasks, such as setting policies, deploying applications, mounting or attaching storages and loading datasets as specified on the post configuration (Cloud-init) section of the deployment script.

- **Share Access Data:** Shares access information (such as endpoints) to the **Access Management Module**, via the **Message Broker**.

Decommissioning

- **Run pre-decommissioning tasks:** Such as making a snapshot of the resources, depending on the business requirements (yet to be clarified by Business).
- **Decommission:** Terminate/destroy the resources.

Storage Solutions: Consists of the **Database**, **Git-Based Repository** and **Wallet** ensuring secure storage and retrieval of deployment scripts and passwords.

- **Database:** Stores metadata and hashes for each script to facilitate integrity verification.
- **Repository:** Hosts the actual deployment scripts for retrieval during provisioning.
- **Wallet:** If a random password generation is necessary for the instance that is going to be provisioned, this password will be temporarily stored on the wallet (Hashicorp vault, in this case), until the provisioning is finalized and the password is going to be communicated to the consumer, and to be deleted from the wallet.

ACV Dynamic - BP 09A - Consumer consumes a data resource from the provider



This section is only describing the capabilities falling behind the scope of the MVP (December 2024) and will be enhanced at a later time. In particular it includes only the direct data download capability for data sharing.

The Data Provider is open to offering straightforward access to the dataset for the consumer. This access can be facilitated through a direct download, making the process simple and efficient. To ensure proper governance, a formal contract will be established between both parties. Since the data is downloaded, Simpli no longer has control over its usage, and therefore this contract will define and enforce legally binding usage policies as well as access policies. These measures will provide clarity and security for both the Data Provider and the consumer, safeguarding proper usage of the data.

Preconditions:

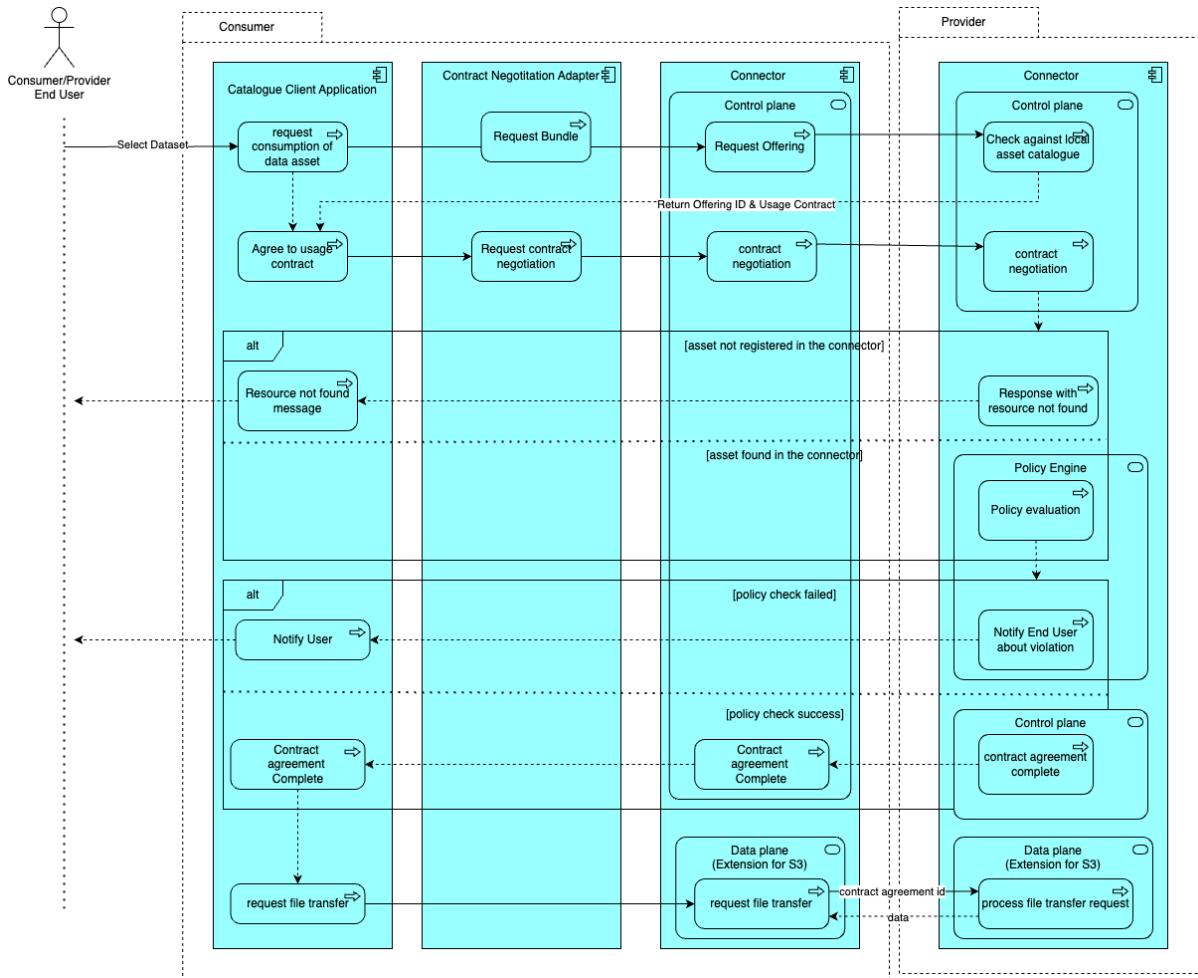
- Data Provider has registered the resource at the Connector;
- Data Provider has created the SD for the resource (meta data description) and uploaded the SD to the data catalogue;
- Consumer has logged in through their agent;
- Consumer found the needed dataset using the searching capabilities on the Data Catalogue;
- If the contract doesn't exist, Consumer and Provider must establish a Contract on the requested resource (BP7 - see relative architecture for further details);
- Consumer has an available and compatible storage.

Assumptions for the MVP:

- The existence of the contract is not checked;

The view shows the dynamic application view of consuming a data resource by directly being given access to the dataset. It outlines the key functional components involved in the process of consuming a data resource.

The consumer initiates a request for the resource previously found in the catalogue, for which a contract has already been established. This request is sent to the provider through the connector. Upon receiving the request, the provider verifies the policies to ensure that the consumer has the necessary permissions to perform the requested action. The policies that are checked are only those that can technically be enforced. For all others, since the dataset is downloaded, the contract enforces the legally binding usage policies. Once the policies are confirmed, the transaction takes place between the two data orchestrator components, which, for the MVP, will be implemented as extensions of the EDC connector. These orchestrator components handle the interface between the connector and the actual source on the provider side and sink on the consumer side of the data.



1. Dataset Selection by the Consumer:

- The process begins in the *Catalogue Client Application* on the consumer side, where the consumer selects a dataset of interest. This action initiates a "*Request Consumption of Data Asset*" message, which is sent to the *Contract Negotiation Adapter*.
- This message signals the consumer's intent to access the resource and moves the negotiation process forward.

2. Creation of Request Bundle:

- The *Contract Negotiation Adapter* takes the consumer's request and compiles a *Request Bundle*.
- This bundle includes information about the selected dataset and any initial parameters needed to facilitate negotiation. It is forwarded to the consumer's *Connector (Control Plane)* for further processing.

3. Requesting an Offering from the Provider:

- The consumer's *Connector (Control Plane)* sends a *Request Offering* message to the provider's *Connector (Control Plane)*.
- This step involves querying the provider's system to locate the requested dataset and determine its availability.

4. Asset Validation by the Provider's Connector:

- The provider's *Connector (Control Plane)* checks the *Asset Catalogue* for the requested dataset:
 - If the dataset is not found**, the provider responds with a *Resource Not Found Message*. This message is propagated back through the consumer's *Connector* and *Contract Negotiation Adapter* to notify the consumer, effectively halting the process.
 - If the dataset is found**, the workflow transitions into the contract negotiation phase.

5. Contract Negotiation Between Connectors:

- Once the dataset is validated, the consumer and provider's *Connectors (Control Planes)* begin negotiating the terms of usage.
- This includes setting access conditions, pricing, compliance requirements, and obligations. The outcome of this step is a draft contract that must undergo further validation.

6. Policy Evaluation on the Provider's Side:

- The draft contract is sent to the provider's *Policy Engine* for evaluation against governance and compliance rules.
 - If the policy check fails**, the *Policy Engine* sends a notification back to the consumer (via the *Connector Control Plane* and *Contract Negotiation Adapter*) explaining the violation. The process halts here unless the consumer modifies the request to comply.
 - If the policy check succeeds**, the *Policy Engine* approves the contract, and the workflow proceeds to finalisation.

7. Notification of Policy Check Results:

- The results of the policy evaluation (either success or failure) are sent back to the consumer's *Connector (Control Plane)*.
- In case of failure**, the *Contract Negotiation Adapter* notifies the consumer, providing details about the violation.

- If the policy check is successful, the contract is finalised and marked as complete.

8. Finalisation of Contract Agreement:

- Once approved, the contract agreement is formalised within the *Control Plane* of both the consumer and provider connectors.
- At this point, both parties have a binding agreement that governs the terms for the upcoming data transfer.

9. Initiation of File Transfer Request:

- With the contract in place, the consumer's *Data Plane Extension for S3* sends a *Request File Transfer* message to the provider's *Data Plane Extension for S3*.
- This request includes the **contract agreement ID** as a reference, ensuring that the data transfer adheres to the agreed terms.

10. Processing the File Transfer:

- The provider's *Data Plane Extension for S3* verifies the file transfer request using the contract ID and cross-checks it against the agreed terms.
- Once validated, the dataset is securely transferred to the consumer, completing the process.

ACV Dynamic - BP 09B - Consumer receives data processing service over a dataset via an Application

The consumer seeks to perform actions such as visualization or processing on a dataset owned by a data provider but does not have direct access to the data itself. Instead, the consumer selects and enters into a contract for an offering from the data provider, which includes the provisioning of an infrastructure resource. An application is then deployed on this infrastructure, enabling the necessary processing of the dataset. Access is provided exclusively through a direct link to the application, ensuring that the consumer cannot directly access the data. As part of the contractual agreement, the consumer is prohibited from attempting to access the data in any way.

Preconditions:

- Data Provider has registered the resource at the Conenctor;
- Data Provider has created the SD for the resource (metadata description) and uploaded the SD to the data catalogue;
- Consumer has logged in through their agent;
- Consumer found the needed resource using the searching capabilities on the Data Catalogue and selected the bundle of dataset, application and infrastructure associated with the dataset.
- If the contract doesn't exist, Consumer and Provider must establish a Contract on the requested resource (BP7) (see relative architecture for further details);

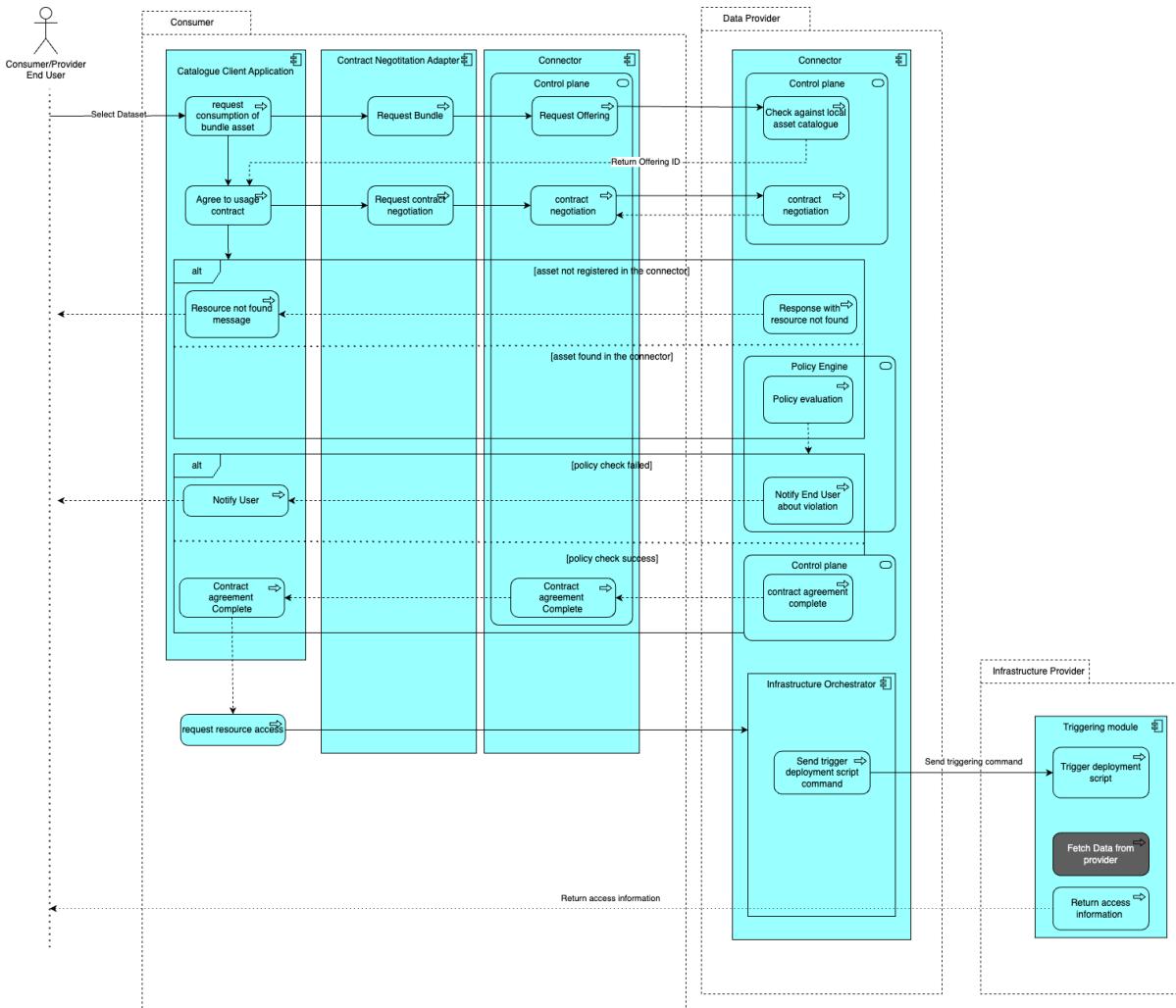
The view shows the dynamic application view of consuming a bundle resource (dataset, application and infrastructure bundled together) by being given access to the provisioned node where the bundle is deployed. It outlines the key functional components involved in the process of consuming the resource.

In this scenario, we make sure that the consumer gains access only to the application, without direct access to the dataset itself. When the consumer selects and contracts a data processing service offering after the contracting is done, the infrastructure resource provisioning and the deployment of the application over the infrastructure instance will take place in the background, and the Consumer will in the end receive the access data and credentials only to the deployed application. The access to the application is not depicted in this scenario.

The components coloured in grey are related to the BP07 (Contract Manager) as well as BP06 (Search) and they are mainly referring to the preconditions.

The diagram represents the action performed to trigger the Bp which consists in using the endpoint, with the needed parameters, contained in the selected description, the user can initiate the contract negotiation process

The diagram also shows the flow how the consumer is requesting a bundled resource via a data provider to the infrastructure provider and receives the respective access. The consumer will request the offering via the Catalogue Client UI, based on a previously identified search result. The Contract Negotiation Adapter is handling the request from the consumer, filtering for the requested asset on the provider's catalogue and return the offering along with the respective usage and access policies. The user is then accepting those and at the same time start the contract negotiation. The contract negotiation adapter is building the request for the connector. After finalising the contract negotiation, the infra structure deployment is triggered. Once this step is completed, the access information is passed to the user,



1. Resource Selection:

- The **Catalogue Client Application** on the Consumer side initiates a request for consuming a bundled resource (dataset, application, and infrastructure) by selecting it from a search result. This request is sent to the **Contract Negotiation Adapter** to begin the process.

2. Request Offering:

- The **Contract Negotiation Adapter** processes the Consumer's request by forwarding it to the **Connector (Control Plane)** on the Data Provider's side.
- The **Connector** checks the requested resource against its **Asset Catalogue**:
 - If the asset is **not found**, a "Resource Not Found" message is sent back to the Consumer.
 - If the asset is **found**, the Connector retrieves the associated offering details, including usage and access policies, and returns them to the Consumer for review.

3. Policy Agreement:

- The Consumer, using the **Catalogue Client Application**, reviews the retrieved offering details.
- Upon agreeing to the usage and access policies, the Consumer initiates a contract negotiation via the **Contract Negotiation Adapter**.

4. Contract Negotiation Request:

- The **Contract Negotiation Adapter** composes a contract negotiation request and sends it to the **Connector (Control Plane)** of the Data Provider.
- The **Policy Engine** evaluates the request based on predefined policy rules:
 - If the **policy check fails**, the Consumer is notified of the violation.
 - If the **policy check succeeds**, the contract is finalised, and a confirmation is sent to the Consumer.

5. Infrastructure Deployment Trigger:

- Once the contract is finalised, the **Connector (Control Plane)** triggers the **Infrastructure Orchestrator** to begin the provisioning process for the infrastructure and application deployment.

6. Triggering Deployment:

- The **Infrastructure Orchestrator** sends a deployment command to the **Triggering Module** of the Infrastructure Provider, detailing the specifications for provisioning the required infrastructure and deploying the application.

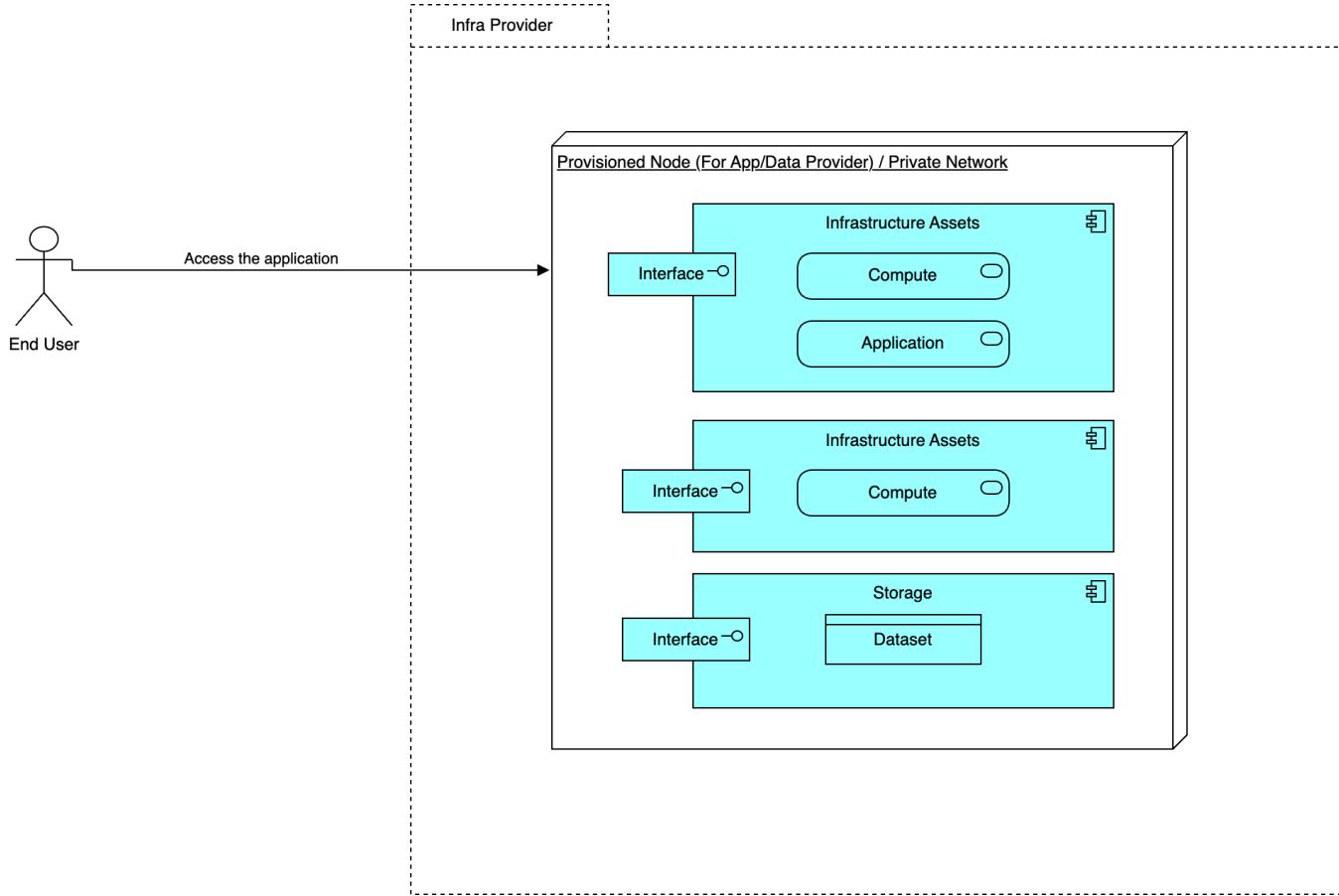
7. Provisioning and Deployment:

- The **Triggering Module** provisions the infrastructure instance and deploys the application onto it as per the deployment command.
- Once the deployment is complete, the **Triggering Module** fetches the access credentials (e.g., application URL, API keys) and sends them back to the **Infrastructure Orchestrator**.

8. Returning Access Information:

- The **Infrastructure Orchestrator** relays the access information to the **Connector (Control Plane)** on the Data Provider's side.
- The **Connector** forwards the access details to the **Contract Negotiation Adapter**, which delivers them to the Consumer, completing the workflow.

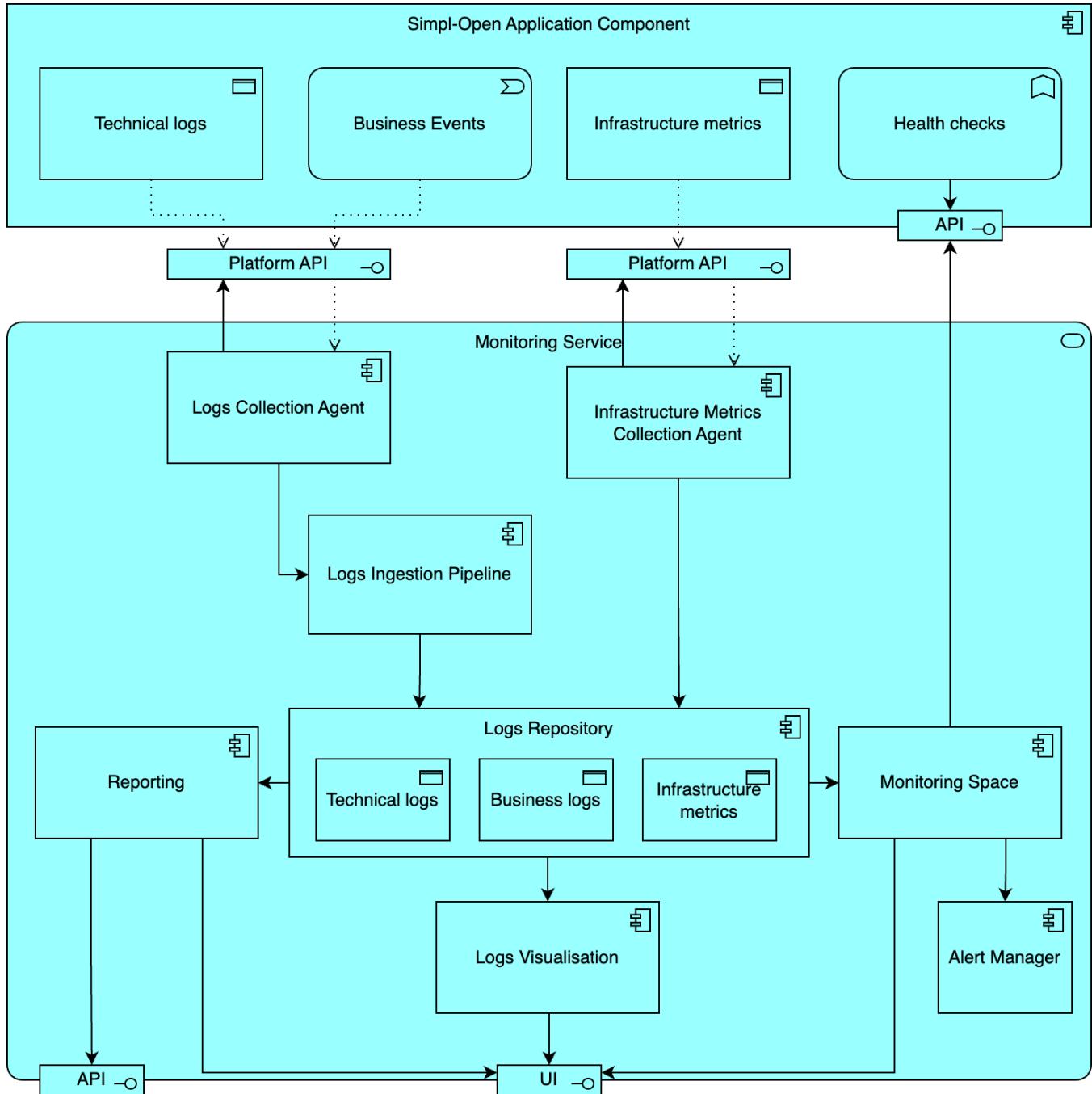
This scenario follows the one above. Once the login credentials are received, the user accesses the dedicated infrastructure through a direct link.



ACV - Domain 3 - Management/Operation of data space

This section describes the architecture for Monitoring and Logging, within a single node (Simpl-Open agent) and does not (yet) consider inter-nodes setup.

The diagram presents a static application view of the domain.



Simpl-Open Application Component

- This component represents an abstraction of any Simpl-Open application component which are being monitored. These components can produce:
 - Technical logs generated by the application and the underlying platform - e.g. access logs, error logs, etc.;
 - Business events generated by the application upon specific triggers in the business workflow - e.g. "Participant successfully onboarded";
 - Infrastructure metrics - e.g. CPU utilisation, RAM utilisation, etc.;
 - Health checks which are APIs implemented by the application components of the Simpl-Open agent to report on the status of the service - e.g. HTTP 200 "OK".

Platform API

- The platform API is an API provided by the platform on which Simpl-Open application components are deployed, which allows to collect enriched logs and metrics.

Monitoring Service

- The **monitoring service** is modeled as a service which is offered to all the Simpl-Open Application Components. It is implemented through the set of components described below.

Log Collection Agent

- The **log collection agent** collects Technical and business logs from each **Simpli-Open application component** and forwards them to the **log ingestion pipeline**.

Log Ingestion Pipeline

- The **log ingestion pipeline** receives the logs from the **log collection agent** and standardises their format before storing them in the **logs repository**.

Infrastructure Metrics Collection Agent

- The **infrastructure metrics collection agent** collects infrastructure metrics from each **Simpli-Open application component** and stores them directly in the **logs repository**.

Logs Repository

- The **logs repository** serves as a central hub to store all types of logs and metrics. It then feeds the **monitoring space**, the **logs visualisation** component and the **reporting** component.

Monitoring Space

- The **monitoring space** displays dashboards built on top of the different logs but can also query directly health endpoints to display their status.

Logs Visualisation

- The **logs visualisation** component allows to run queries on the logs and visualise them in a user interface. Logs visualisation and monitoring space share a common UI with distinct tab for each functionality.

Reporting

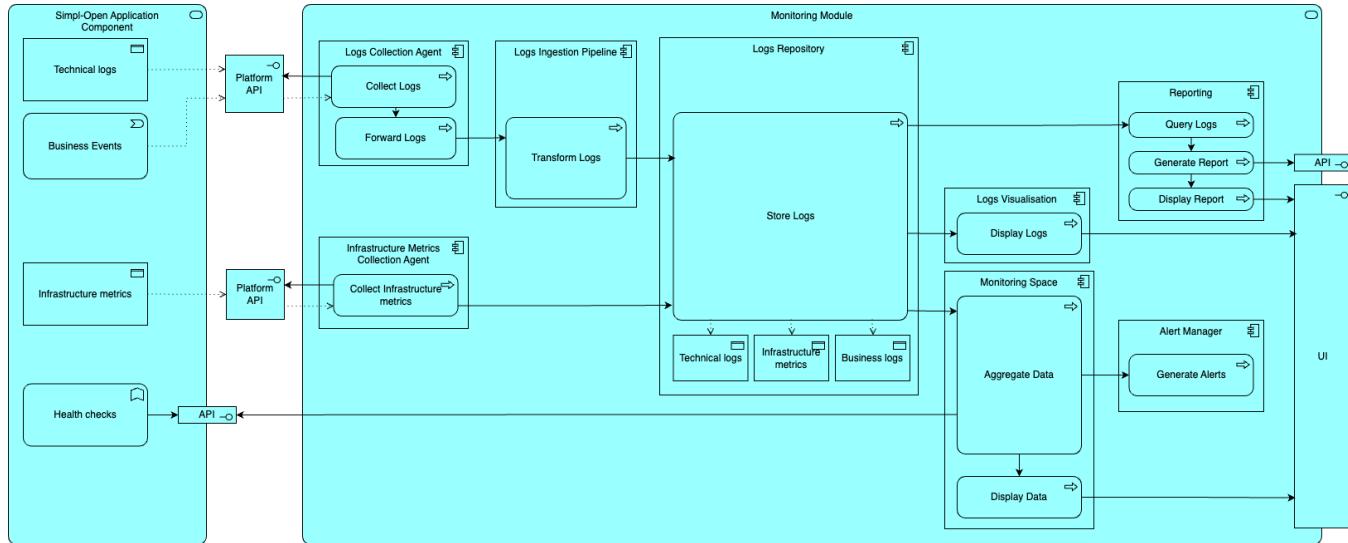
- The **reporting** component includes both a user interface from which exports can also be performed, and an API to query logs for other purposes such as monitoring federation or billing.

Alert Manager

- The **monitoring space** is connected with an **alert manager** to trigger alerts based on predefined thresholds.

ACV Dynamic - WF 12B - Local Node Logging and Monitoring

Below diagram describes how the components presented above interact with each other to rend the functionalities.



The Simpli-Open Application Component generates various types of data, including technical logs, business events, infrastructure metrics, and health check outputs. These data streams are exposed via APIs for collection.

- Log and Event Generation:** The Simpli-Open Application Component produces technical logs, business events, infrastructure metrics, and health check data, exposing these via APIs.
- Logs Collection:** The Logs Collection Agent retrieves technical logs and business events, forwarding them to the Logs Ingestion Pipeline for processing.
- Metrics Collection:** The Infrastructure Metrics Collection Agent gathers infrastructure metrics and directly forwards them to the Logs Repository.
- Log Transformation:** The Logs Ingestion Pipeline processes and transforms the raw logs into a standardised format before storing them in the Logs Repository.

5. **Centralised Storage:** The Logs Repository stores technical logs, infrastructure metrics, and business events in dedicated sections, ensuring they are accessible for subsequent steps.
6. **Log Visualisation:** The Logs Visualisation component retrieves and displays logs for analysis, allowing users to review technical, infrastructure, and business-related events.
7. **Data Aggregation:** The Monitoring Space aggregates logs and metrics, enabling real-time analysis of system health and performance.
8. **Alert Generation:** The Alert Manager processes aggregated data, generating alerts for any anomalies or threshold breaches, and notifying relevant stakeholders.
9. **Report Generation:** The Reporting Module queries logs and metrics from the Logs Repository to create detailed reports.
10. **Report Presentation:** These reports are displayed through a user-friendly interface, providing actionable insights for decision-making.

Interfaces

APIs

Below table presents the APIs of all the components depicted on the application deployment views. These APIs can be correlated to the Application Components Views static diagrams (per domain) through the numbering appearing on both the diagrams and the first column of this table.

Simpl-Open uses 2 types of APIs:

1. Synchronous JSON/HTTP APIs;
2. Asynchronous JSON/Kafka APIs.

Each API is described in a functional way and linked to the technical contract definition (e.g. OpenAPI definition for sync APIs) which is stored in GitLab.

For custom-built REST APIs, the following guidelines from the Belgian Interoperability Framework will be used (after the MVP) as reference: <https://www.belgif.be/specification/rest/api-guide/>

Specifically, the following guidelines will be applied in priority:

- The contract-first principle SHOULD be followed when developing an API. In this approach, the specifications of the REST API are created first and not generated from the code. More details [here](#).
- Specifications of the API SHOULD be provided using [OpenAPI 2.0](#) (aka Swagger) or [OpenAPI 3.0](#). OpenAPI uses the [OpenAPI Schema Object](#) to describe the JSON representation of resources, which is a variant of [JSON Schema](#), with some significant incompatibilities. More details [here](#).
- Path segments and query parameters within an API SHOULD use lowerCamelCase notation to enhance readability and to separate compound names. As lowerCamelCase is used for JSON property names as well, the casing is consistent throughout the API. Trailing slashes MUST NOT be used. More details [here](#).
- The URI SHOULD NOT contain a file extension. More details [here](#).
- A plural noun SHOULD be used for collection names, for example 'employers' or 'people'. More details [here](#).
- The HTTP methods SHOULD be used that are appropriate for the type of action performed on the resource. More details [here](#).
- The HTTP response status code SHOULD be returned that best describes the outcome of the treatment of the request. More details [here](#).
- Following guidelines SHOULD be respected when determining a name for a JSON property:
 - use *lowerCamelCase* notation
 - also for abbreviations: all letters after the first should be lowercase
 - use American English
 - do not use underscores (_), hyphens (-) or dots (.) in a property name, nor use a digit as first letter
 - don't use overly generic terms like `info(rmation)` and `data` as property name or as part of it
 - the name should refer to the business meaning of its value in relation to the object in which it is specified, rather than how it is defined
 - omit parts of the name that are already clear from the context

Properties used from other standards, like OpenID Connect and OAuth2, are allowed to deviate from this rule. More details [here](#).

- Add example response values to the OpenAPI specification under the `examples` property. More details [here](#).
- Kebab-Case with uppercase SHOULD be used for HTTP header names. More details [here](#).

#	Component	Sync APIs		Async APIs	
		Name	Technical definition	Name	Technical definition
1	SD Tooling	<ul style="list-style-type: none"> • retrieve all available shapes/schemas • Register Asset to the Connector • Validate Self-Descriptions • Retrieve available identity attributes • Create Access Policies • Create Usage Policies • Publish Self Descriptions 	https://creation-wizard-api.dev.simpl-europe.eu/swagger-ui/index.html#/	c: schema-changed	
5	Signer Service	<ul style="list-style-type: none"> • <code>/v1/sign</code>: sign self-description. 	https://gitlab.eclipse.org/eclipse/xfsc/tsa/signer/-/blob/ocm-wstack/gen/http/openapi3.json	/	/
6	Catalogue Client Application	<ul style="list-style-type: none"> • <code>quickSearch</code>: quick search; • <code>advanceSearch</code>: advanced search. 	https://xsfc-advsearch-be.dev.simpl-europe.eu/swagger-ui/index.html	/	/

7	Schema Management	<ul style="list-style-type: none"> changeSchemaConfiguration: update the yaml configuration; uploadSchemaConfiguration: add a new yaml configuration; revokeSchema: delete the configuration and remove the schema from the catalogue; listAvailableSchema: list all the schemas currently loaded in the catalogue; transformSchema: transform the yaml configuration to a semantic schema, publish to the catalogue, inform provider; syncSchema: return all the events in topic schema-changed since offset. 	https://code.europa.eu/simpl/simpl-open/development/gaia-x-edc/poc-gaia-edc-/blob/feature/1234/Swagger.json	p: schema-changed		
8	Vocabulary Management	<ul style="list-style-type: none"> listVocabularies; removeVocabulary; UploadVocabulary. 	https://code.europa.eu/simpl/simpl-open/development/gaia-x-edc/poc-gaia-edc-/blob/feature/1234/Swagger.json	/	/	
9	Catalogue	<ul style="list-style-type: none"> a) get a full list of all vocabularies and schemas stored in the catalogue; b) add a vocabulary or schema to the catalogue; c) get a specific schema/vocabulary; d) delete a schema/vocabulary; e) revoke a self-description; f) get the complete self-description; g) publish a self-description to the catalogue; h) sends a cypher query to the Neo4J database for search (used for quick and advanced search); i) validate a self description on Semantics, VPSignature and/or VCSignature depending on the options selected in the request. 	https://code.europa.eu/simpl/simpl-open/development/gaia-x-edc/poc-gaia-edc-/blob/feature/1234/Swagger.json	/	/	
12	Query Mapper A adapter	<ul style="list-style-type: none"> quickSearch AdvancedSearch 	https://adapter.dev.simpl-europe.eu/swagger-ui/index.html			
13	Tier 1 Authentication Provider	Keycloak <ul style="list-style-type: none"> OIDC: set of APIs for managing authentication; GET Realms: API for retrieving all the Realms configured; GET Realm by Realm Name: API for retrieving a Realm using its name; GET Users: API for retrieving the user list of a Realm; GET Roles: API for retrieving the role list of a Realm; GET Roles by User ID: API for retrieving the roles assigned to a user. 	https://www.keycloak.org/docs-api/latest/rest-api/index.html			
14	Tier 2 Authentication Provider	Keypair Controller <ul style="list-style-type: none"> GET /keypair - Get installed keypair POST /keypair - Import KeyPair POST /keypair/generate - Generate KeyPair CRS Controller <ul style="list-style-type: none"> POST /csr/generate - Generate CSR 				
15	User & Roles	User Roles <ul style="list-style-type: none"> role-controller: a set of APIs for managing roles and the assignment between role and identity attributes; <ul style="list-style-type: none"> POST /role/{id}/identity-attributes - assigns the identity attributes to a role; POST /role/{id}/duplicate-identity-attribute - copies the identity attributes assignment from a source role to another target role; POST /role/assigned-identity-attributes - retrieves the identity attributes for a set of roles; DELETE /role - delete the assignment of identity attributes from a role. credential-controller: a set of APIs for getting the information about the credential installed within the agent; <ul style="list-style-type: none"> GET /credential - gets the information about the credential installed; POST /credential - installs the credential; DELETE /credential - deletes the credential installed; GET /credential/public-key - gets the public key information about the credential installed; GET /credential/my-id - gets the id information about the credential installed. user-controller: a set of APIs for managing users and related information associated with the user; <ul style="list-style-type: none"> GET /user - gets the Tier 1 users; GET /user/{uuid}/roles - gets the Tier 1 roles associated with the user. session-controller: a set of APIs for managing information about the session of a participant, including the retrieval of identity attributes from the ephemeral proof; <ul style="list-style-type: none"> GET /session/{participantId} - retrieves the session information of a participant. identity-attribute-controller: a set of APIs for managing information about the identity attributes stored locally in the agent; <ul style="list-style-type: none"> GET /identity-attribute/search - searches the identity attributes stored locally in the agent. agent-controller: a set of APIs for managing agent communication and providing request and synchronization functionalities for identity attributes from the ephemeral proof to the local copy. 	https://code.europa.eu/simpl/simpl-open/development/iaa/users-roles-/tree/main/openapi?ref_type=heads			
16	Security Attributes Provider	Security Attributes Provider <ul style="list-style-type: none"> identity-attribute-controller: a set of APIs for managing identity attributes and the assignment/unassignment functionalities to the participant; <ul style="list-style-type: none"> GET /identity-attribute/{id} - retrieves the identity attribute; PUT /identity-attribute/{id} - updates the identity attribute; POST /identity-attribute - creates the identity attribute; GET /identity-attribute/search - searches the identity attributes; DELETE /identity-attribute/{id} - deletes the identity attribute; PUT /identity-attribute/assignable/{value} - changes the assignable value to make an attribute assignable or not assignable; PUT /identity-attribute/assign-participant/{userId} - assigns the identity attributes to a participant; DELETE /identity-attribute/unassign-participant/{userId} - deletes the assignment (unassign) the identity attributes to a participant; PUT /identity-attribute/add-participant-type/{participantType} - assigns the identity attributes to a participant type. mtls-controller 	https://code.europa.eu/simpl/simpl-open/development/iaa/security-attributes-provider-/tree/main/openapi?ref_type=heads			

		<ul style="list-style-type: none"> ◦ GET /mtls/identity-attribute - retrieves the identity attributes assigned to a participant's ID; ◦ GET /mtls/identity-attribute/{certificateId} - retrieves the identity attributes assigned to a credential Id. 		
17	Identity Provider	<p>Identity Provider</p> <ul style="list-style-type: none"> • participant-controller: a set of APIs for managing participants including the onboarding functionalities <ul style="list-style-type: none"> ◦ PUT /participant/{userId} - changes the detail of a participant; ◦ GET /participant - retrieves the participant's information; ◦ POST /participant - creates an applicant participant; ◦ POST /participant/initialize - at the approval of the onboarding request, creates the participant and its credentials; ◦ POST /participant/attachment - uploads the documents needed during the onboarding procedure; ◦ GET /participant/{participantId}/credential-validity - retrieves the expiration date of the credential of a participant; ◦ GET /participant/search - searches for participants. • mtls-ephemeral-proof-controller <ul style="list-style-type: none"> ◦ POST /mtls/token - generates the ephemeral proof of a participant. • mtls-controller <ul style="list-style-type: none"> ◦ GET /mtls/echo - technical API for testing mTLS communication. • certificate-controller <ul style="list-style-type: none"> ◦ GET /certificate/{certificateId} - retrieves the credential of a participant; ◦ DELETE /certificate/{certificateId} - deletes (revokes) a credential of a participant. 	https://code.europa.eu/simpl/simpl-open/development/ia/identity-provider/-/tree/main/openapi?ref_type=heads	
		<p>EJBCA REST Interface API</p> <ul style="list-style-type: none"> ◦ GET /ejbca/publicweb/status/ocsp - checks the status and availability of the OCSP (Online Certificate Status Protocol) service provided by EJBCA; ◦ GET /ejbca/publicweb/webdist/certdist - distributes certificate-related files, such as Certificate Authority (CA) certificates and Certificate Revocation Lists (CRLs). 	https://docs.keyfactor.com/ejbca/latest/ejbca-rest-interface	
18	Onboarding	<p>Onboarding Template APIs</p> <ul style="list-style-type: none"> • GET /onboarding-template/{participantType}: Get Onboarding Template by Participant Type • PUT /onboarding-template/{participantType}: Update Onboarding Template • DELETE /onboarding-template/{participantType}: Delete Onboarding Template • PATCH /onboarding-template/{participantType}: Update Onboarding Template • GET /onboarding-template: Get Onboarding Templates <p>Onboarding Status APIs</p> <ul style="list-style-type: none"> • GET /onboarding-status/{value}: Get Onboarding Status by Value • PUT /onboarding-status/{value}: Update Onboarding Status • DELETE /onboarding-status/{value}: Delete Onboarding Status • GET /onboarding-status/is-final: Check if Onboarding Status is Final • GET /onboarding-status/initial: Get Initial Onboarding Status <p>MIME Type APIs</p> <ul style="list-style-type: none"> • GET /mime-type/{id}: Get MIME type by ID • PUT /mime-type/{id}: Update MIME type by ID • DELETE /mime-type/{id}: Delete MIME type by ID • GET /mime-type: Get all MIME types • POST /mime-type: Create a new MIME type <p>Onboarding Request APIs</p> <ul style="list-style-type: none"> • GET /onboarding-request: Search Onboarding Requests • POST /onboarding-request: Create Onboarding Request • POST /onboarding-request/{id}/document: Add Document to Onboarding Request • PATCH /onboarding-request/{id}/document: Set Document for Onboarding Request • PATCH /onboarding-request/{id}/status: Set Onboarding Request Status • PATCH /onboarding-request/{id}/expiration-timeframe: Set Expiration Timeframe for Onboarding Request • PATCH /onboarding-request/{id}/comment: Add Comment to Onboarding Request • GET /onboarding-request/{onboardingRequestId}: Get Onboarding Request • GET /onboarding-request/{onboardingRequestId}/document/{documentId}: Get Document from Onboarding Request • DELETE /onboarding-request/{onboardingRequestId}/document/{documentId}: Delete Document from Onboarding Request <p>Participant Type APIs</p> <ul style="list-style-type: none"> • GET /participant-type: Get all participant types <p>Credential Request APIs</p> <ul style="list-style-type: none"> • POST /credential-request: Create a new credential request 	https://code.europa.eu/simpl/simpl-open/development/ia/onboarding/-/tree/main/openapi?ref_type=heads	
20	Connector	<p>Management API</p> <ul style="list-style-type: none"> • Configuring and managing Assets (Any kind of resources); • Assign Policies; • Assign Contract Templates. 	https://app.swaggerhub.com/apis/eclipse-edc-bot/management-api/0.7.0	
21	Connector	<p>Control Plane</p> <ul style="list-style-type: none"> • Start Contract negotiation. 	https://app.swaggerhub.com/apis/eclipse-edc-bot/control-api/0.7.0	
22	Connector	<p>Data Plane</p> <ul style="list-style-type: none"> • Data Transfer, CRUD. 	https://app.swaggerhub.com/apis/eclipse-edc-bot/public-api/0.7.0	

23	Triggering Module	<p>Deployment Script ID management API is providing the following functionalities:</p> <ul style="list-style-type: none"> Adding the Deployment Script (also from the UI, which relies on this API); Modifying the Deployment Script (also from the UI, which relies on this API); Removing the Deployment Script (also from the UI, which relies on this API). The removal process will remove the deployment script only from the repository, and not from the database, to keep a trace for future potential business needs. If the business decides to also remove it from the data base, the technological implementation is effortless; Triggering the deployment Script (at the time of contracting an offer on the catalogue, if the offer has a DeploymentScriptID, the API will be called and that script will be executed). 	https://code.europa.eu/simpl/simpl-open/development/infrastructure/infrastructure-be-/blob/develop/openapi/script-management-openapi.yaml		
24	Contract Manager Orchestrator	<ul style="list-style-type: none"> Issue Verifiable credential <ul style="list-style-type: none"> POST /contract/v1/credentials/agreements/{contractAgreementId}/definitions/{contractDefinitionId} Confirm signing of the Contract Agreement <ul style="list-style-type: none"> POST /contract/v1/agreements/{contractAgreementId}/definitions/{contractDefinitionId}/signatures/confirmation Get Contract Agreement <ul style="list-style-type: none"> GET /contract/v1/agreements/{contractAgreementId} Get Contract Agreement File <ul style="list-style-type: none"> GET /contract/v1/agreements/file/{contractAgreementId} 	https://code.europa.eu/simpl/simpl-open/development/contract-billing/contract/-/tree/develop/http	<ul style="list-style-type: none"> Issue Verifiable credential Response <ul style="list-style-type: none"> Contract Agreement Response Event 	src/main/java/eu/europa/ec/simpl/contracts/kafka/events-main-Simpl/Simpl-Open/Development/Contract-Billing/contract-GitLab
25	Contract Manager Backend	/	/	<ul style="list-style-type: none"> Issue Verifiable credential Request <ul style="list-style-type: none"> Contract Agreement Request Event Confirm signing of the Contract Agreement <ul style="list-style-type: none"> Status Update Request Event 	src/main/java/eu/europa/ec/simpl/contracts/kafka/events-main-Simpl/Simpl-Open/Development/Contract-Billing/contract-GitLab
27	VC Issuer	<ul style="list-style-type: none"> Issue Verifiable Credential. 	<div style="border: 1px solid #ccc; padding: 5px; display: inline-block;">  Currently under investigation </div>	/	

28	Contract Consumption Adapter	APIs for starting contract negotiation & requesting a resource transfer <ul style="list-style-type: none"> • Request an offering from a provider • Request contract negotiation for a specific asset • Receive the Status of contract negotiation 	https://contract-consumption-be.dev.simpl-europe.eu/swagger-ui/index.html		
----	------------------------------	---	---	--	--

completed, planned

User Interfaces

 This section is a placeholder for content that will be added by 31 déc. 2024.

Traceability from the functional architecture

The following table presents a mapping between the components from the functional architecture and the ones from the application architecture.

Functional Component	Application Component
Onboarding	Onboarding
IAA	Authorisation
	Tier 1 Authentication Provider
	Tier 2 Authentication Provider
	Identity Provider
	Security Attributes Provider
	User & Roles
	Credential Database/Vault
Vocabulary Management	Vocabulary Management
Schema Management	Schema Management
	Schema Registry
Service Offering Editor	SD Tooling
	Signer Service
	Wallet
	Policy Template Datastore
Federated Catalogue	Catalogue
Search	Catalogue Client Application
Data Space Connector	Connector
Contract Management	Contract Manager Orchestrator
	Contract Manager Backend
	Contract Template Datastore
Data Transfer	Connector
Infrastructure Management	Triggering Module
	Infrastructure Provisioner
Observability	Monitoring Module

Simpl-Open Data Architecture

Simpl-Open Data Architecture presents data entities and/or collections and how they are structured within the system.

Given that Simpl-Open combines existing/reusable open-source components and custom-built components, the following approach is followed:

1. For open-source components, the dedicated sub-section provides a link to the available data model documentation of the component.
2. For custom components, the dedicated sub-section describes the data model per component (as per the microservices approach) through following layers:

Layer	Description
Conceptual	The conceptual data model (CDM) operates at a high level, providing an overarching perspective on the application's data needs. It defines a broad and simplified view of the data to create a shared understanding of the application by capturing the essential concepts. These essential concepts are captured in an Entity Relationship Diagram (ERD) and the accompanying entity definitions.
Logical	The logical data model (LDM) contains representations that fully defines relationships in data, adding the details and structure of essential entities. The LDM remains data platform agnostic because it focuses on business needs, flexibility, and portability. The LDM includes the specific attributes of each entity, the relationships between entities, and the cardinality of those relationships.
Physical	The physical data model (PDM) is a data model that represents relational data objects. It describes the technology-specific and database-specific implementation of the data model and is the last step in transforming from the logical data model to a working database. The physical data model includes all the needed physical details to build the database.

Open-Source Components Data Model

OSS	Data Model
XFSC Signer	The self-description is wrapped into a verifiable credentials and the proof section of the VC contain the signature. The data model is defined here: https://www.w3.org/TR/vc-data-model/#proofs-signatures
XFSC catalogue	XFSC catalogue stores the data in three different ways: <ul style="list-style-type: none"> • File storage for the JSON-LD serialisation. Definition of the file format can be found https://json-ld.org/. The data model itself depend on the schema definition used (defined in Additional Technical Specifications about Capabilities section). • Graph-DB (Neo4J) used as index to allow semantic queries. The data model of the database also depends on the schema. • Metadata stored in a relational database (PostgreSQL). Data Model is described in https://gaia-x.gitlab.io/data-infrastructure-federation-services/cat/architecture-document/architecture/catalogue-architecture.html#_metadata_store
HashiCorp Vault	Secrets data is stored in secrets engine https://developer.hashicorp.com/vault/docs/secrets . The data model depends on the model used currently a Key-Value (KV) Store Data Model is used.
Keycloak	Keycloak use a "code first" approach to data modelling. There are no data model diagrams available in their documentation but the data model is described in their code repository: https://github.com/keycloak/keycloak/tree/main/model
EJBCA	EJBCA data model diagram is located https://doc.primekey.com/ejbca/ejbca-introduction/ejbca-architecture/internal-architecture
Crossplane	Resource Definition: https://docs.crossplane.io/latest/concepts/composite-resource-definitions/
Kubernetes	Kubernetes objects: https://kubernetes.io/docs/concepts/overview/working-with-objects/
Ansible	Ansible Data Manipulation: https://docs.ansible.com/ansible/latest/playbook_guide/complex_data_manipulation.html
ArgoCD	RBAC Model https://argo-cd.readthedocs.io/en/stable/operator-manual/rbac/#rbac-model-structure

Custom Components Data Model



This section is a placeholder for content that will be added by 31 déc. 2024.

Conceptual Data Model

Logical Data Model

Physical Data Model

Simpl-Open Technology Architecture

Simpl-Open Technology Architecture develops the target technology architecture that enables the application architecture to be delivered through technology components and technology services. Each application component is mapped to a technology implementing the capabilities.

It identifies technology components through the following views:

View	Description
Technology Components Static View	Provide, per business domain, an enriched view of the Application Components Static View by adding technology components that support the implementation of the application components.
Technology Components Dynamic View	Provide a dynamic view (sequence diagrams) per business process (or sub-process) on how technology components are used to satisfy different workflows.
Technology Deployment View	Provides an aggregated view of how the different technology components (cross BPs and domains) are deployed for all Simpl-Open agent types (Governance Authority, Data Provider, Infrastructure Provider, Application Provider, Consumer).

Next to these architecture views, the following are provided:

- A table of OSS Technology - with reasons for selecting them and links to existing documentation such as data models and installation guides;
- Detailed technical specification that are particularly relevant for contributing to Simpl-Open and/or implementing it in a data space.

Technology Components Views

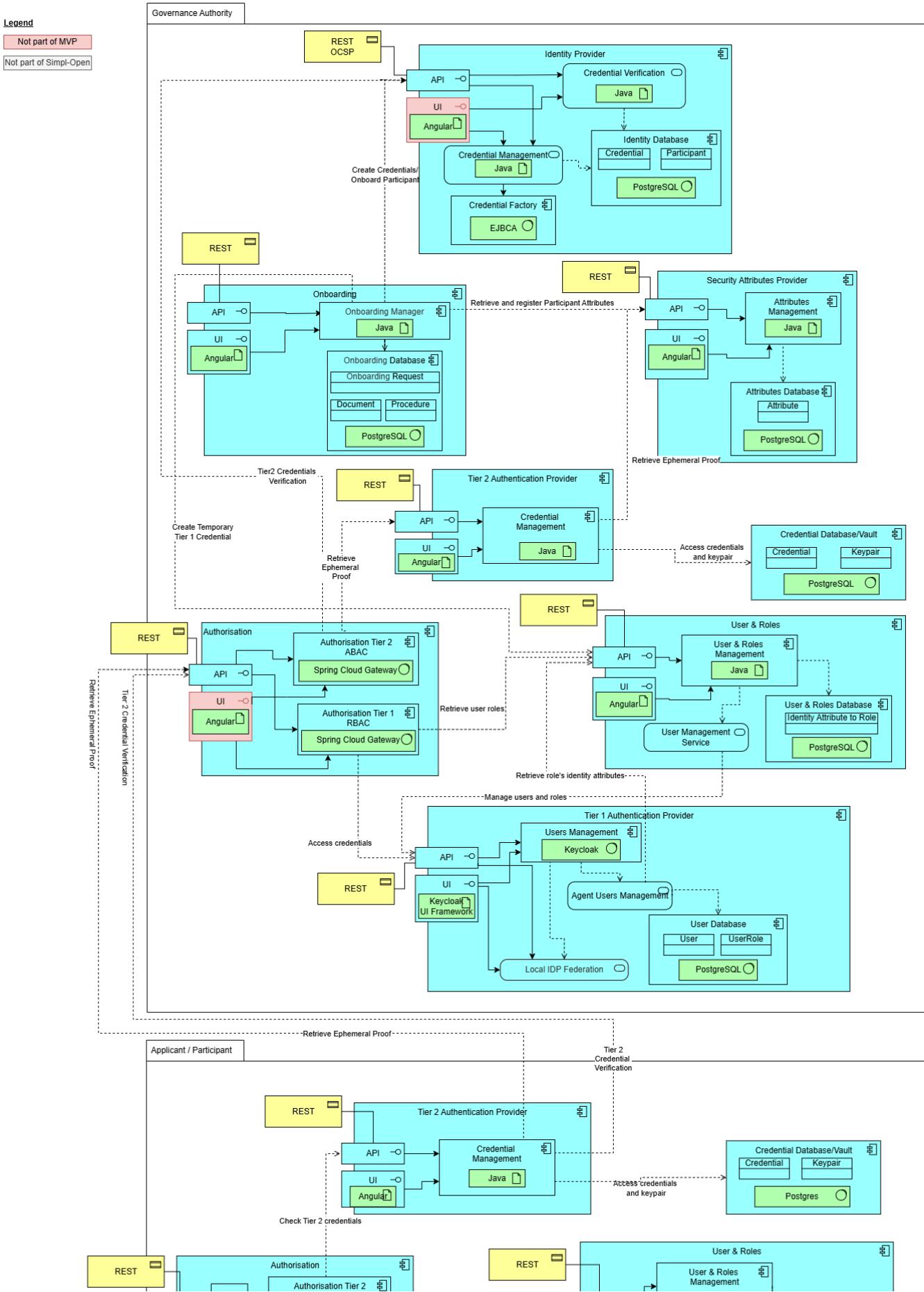
Technology components views are presented per functional domain in following sub-sections.

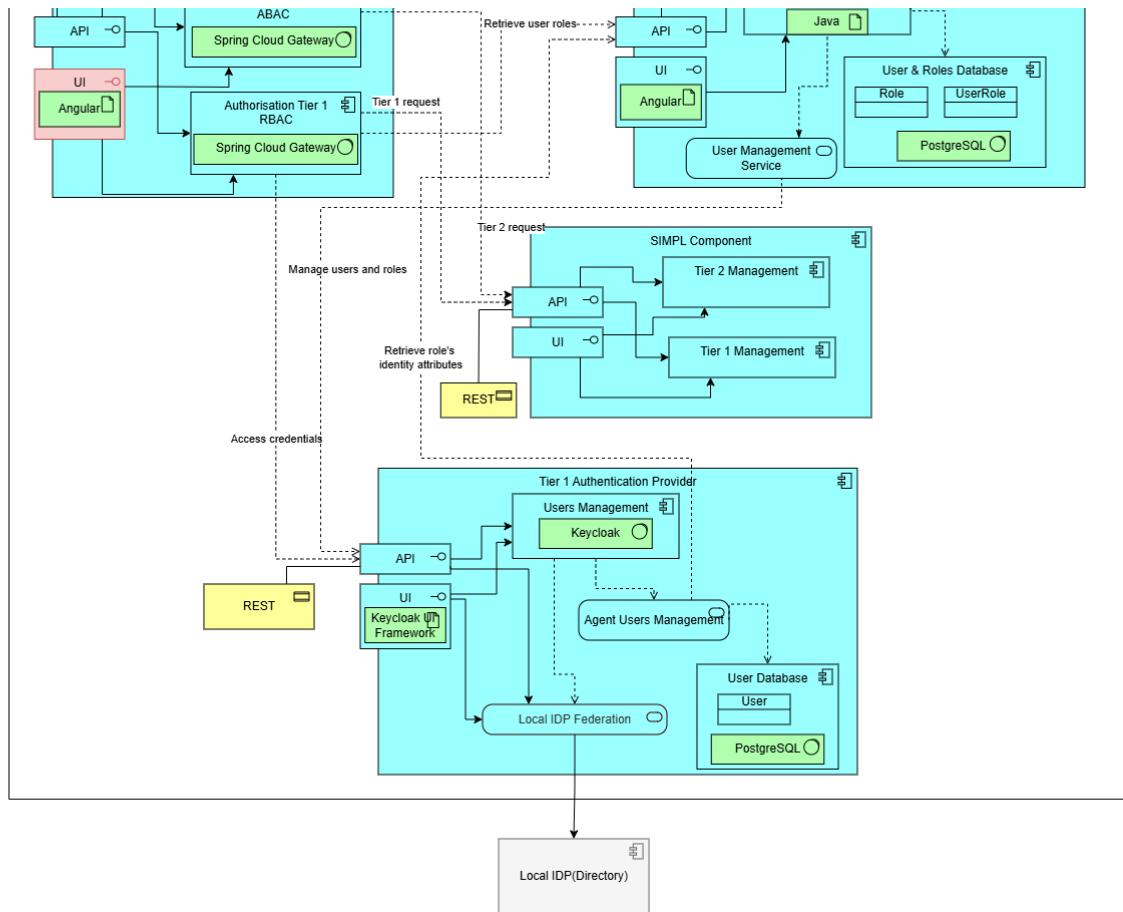
For each functional domain, are presented:

1. a static view of the entire domain which enriches the application components view with the technologies that are implementing the components;
2. a set of dynamic views (sequence diagrams) that present how a subset of the technology components are used to satisfy different (parts of) business processes.

TCV - Domain 1 - Onboarding & IAA

This perspective illustrates the correlation between the architectural elements and the technologies, components, and interfaces intended for use in implementing the application components.





Onboarding

- The **Onboarding Manager** component is implemented as a Java backend application.
- The **Onboarding UI** component is implemented as an Angular frontend application.
- The **Onboarding Database** component is implemented in PostgreSQL.

Identity Provider

- The **Credential Management** service is implemented as a Java backend application.
- The **Credential Verification** service is implemented as a Java backend application.
- The **Identity Provider UI** component is implemented as an Angular frontend application.
- The **Credential Factory** component is implemented with Enterprise JavaBeans Certificate Authority (EJBCA).
- The **Identity Database** is implemented in PostgreSQL.

Security Attributes Provider

- The **Attributes Management** component is implemented as a Java backend application.
- The **Security Attributes Provider UI** component is implemented as an Angular frontend application.
- The **Attributes Database** is implemented in PostgreSQL.

User & Roles

- The **User & Roles Management** component is implemented as a Java backend application.
- The **User & Roles UI** component is implemented as an Angular frontend application.
- The **User & Roles Database** is implemented in PostgreSQL.

Tier 1 Authentication Provider

- The **Users Management** component, providing the Agent Users Management and Local IDP Federation services, is implemented with Keycloak.
- The **Tier 1 Authentication Provider UI** component is implemented as an Angular frontend application.
- The **User Database** is implemented in PostgreSQL.

Tier 2 Authentication Provider

- The **Credential Management** component is implemented as a Java backend application.
- The **Tier 2 Authentication Provider UI** component is implemented as an Angular frontend application.

Credentials Database/Vault

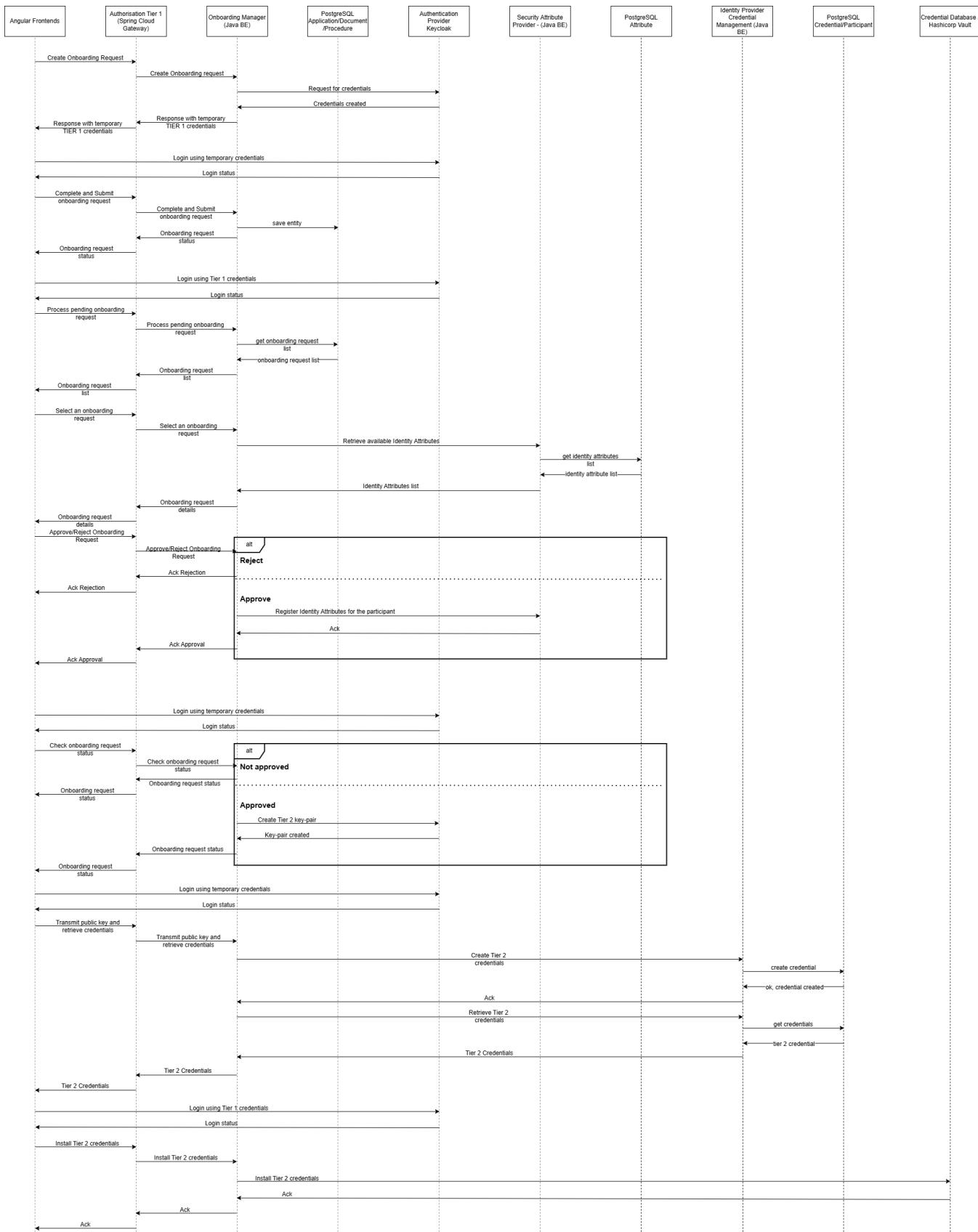
- The **Credentials Database/Vault** is implemented with Postgres. Will be replaced Hashicorp Vault in the future.

Authorisation

- The **Authorisation Tier 1 RBAC** component is implemented with Spring Cloud Gateway.
- The **Authorisation Tier 2 ABAC** component is implemented with Spring Cloud Gateway.
- The **Authorisation UI** component is implemented as an Angular frontend application.

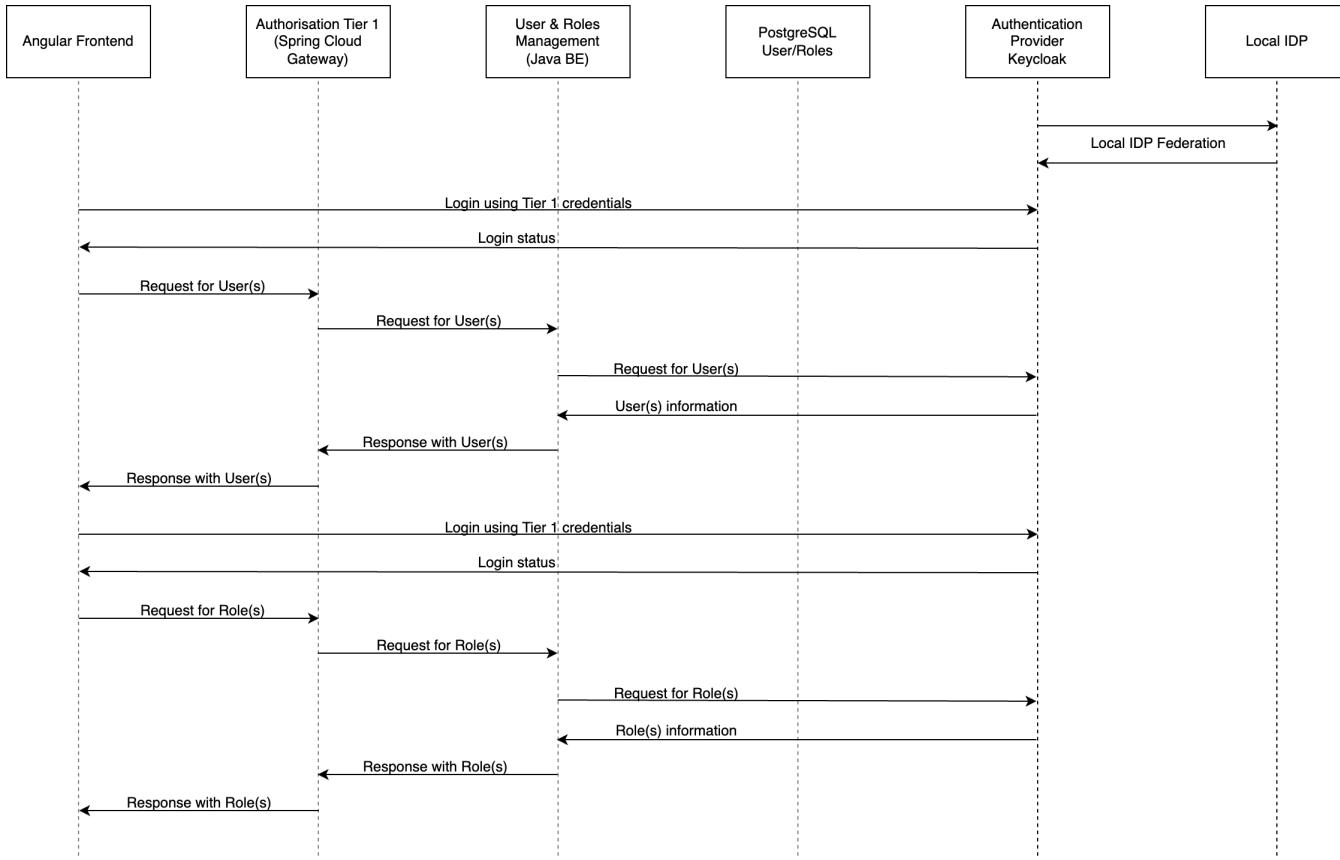
TCV Dynamic - BP 03A - Onboarding of a participant - Tier II

This perspective illustrates the interactions and the flows between all the technological components.



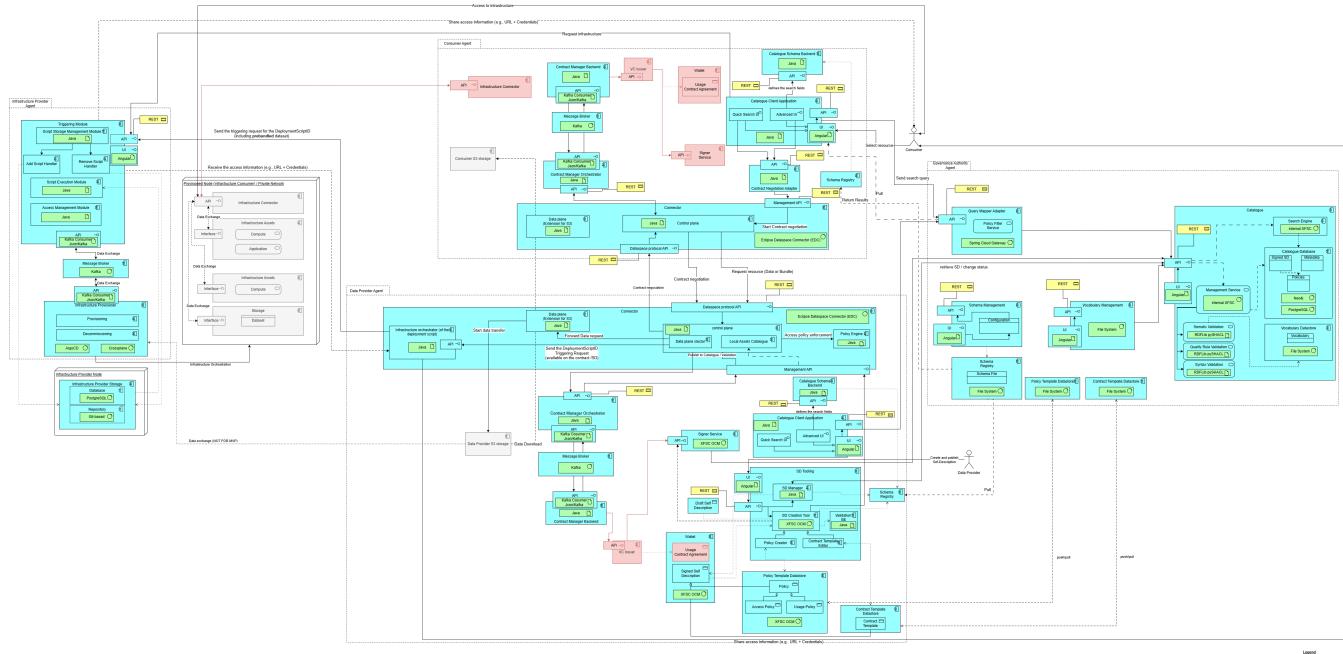
TCV Dynamic - BP 03B - Onboarding Tier 1 - Organisation Local IDP(Directory) Connection/Mapping

This perspective illustrates the interactions and the flows between all the technological components.



TCV - Domain 2 - Publish and consume resources

This perspective illustrates the correlation between the architectural elements and the technologies, components, and interfaces intended for use in implementing the application components.



Triggering Module

- The **Script Storage Management Module** component is implemented as a Java backend application.
 - The **Script Execution Module** component is implemented as a Java backend application.

- The **Access Management Module** component is implemented as a Java backend application.
- The **Triggering Module UI** component is implemented as an Angular frontend application.
- The **API** interface is implemented as a Kafka consumer Json/Kafka.

Message Broker

- The **Message Broker** component is implemented as a Kafka.

Infrastructure Provisioner

- The **Infrastructure Provisioner** component is implemented in ArgoCD.
- The **Infrastructure Provisioner** component is implemented in Crossplane.
- The **API** interface is implemented as a Kafka consumer Json/Kafka.

Infrastructure Provider Storage

- The **Database** is implemented in PostgreSQL.
- The **Repository** is implemented in Git-based.

Contract Manager Backend

- The **Contract Manager Backend** component is implemented as a Java backend application.
- The **API** interface is implemented as a Kafka consumer Json/Kafka.

Contract Manager Orchestrator

- The **Contract Manager Orchestrator** component is implemented as a Java backend application.
- The **API** interface is implemented as a Kafka consumer Json/Kafka.

Connector

- The **Connector** component is implemented as an Eclipse Dataspace Connector.
- The **Control plane** component is implemented as a Java backend application.
- The **Data plane** component is implemented as a Java backend application.
- The **Infrastructure orchestrator** is implemented as a Java backend application.
- The **Policy engine** component is implemented as a Java backend application.

Catalogue Client Application

- The **Catalogue Client Application** component is implemented as a Java backend application.
- The **Catalogue Client Application UI** component is implemented as an Angular frontend application.

Catalogue Schema Backend

- The **Catalogue Schema Backend** component is implemented as a Java backend application.

Contract Negotiation Adapter

- The **Contract Negotiation Adapter** component is implemented as a Java backend application.

SD Tooling

- The **SD Manager** component is implemented as a Java backend application.
- The **Validation BE** component is implemented as a Java backend application.
- The **SD Creation Tool** component is implemented with XFSC Organisation Credential Manager.
- The **SD Tooling UI** component is implemented as an Angular frontend application.

Signer Service

- The **Signer service** component is implemented with XFSC Organisation Credential Manager.

Wallet

- The **Wallet** component is implemented with XFSC Organisation Credential Manager.

Policy Template Datastore

- The **Policy template datastore** component is implemented with XFSC Organisation Credential Manager.

Query Mapper Adapter

- The **Query mapper adapter** component is implemented with Spring Cloud Gateway.

Schema Registry

- The **Schema registry** component is implemented as a File System.

Policy Template Datastore

- The **Policy Template Datastore** component is implemented as a File System.

Contract Template Datastore

- The **Contract Template Datastore** component is implemented as a File System.

Vocabulary Management

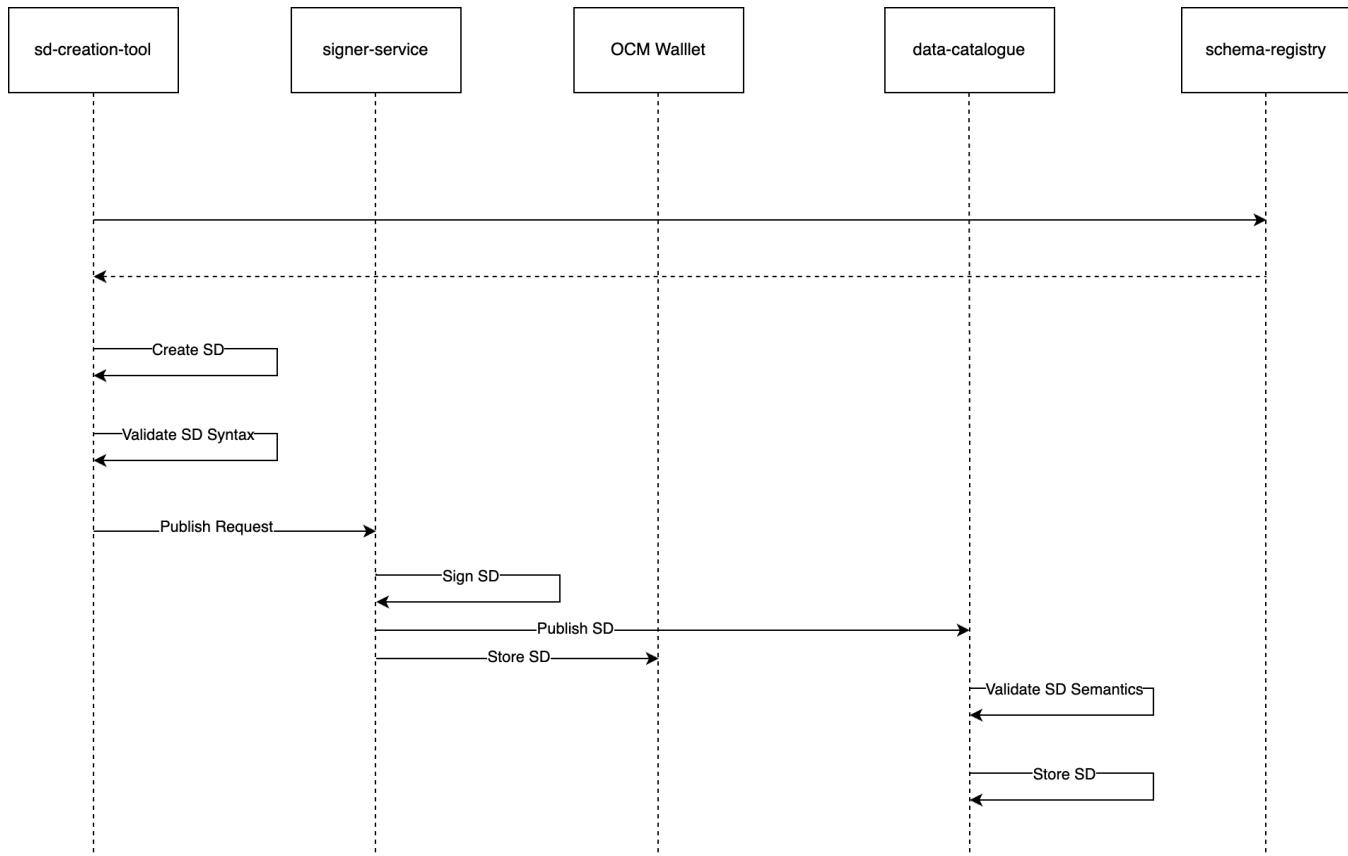
- The **Vocabulary management** component is implemented as a File System.
- The **Vocabulary management UI** component is implemented as an Angular frontend application.

Catalogue

- The **Search Engine** component is implemented with XFSC .
- The **Catalogue Database** component is implemented in PostgreSQL with Neo4J.
- The **Vocabulary Datastore** component is implemented as a File system.
- The **Management Service** is implemented with XFSC.
- The **Sematic Validation** service is implemented with RDFLib pySHACL.
- The **Quality rule validation** service is implemented with RDFLib pySHACL.
- The **Syntax Validation** service is implemented with RDFLib pySHACL.

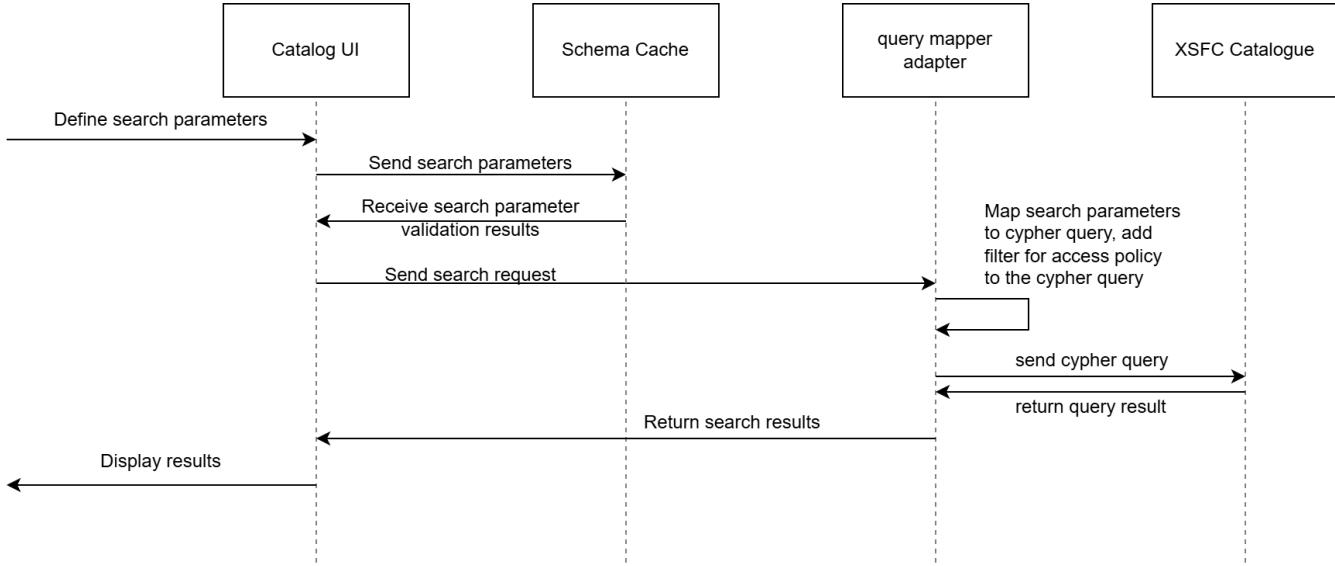
TCV Dynamic - BP 05A - Add or Update Resource (Publish) on Catalogue

This perspective illustrates the interactions and the flows between all the technological components.



TCV Dynamic - BP 06 - Search on Catalogue (Infrastructure, Data, Application)

This perspective illustrates the interactions and the flows between all the technologies.



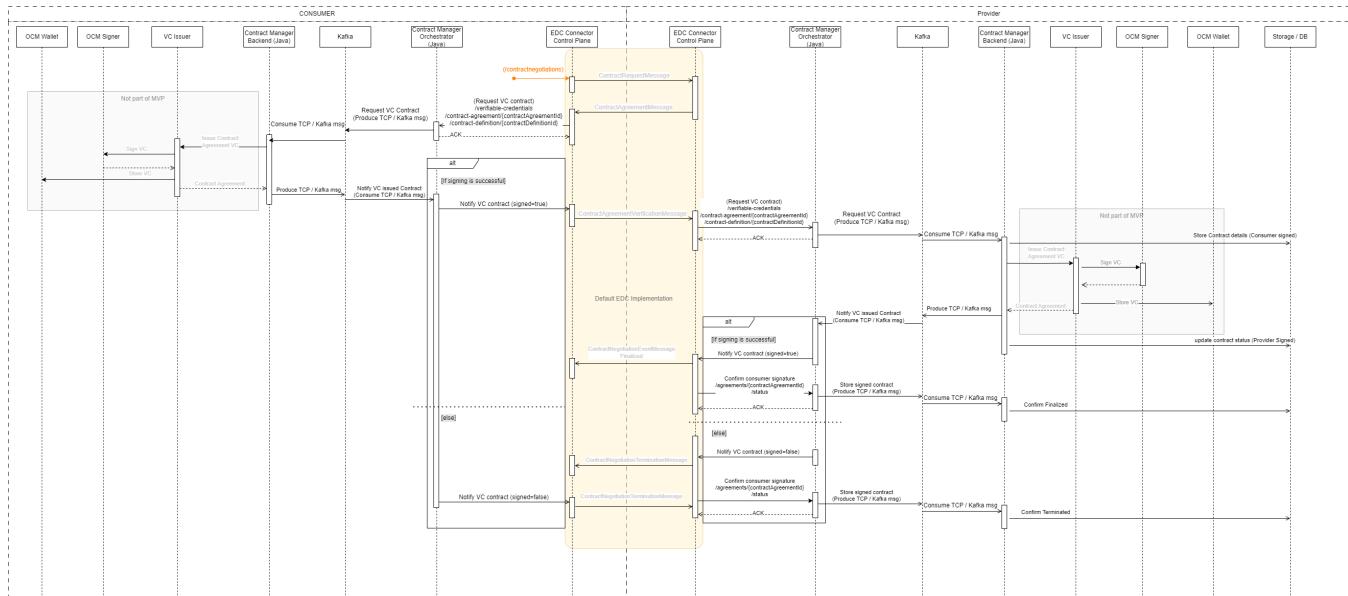
The search stack is split into a consumer/provider part and a centralized one.

The first one includes a client that offers a UI to the end user. The frontend application, for the advanced search, checks the parameters of the search with a local instance of the schema cache system previously synced with the instance present on the Governance Authority side. This allows us to perform a check on the parameters inserted in the Advanced search UI and send to the Query Mapper Adapter queries that are consistent with the schemas of the resources present in the data space. Furthermore on both sides is present an instance of the Spring Cloud API Gateway which takes care of securing the connection towards the other agent.

In the Governance Authority instance, apart from the already mentioned components, there is the Query Mapper & Filter Adapter which is in charge of translating the incoming query to the required query language and applying the filtering based on access policies. Then this last component redirects the resulting query to the API of the catalogue. The XSFC catalogue includes an internal search engine that will be used to perform the query on the Self-description present underneath Neo4J DB. The catalogue has behind also a Postgres DB for managing metadata and ensuring efficient file identification and data consistency.

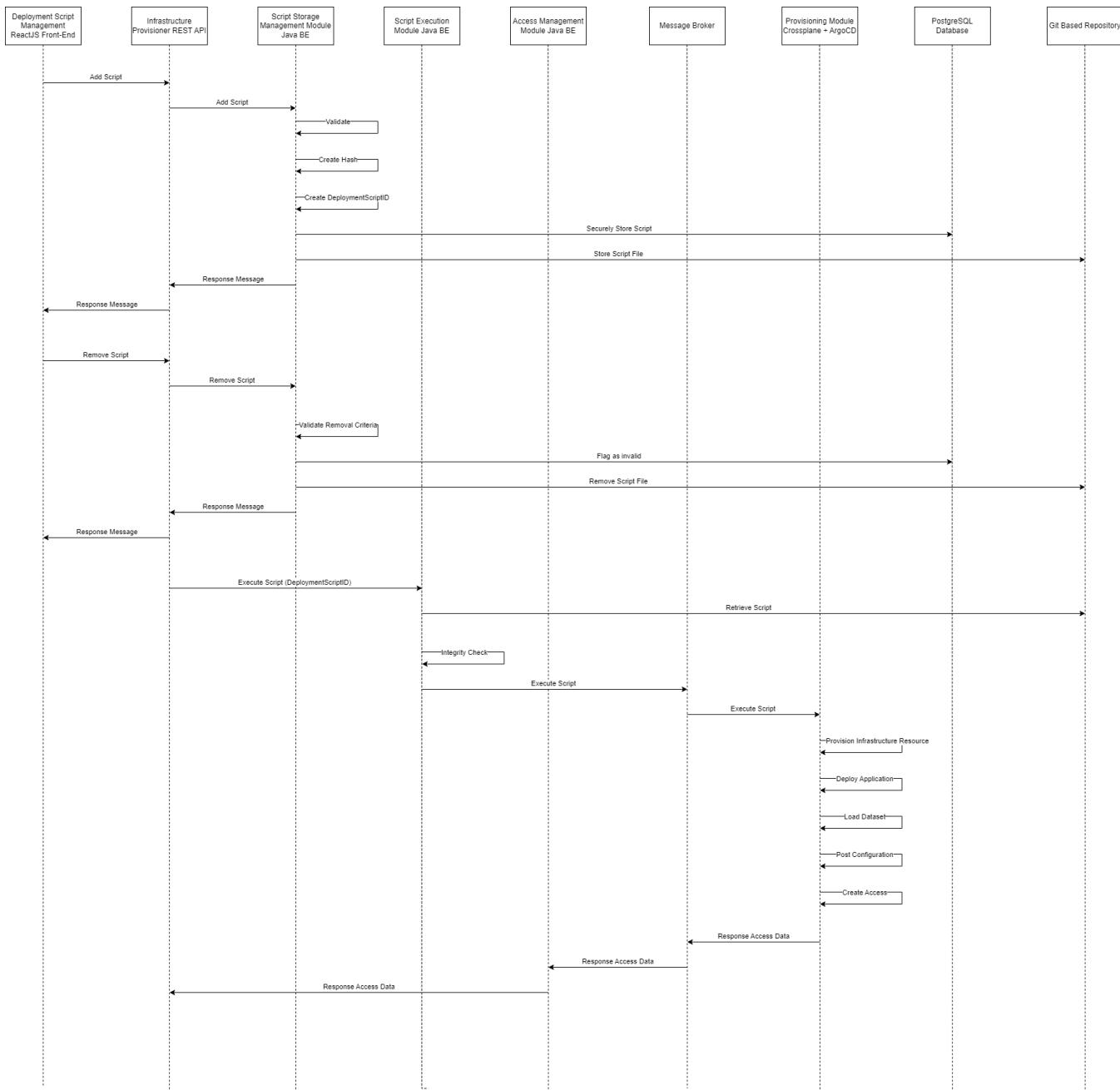
TCV Dynamic - BP 07A - Establish a usage contract agreement

This perspective illustrates the interactions and the flows between all the technological components.



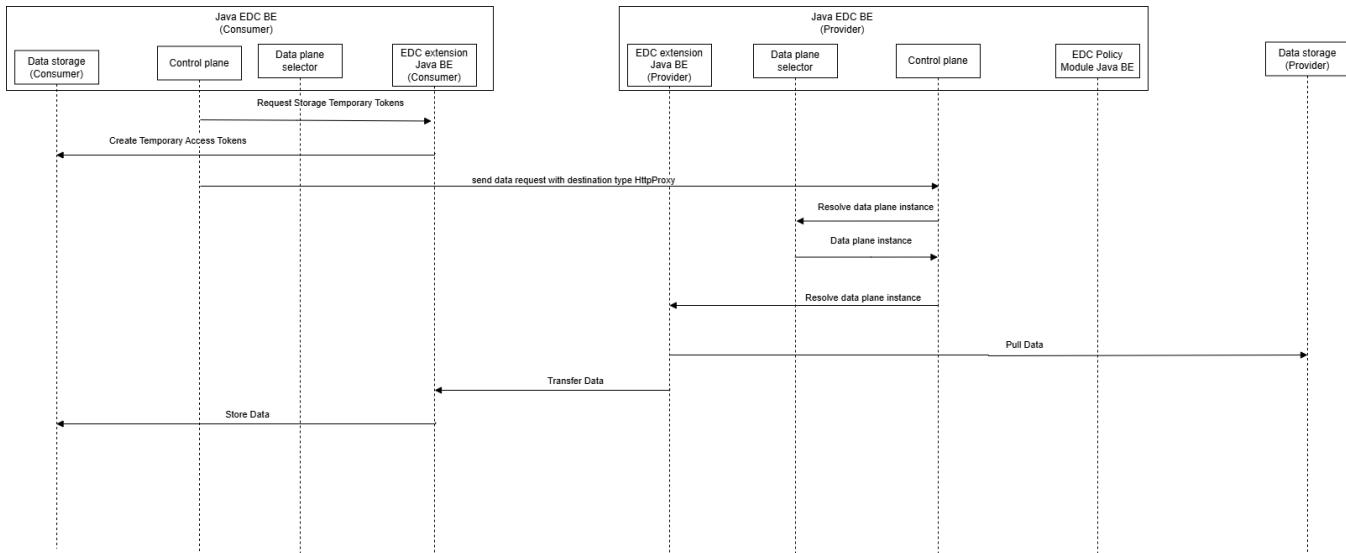
TCV Dynamic - BP 08 - Consumers select and use an Infrastructure Catalogue Resource from the Infrastructure Provider

This perspective illustrates the interactions and the flows between all the technological components.



TCV Dynamic - BP 09A - Data Resource Consumption - Consumer consumes a data resource from the provider

This perspective illustrates the interactions and the flows between all the technologies.



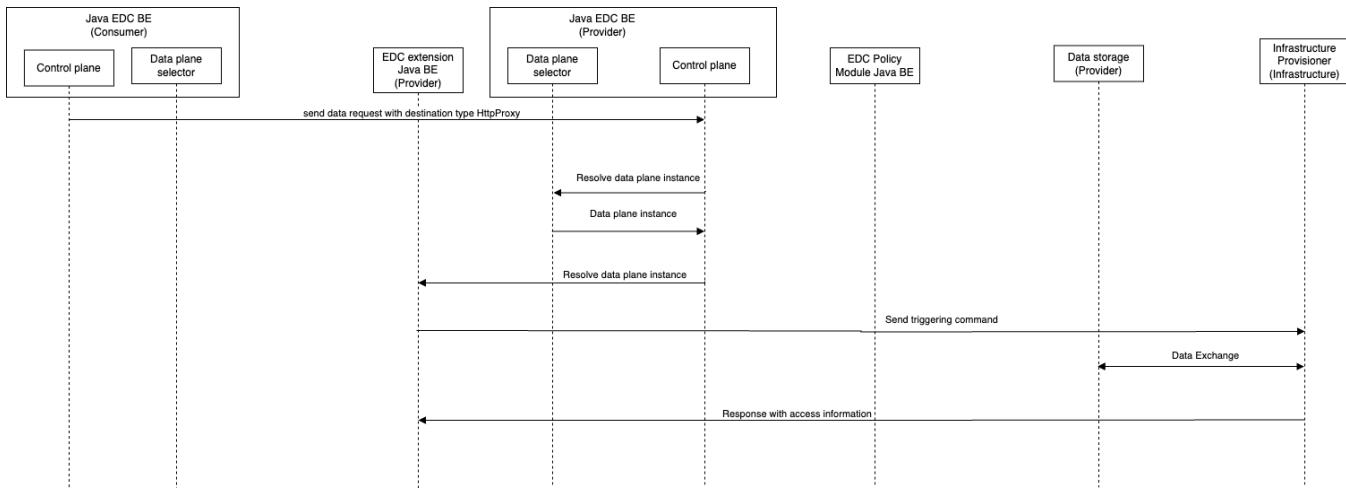
The data consumption BP9A is mainly addressed by the EDC connector Java backend.

The EDC connector is in charge of various steps of the DataSpace protocol. In particular, the core of the backend is the control plane that is in charge of the contract negotiation and the selection of the correct data plane depending on the type of resource requested while the actual data transfer is performed by the selected EDC connector Java extension which will connect to the real data source.

In the consumption process policies should be checked and this action is performed by the Policy Module present in the EDC connector Java backend.

TCV Dynamic - BP 09B - Data Resource Consumption - Consumer receives data processing service over a dataset via an Application

This perspective illustrates the interactions and the flows between all the technologies.



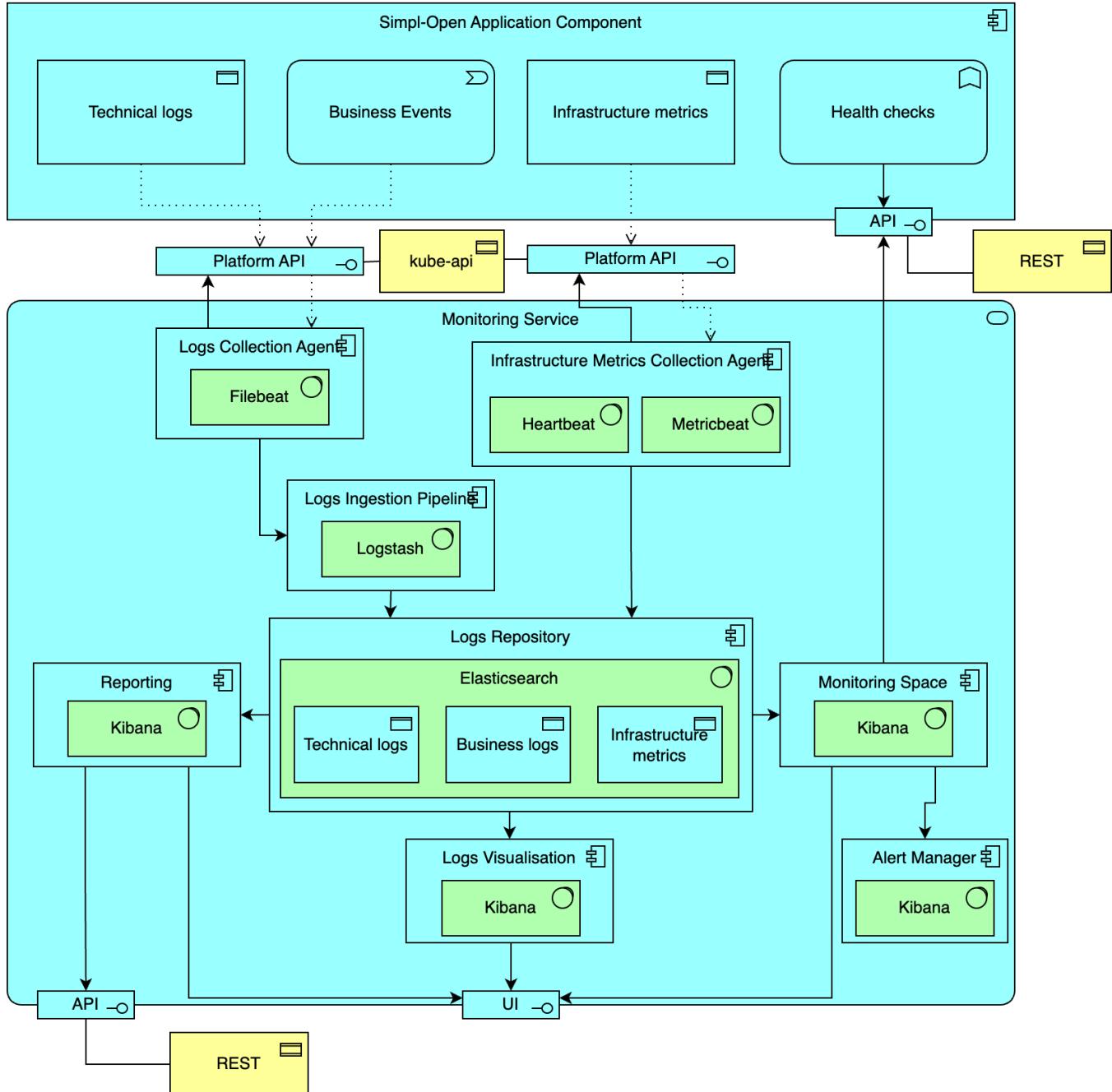
The data consumption BP which encompasses also the infrastructure provider is addressed by the infrastructure-related components (see BP8 for the details about this part).

The request from the user is directed to the Data provider EDC connector Java backend which forwards the request to the custom EDC extension connector that is capable of interacting with infrastructure Provider APIs.

TCV - Domain 3 - Management/Operation of data space

This section describes the architecture for Monitoring and Logging, within a single node (Simpl-Open agent) and does not (yet) consider inter-nodes setup.

This perspective illustrates the correlation between the architectural elements and the technologies, components, and interfaces intended for use in implementing the application components.



The monitoring service is based primarily on the Elastic stack.

Filebeat and Metricbeat are used to collect respectively technical logs and infrastructure metrics.

As Simpl-Open application services are deployed as containers in Kubernetes, both technical logs and infrastructure metrics are collected via the kube-api.

Technical logs are then forwarded to Logstash for processing and potential transformation. Business logs are directly sent by the application services to Logstash.

Metricbeat, Heartbeat and Logstash forward respectively infrastructure metrics and logs (technical and business) to Elasticsearch which acts as central logs repository.

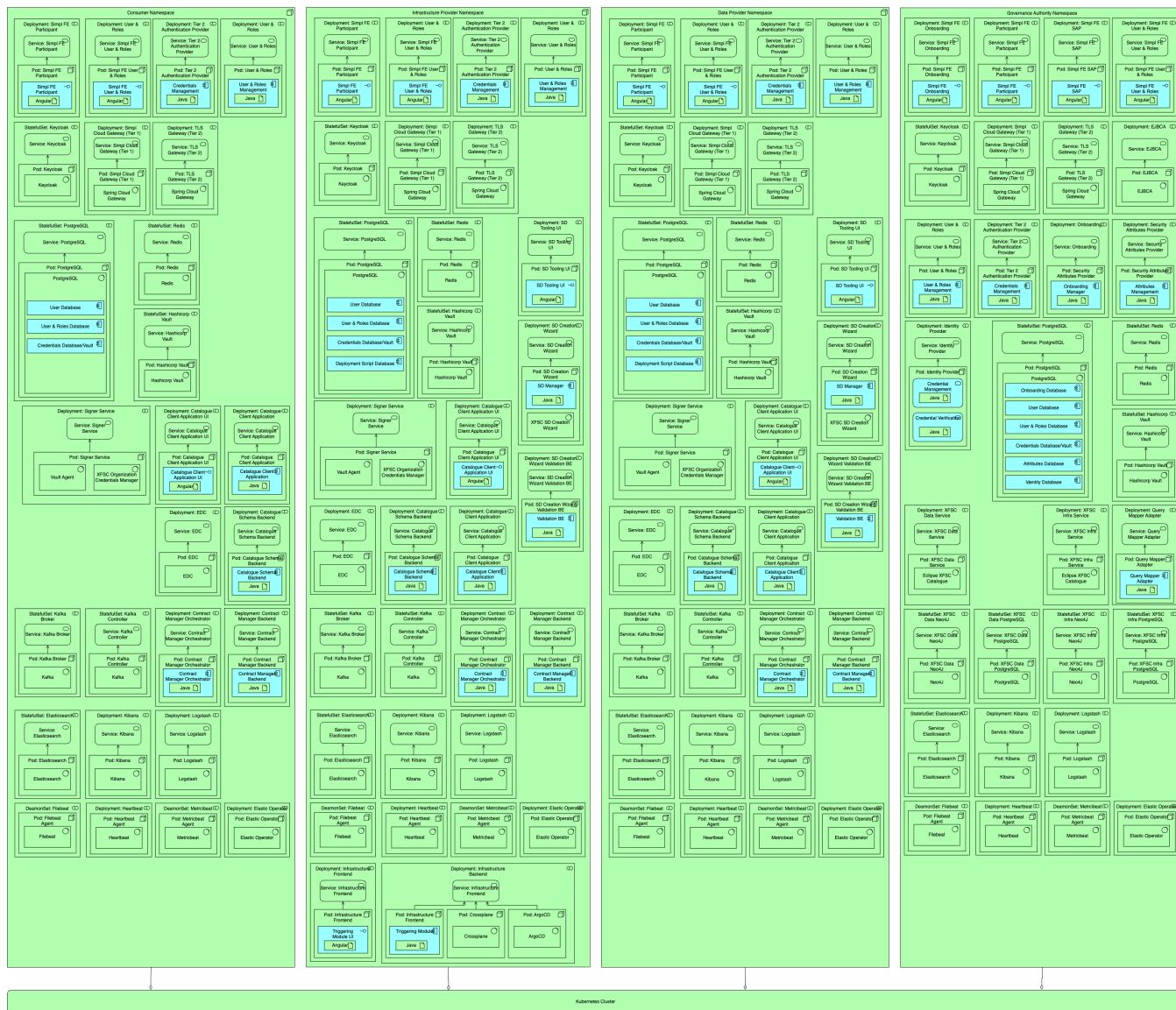
Kibana is used as user interface to provide reporting, log visualisation, monitoring space and alerting capabilities. Kibana also queries health endpoints of the services, exposed as REST/JSON APIs, to display their health in a dashboard.

A custom reporting application exposes a REST/JSON API to query logs for other purposes such as monitoring federation (i.e. forwarding some logs to the Governance Authority) or billing.

Technology Deployment View

Information: The content presented in this section presents a view on the MVP (December 2024) for the GA, Data Provider, Infra Provider and Consumer. Application Provider view fall behind the scope of the MVP.

The following Technology Deployment View describes how the different technology components are deployed for all Simpl-Open agent types (Governance Authority, Data Provider, Infrastructure Provider, Application Provider, Consumer):



Simpl-Open is designed to be a container-native application and is provided with all the required deployment artefacts to be deployed on a pre-existing **Kubernetes Cluster**.

Each agent is deployed inside its own **Kubernetes Namespace**.

Three types of Workloads are used:

1. **Deployment** - used for managing a stateless application workload, where any Pod in the Deployment is interchangeable and can be replaced if needed.
2. **StatefulSet** - used to run one or more related Pods that do track state somehow (for example, if the workload records data persistently). StatefulSet can match Pods with PersistentVolumes.
3. **DaemonSet** - used for Pods that provide facilities that are local to nodes. Every time a node is added to the cluster and it matches the specification in a DaemonSet, the control plane schedules a Pod for that DaemonSet onto the new node. Each pod in a DaemonSet performs a job similar to a system daemon on a classic Unix / POSIX server.

Kubernetes Services are used to expose certain components, running as one or more pods, behind a single outward-facing endpoint, even when the workload is split across multiple nodes.

Technology Open-Source Products

The present section is divided in 2 parts:

- Roadmap of 3 Years with draft consideration about Open-Source Software product selection;
- Open-Source Product Decision, as architecture is further analysed and components / interface are confirmed.

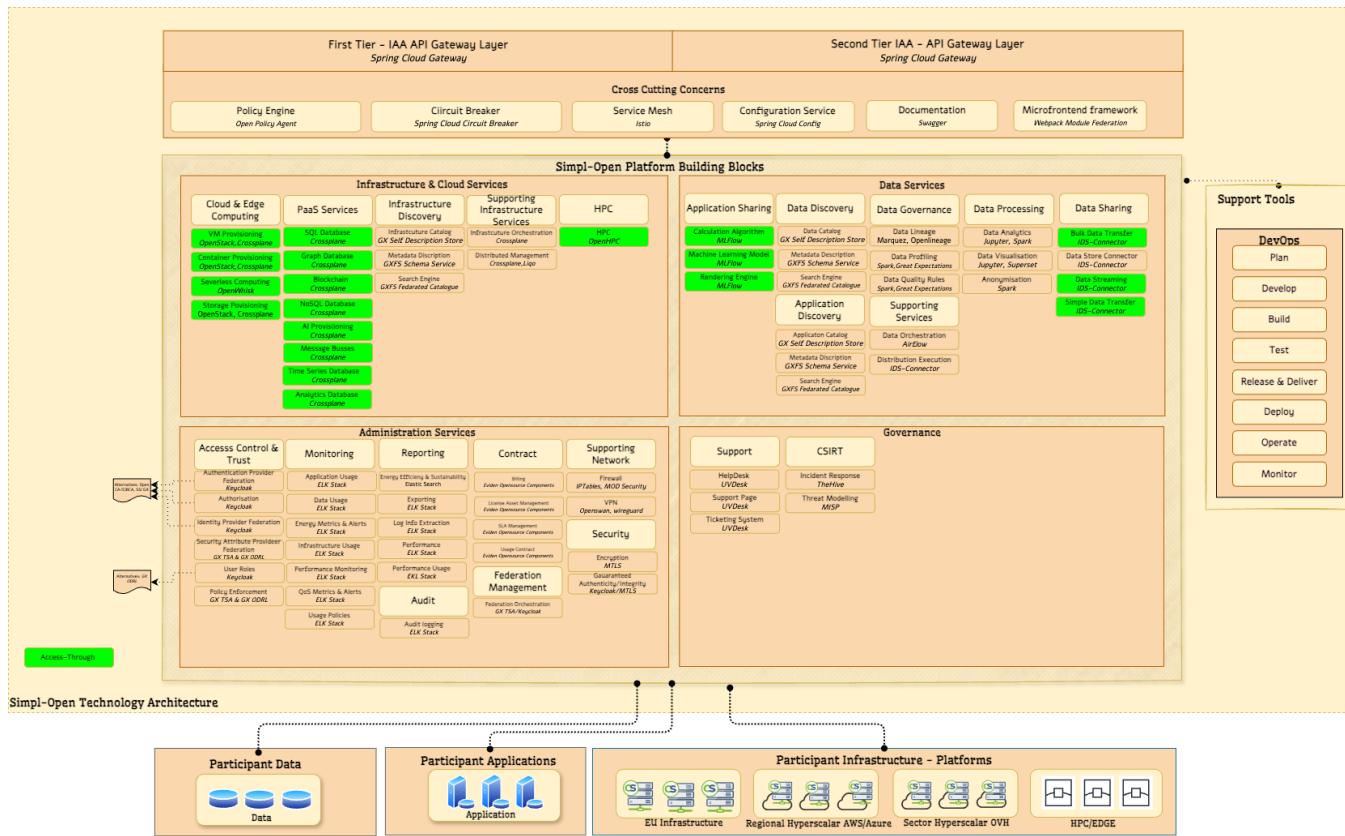
We assume to be valid the Roadmap OSS selection, until the respective capabilities are confirmed or amended by the Decisions that will happen in Agile fashion quarter by quarter.

Simpl-Open Technology Roadmap

The following illustration presents the Draft 3 Years Roadmap of Open-Source Software product selection to implement the functional capabilities required by Simpl-Open.

Also below is presented the table with the rational for selection, available today.

As general process, quarter by quarter, release by release, the Architecture team will further analyse capability by capability and confirm or amend selection based on detailed requirement and detailed architecture, including interaction with other technologies/components.



The draft table below provide a first rationale of selection identified as preliminary stages.

Tools	Description	Rationale
Event-driven Open Source	Partitum is a Proven solution component of the Eviden Clearing house as a service. This solution is currently running at Athumi (Belgium – Flanders). A data space intermediate, that is responsible for securely exchanging data between the different actors in a data space community and monetisation.	No integrated toolset available in the market matching client requirements.
Source code	The product provides the necessary tools to remove financial burden for the actors by:	

	<ul style="list-style-type: none"> Onboard the different actors in your eco-system and taking care of the contractual and financial agreements necessary to exchange data; Clearing of transactions based upon contractual agreements between the actors and their risk profile; Settlement of executed transactions between different actors; Automatically invoicing through billing or self-billing. 	
D A P S	Issue dynamic identity attributes based on scoped request.	It fits the second authentication mechanism described in Annex III of the "Architecture Vision Document" where identity attributes are dynamically by the Identity Attributes along with an ephemeral proof.
EJ B CA	Public key infrastructure certificate authority software. https://www.ejbca.org/	It is needed in all the envisioned authentication mechanisms between Participants as they require the issuance of a x.509 certificate.
Ke ycl oak	Identity and Access Management software. https://www.keycloak.org/	This component will manage the authentication and authorisation of the End Users. It can be easily federated with existing Participants' identity providers and extended to implement several types of authentication mechanisms (2FA, Digital Wallet, etc.).
EL K St ack	https://www.elastic.co/elastic-stack/	As suggested by Tenders Specifications and based on Market Standard.
Pr o m et he us	https://prometheus.io/	As suggested by Tenders Specifications and based on Market Standard.
Gr af ana	https://grafana.com/	As suggested by Tenders Specifications and based on Market Standard.
M TLS	Mutual TLS (mTLS) is a security practice that provides encrypted communications between every workload and application in your infrastructure, regardless of location.	Recognised protocols by several Open-Source products.
Cr os spl ane	Crossplane enables cloud-agnostic infrastructure provisioning and management. https://www.crossplane.io/	To abstract away cloud-specific APIs, enabling consistent control of resources across various cloud providers. It empowers DevOps teams to define infrastructure as code (IaC) and easily manage multi-cloud environments, enhancing agility and reducing vendor lock-in.
Terraform	Terraform automates infrastructure as code, simplifying provisioning and scaling. https://www.terraform.io/	For its declarative IaC approach, enabling infrastructure automation through code. Terraform's extensive provider ecosystem ensures broad cloud support and efficient orchestration, facilitating rapid scaling and reducing operational overhead.
An sib le	Ansible orchestrates application deployment and configuration with minimal complexity. https://www.ansible.com/	Agentless automation for simplified application provisioning and configuration management. Ansible's idempotent playbooks, robust modules, and YAML-based syntax simplify complex tasks, ensuring consistency and efficient operations across infrastructure.
Ku be rn et es	Kubernetes is a container orchestration platform, simplifying application deployment and scaling. https://kubernetes.io/	For containerised workload management and orchestration. Its advanced features, including auto-scaling, rolling updates, and service discovery, simplify application lifecycle management and enhance resource utilisation, making it a top choice for container-based applications.
U FW	Uncomplicated Firewall (UFW) simplifies firewall management for Linux systems.	Straightforward firewall rule management on Linux. Its user-friendly interface and uncomplicated syntax make it a powerful tool to secure systems against unwanted network traffic while simplifying the configuration of firewall policies.
Wi re G	WireGuard offers secure, efficient VPN solutions for network privacy and protection.	To secure network communications with state-of-the-art cryptography. lightweight design, minimal attack surface, and dynamic routing capabilities to provide robust VPN security, ensuring high-speed, low-latency connections for infrastructure.

ua rd	https://www.wireguard.com/	
nft ab les	nftables is a versatile packet filtering framework for fine-grained network control.	For advanced network filtering and routing. Its expressive syntax and performance optimisations help network administrators to efficiently manage packet filtering, firewall rules, and network address translation (NAT).
M od Se cu ri ty	ModSecurity provides web application firewall (WAF) protection against online threats.	To secure web applications with robust WAF capabilities. Its comprehensive rule sets and real-time threat detection safeguard applications from web-based attacks, ensuring data integrity and user trust.
Ce ph	Ceph is a distributed storage system for scalable, reliable data storage. https://ceph.io/en/	For cost-effective, highly available storage solutions. Its distributed architecture, erasure coding, and RADOS (Reliable Autonomic Distributed Object Store) technology deliver scalable, fault-tolerant storage, making it ideal for cloud and data-intensive workloads.
O K D (O pe nS hif t)	OKD, the open-source version of OpenShift, offers container orchestration and management. https://www.okd.io/	To deploy, manage, and scale containerised applications with Kubernetes simplicity. OKD's developer-friendly features, integrated CI/CD, and extensive ecosystem enhance DevOps workflows and application delivery, without worrying about the infrastructure.
O pe nS ta ck	OpenStack is an open-source cloud computing platform for building private and public clouds. https://www.openstack.org/	To create customisable, private cloud environments. The modular architecture provides flexibility and control over cloud resources, enabling tailored cloud solutions, reducing costs, and avoiding vendor lock-in.
Ku be less	Kubeless is a serverless framework for Kubernetes, enabling function-as-a-service (FaaS).	Serverless application development over Kubernetes. Simplifies event-driven, microservices-based architectures, providing rapid scaling and efficient resource utilisation, perfect for modern application workloads. Suitable for providers who are already running Kubernetes.
O pe nH PC	OpenHPC provides a comprehensive high-performance computing (HPC) stack for clusters.	To build and manage high-performance computing clusters. OpenHPC simplifies the integration of HPC software components, ensuring optimised performance for scientific and computational workloads.
O pe nW hisk	OpenWhisk is an open-source serverless platform with support for multiple programming languages. https://openwhisk.apache.org/	Serverless capabilities for flexible, event-driven application development. OpenWhisk's language-agnostic approach simplifies serverless computing, facilitating faster development and deployment of cloud-native functions.
eD eli ve ry	eDelivery helps public administrations to exchange electronic data and documents with other public administrations, businesses and citizens at the national level and across borders, in an interoperable, secure and reliable way.	Part of Digital Building Blocks from European Commission.
eS ig na ture	The DIGITAL eSignature Building Block allows public administrations, businesses, and citizens to electronically sign any document, anywhere in Europe, at any time, in line with the eIDAS Regulation for e-signatures, e-seals and related services offered by Trust Service Providers.	Part of Digital Building Blocks from European Commission.
el nv oi cing	The elnvoicing Building Block aims to promote the successful uptake of electronic invoicing in Europe, respecting the European standard on electronic invoicing and Directive 2014/55/EU on electronic invoicing in public procurement.	Part of Digital Building Blocks from European Commission.
eID	The eID Building Block allows public administrations and private service providers to easily extend the use of their online services to citizens from other Member States, in line with the eIDAS Regulation. In the digital age, public administrations and businesses need to carry out fast, secure electronic transactions and validate the identities of	Part of Digital Building Blocks from European Commission.

	those involved with the same legal validity as traditional paper processes. Electronic identification (eID) makes this possible.	
Eclipse EDC	The EDC connector is a software installed by the participating company or a platform thereby providing technical access to the ecosystem. A connector can consist of monolithic or self-contained software. https://github.com/eclipse-edc/Connector	As an open source project hosted by the Eclipse Foundation, the EDC provides a growing list of modules for many widely-deployed cloud environments (AWS, Azure, GCP, OTC, etc.) "out-of-the-box" and can easily be extended for more customised environments, while avoiding any intellectual property rights (IPR) headaches.
XFS C Federated Catalogue	The "Federated Catalogue" service includes a catalogue where Gaia-X resources, asset items, and participants can be found by potential consumers and end users. Resources, asset items and participants are provided at Gaia-X using self-descriptions. https://gitlab.eclipse.org/eclipse/xfsc/cat	The reference implementation of organisational Federated Catalogue supporting SD according to the Gaia-X Trustmodel.
piveau	piveau is a data management ecosystem for the public sector. https://www.piveau.de/en/	It provides components and tools to support the entire data processing chain from harvesting, aggregation, provision, and use. It is highly extensible, focuses on open standards and is designed for use in the cloud and reacts reliably and quickly to unforeseen access peaks.
XFS C OCM	The "Organisation Credential Manager" service establishes trust between the different participants within the decentralised Gaia-X ecosystem. It includes all trust-related functions required to manage and offer Gaia-X self-descriptions in the W3C Verifiable Credential Format. https://gitlab.eclipse.org/eclipse/xfsc/ocm	The reference implementation of organisational Credential Manager due to Gaia-X Trustmodel.
XFS C PCM	The "Credential Manager" service enables Gaia-X users to manage their credentials themselves. To do this, the user needs secure storage (user wallet) and presentation capabilities in the authentication and authorisation processes. https://gitlab.eclipse.org/eclipse/xfsc/pcm	The reference implementation of personal Credential Manager due to Gaia-X Trustmodel.
Apache Spark	Apache Spark is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters. https://spark.apache.org/	Apache Spark is highly adopted by thousands of companies. It also integrates with all important frameworks on Data Science and Machine Learning, SQL Analytics and BI and Storage and Infrastructure.
Great Expectations	A powerful platform to uphold data quality. https://greatexpectations.io/	Great Expectations offer broad flexibility and control when creating data quality tests. It also provides auto-updating documentation to ease reports of test suites and results in collaborative environments.
Marquez, OpenLineage	OpenLineage is an open platform for collection and analysis of data lineage. It tracks metadata about datasets, jobs, and runs, giving users the information required to identify the root cause of complex issues and understand the impact of changes. https://openlineage.io/	OpenLineage contains an open standard for lineage data collection, a metadata repository reference implementation (Marquez), libraries for common languages, and integrations with data pipeline tools.
MLflow	MLflow is an open-source platform to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry. https://mlflow.org/	MLflow offers several key components to access, evaluate, process and deploy Large Language Models (LLM).
JupyterLab	JupyterLab is a web-based interactive development environment for notebooks, code, and data.	

Jupyter	https://jupyter.org/	Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. A modular design invites extensions to expand and enrich functionality. This tool is highly adopted in the data science community
Superset	Apache Superset is an open-source modern data exploration and visualisation platform. https://superset.apache.org/	Superset is fast, lightweight, intuitive, and loaded with options that make it easy for users of all skill sets to explore and visualise their data, from simple line charts to highly detailed geospatial charts. It supports a wide range of data bases.
UVdesk	https://www.uvdesk.com/en/	open-source ITSM tool selected as the tool best matching the tender requirements.
TheHive	https://thehive-project.org/	Same toolset as the one used for cert.eu and other governments institutions.
MISP	https://www.misp-project.org/	Same toolset as the one used for cert.eu and other governments institutions.
Spring Cloud Gateway	software components that act as an API Gateway. https://spring.io/projects/spring-cloud-gateway	This component will manage the routing of API Requests to the several services that compose the SMP middleware. It is easily extendable and configurable in order to implement specific cross cutting concerns as security and the control of Access & Usage policies.
Spring Cloud Circuit Breaker	Library implementing the Circuit Breaker pattern and other HA patterns. https://spring.io/projects/spring-cloud-circuitbreaker	Mitigates high response times and network errors, enhancing system reliability. It implements the Circuit Breaker, Retry and Bulkhead patterns. It is useful for communication inside and outside the SMP Agent perimeter.
Web pack Module Federation	Technology enabling the creation of micro-frontends.	A common Application Shell will be implemented, that dynamically loads the several autonomous Front End modules. Each module can be mapped to a specific micro-service and developed independently by the same Team that is in charge of it, increasing the speed of development of distributed and scalable applications.
Aruba Consent Management	Consent management service.	It manages consent given by Data Providers to the Consumers. It binds consents to specific versions of a legal text. Data Providers can revoke their consent at any time. Specific events are raised for every notable change in the system, that can be easily reviewed and audited.
Spring Cloud Config	https://spring.io/projects/spring-cloud-config	

S wa gg er	https://swagger.io/	The de facto standard of documentation for REST APIs.
Data Mashup Editor (Engineering open source)	The mission of the Data Mashup Editor is to develop a powerful and intuitive graphical tool that simplifies the process of harmonising data from diverse sources, leveraging cutting-edge technologies and intelligent data integration techniques. The Data Mashup Editor is dedicated to ensuring data accessibility, usability, and accuracy, enabling informed decision-making across industries and domains and unlocking the true value of data assets.	The Data Mashup Editor was chosen as one of the tools for data processing building block and data sharing building block due to its ability to seamlessly handle both real-time and batch data streams, while redirecting the output to various entities adopting different technologies and protocols simultaneously. Its internal architecture makes it highly suitable for cloud deployment, ensuring optimal performance and distributed executions. Additionally, it offers an intuitive user experience through its graphical interface, making it easy for users to utilise the tool effectively.
Rule Manager (Engineering open source)	The Digital Enabler Rule Manager is a powerful tool designed for managing trigger rules and automated responses based on specific data values within your platform. This tool offers a user-friendly guided wizard for defining and implementing rules for data processing within the platform.	The Rule Manager was chosen as one of the tools for data processing building block and data sharing building block due to its capability to create rules of varying complexity based on the data within the system and this gives the possibility of adding a monitoring layer in the processing steps. It integrates seamlessly with the Data Mashup Editor, providing a comprehensive solution for data manipulation. Its internal architecture is well-suited for cloud deployment, ensuring excellent performance and distributed executions. Furthermore, its graphical interface provides users with an intuitive experience, simplifying the process of effectively utilising the tool.
Airflow	Apache Airflow is an open-source platform for developing, scheduling, and monitoring batch-oriented workflows. https://airflow.apache.org/	Airflow was chosen as the data orchestration component, in the supporting data services building block, due to its exceptional flexibility, allowing the installation of plugins as needed. Moreover, it seamlessly integrates with cloud architectures, providing excellent support for distributed execution in a microservices environment.

Simpl-Open Technology Choices

The table below presents the list of Open-Source Software used by Simpl-Open (MVP).

Initiative and Recognised Business Open-Source

Capability	Sub-Capability	Tool	Description	URL	Rationale	Additional Considerations
Discovery	Metadata	SD (GX-Trustframework)	Metadata of Participants and service offerings (App, Data, Infra) described as GAIA-X Self-Description using an ontology. The SD uses a linked data format and allows the definition of constraints and quality rules.	https://gaia-x.gitlab.io/policy-rules-committee/trust-framework/	<ul style="list-style-type: none"> • Licence: Creative Commons • Community Support: Gaia-X • Documentation Available: here • Extensibility: yes • Adoption by Business: Gaia-X Light house, all data space initiatives claiming to be GAIA-X compliant. 	Ontology highly adopted in data space initiatives. Best choice to convince participants to provide self-descriptions in this way. It can be easily enhanced with sectoral specific parameters.
Discovery	Metadata	XFSC SD Tooling	Tooling to create self descriptions describe the service offerings (Data, App, Infrastructure).	https://gitlab.eclipse.org/eclipse/xfsc/self-description-tooling	<ul style="list-style-type: none"> • Licence: Apache 2.0 • Community Support: XF SC • Documentation Available: yes • Extensibility: yes 	No other FOSS tool available to create customised SD. Schemas can be created via LinkML Generator Tool Fully customisable SD definitions possible.

						• Adoption by Business: TrustedCloud (Spec)	
Discovery	Catalogue at Governance Authority	XFSC Federated Catalogue	Federated Catalogue providing Discovery capability to look up on Self Descriptions of service offerings (Data, App, Infrastructure).	https://gitlab.eclipses.org/eclipse/xfsc/cat	<ul style="list-style-type: none"> • Licence: Apache 2.0 • Community Support: XF SC • Documentation Available: Web, PDF • Extensibility: yes • Adoption by Business: Gaia-x Lighthouse 	<p>The only implementation of a FOSS federated catalogue supporting SD, i.e. validation of SD when published and searching for SD providing an internal search engine. It also already support semantic validation. In addition the search engine is based on NoSQL which provides the base for knowledge search needed for M2M use cases.</p> <p>CKAN is using PostgreSQL as database. Hence it is not well prepared for ontology searches. There are plugins available to enable limited Ontology search capabilities like SparQL extensions. However, they do not scale and will fail on complex knowledge graph search as needed for ML algorithms.</p> <p>Either a PropertyGraph Database like Neo4J or an RDF-Triple Storage like Apache Fuseki Jena, Virtuoso etc. is needed.</p>	
Discovery	Credential Manager at Provider	XFSC OCM	The credential manager to store the Self Descriptions on organisational side. It also covers signing of Self Descriptions created by a provider, revoking a credential, verification and retrieval of credentials as microservices.	https://gitlab.eclipses.org/eclipse/xfsc/organisational-credential-manager-w-stack	<ul style="list-style-type: none"> • Licence: Apache 2.0 • Community Support: XF SC • Documentation Available: Web • Extensibility: yes • Adoption by Business: 	<p>This is created as part of XFSC matching the needs best for SD. Can be easily replaced with any other wallet solution providing the same protocols in exchanging credentials (OIDC4VF and OIDC4VC).</p>	
Access control & trust	Authentication Provider	Keycloak	<p>Open-Source Identity and Access Management</p> <p>Add authentication to applications and secure services with minimum effort.</p> <p>No need to deal with storing users or authenticating users.</p> <p>Keycloak provides user federation, strong authentication, user management, fine-grained authorisation, and more.</p>	https://www.keycloak.org/	<ul style="list-style-type: none"> • License: Apache 2.0 • Community Support: Huge community based upon years of being active (21K stars on github)https://www.keycloak.org/community • Documentation Available: Documentation is wide and focus on every aspects of the tool https://www.keycloak.org/documentation • Extensibility: yes natively and through REST API(https://www.keycloak.org/docs/latest/server_development/index.html) • Adoption by Business: Spread adoption around the world 	<p>An on-premise solution that is a de facto standard and offers a wide-range set of features and a native(java) extensible interface.</p>	
Provisioning	VM/Container /Storage provisioning	Crossplane	Crossplane is an open-source Kubernetes add-on that allows to define and automate the infrastructure using Kubernetes-style configuration files. It extends the Kubernetes API to allow to provision and manage cloud resources and services from various providers, such as AWS, GCP, Azure, and more, in a unified manner.	https://www.crossplane.io/	<ul style="list-style-type: none"> • License: Apache 2.0 • Community Support: Huge community based upon years of being active (9.6K stars on github)https://www.crossplane.io/community • Documentation Available: https://docs.crossplane.io/ • Extensibility: Yes, highly extensible with support for custom resource definitions 	<p>Multi-cloud environment operation.</p> <p>Crossplane simplifies infrastructure management by bringing the benefits of the Kubernetes declarative model to cloud provisioning. By using Crossplane, teams can leverage the familiar Kubernetes tools and workflows to manage infrastructure alongside their applications, leading to a more consistent, scalable, and efficient infrastructure management process.</p> <p>Crossplane is favored over Terraform (https://blog.crossplane.io/crossplane-vs-terraform/), also because of more permissive license.</p>	

					(CRDs) and integration with various cloud providers and on-premises environments.	
Provisioning	VM/Container /Storage provisioning	ArgoCD	ArgoCD is a declarative, GitOps continuous delivery tool for Kubernetes. It is designed to simplify the process of deploying and managing applications on Kubernetes clusters. ArgoCD uses a GitOps approach, which means it uses Git repositories as the source of truth for application configurations.	https://argo-cd.readthedocs.io/en/stable/	<ul style="list-style-type: none"> Adoption by Business: Increasingly adopted by organizations seeking Kubernetes-native solutions for infrastructure automation and application management. 	
Provisioning	Post Configuration / Application Deployment	Cloud-init	Cloud-init is a popular tool for automating the initialization and configuration of cloud instances. It is designed to simplify the process of deploying and configuring cloud instances, and is widely used in cloud computing environments.	https://cloud-init.io/	<ul style="list-style-type: none"> License: Apache 2.0 Community Support: Large and active community with many contributors and users Documentation Available: Extensive documentation available, including user guides, API references, and tutorials Extensibility: Highly extensible with support for custom plugins and integrations Adoption by Business: Widely adopted by businesses and organizations, including many Fortune 500 companies 	Cloud-init is selected for its ability to automate the initialization and configuration of cloud instances, as well as run post-provisioning tasks such as deploying or installing applications, automated network configuration, storage setup, and security hardening. Its modular and customizable approach ensures that instances are properly configured and secured, reducing the risk of errors and improving overall reliability.

					DevOps spaces	
Provisioning	Storage (Repository)	Gitea	Gitea is a lightweight, open-source, and highly extensible repository management tool that provides a simple and intuitive way to manage code repositories. It offers a web-based interface for creating, managing, and organizing code repositories, and provides features such as collaboration, version control, and issue tracking.	https://about.gitea.com/	<p>License: MIT License Community Support: Large and active community with many contributors and users Documentation Available: Extensive documentation available, including user guides, API references, and tutorials Extensibility: Highly extensible with support for custom plugins and integrations Adoption by Business: Widely adopted by businesses and organizations, particularly in the open-source and developer communities</p>	Gitea is chosen for its ability to provide a lightweight, flexible, and highly extensible repository management solution. Its ease of use, scalability, and customizability make it an ideal tool for managing code repositories.
Data Exchange	Data Exchange Service	EDC	The data exchange service implementing the negotiation protocol (data space protocol).	https://projects.eclipse.org/projects/technology.edc	<ul style="list-style-type: none"> • Licence: Apache 2.0 • Community Support: Tractus-X and EDC • Documentation Available: Tractus-X and EDC • Extensibility: well structured interfaces to customise component • Adoption by Business: Catena-X, Eona-X, several other data initiatives using forks of it. Known Friends of EDC. 	<p>Can be replaced with any other IDS connector implementing the IDSA Dataspace Protocol and using ODRL expressions for policy . The EDC connector is chosen because it has a good documentation, provides good interfaces and can be easily customised. Second there are two joined active communities to drive the development: Tractus-X and EDC.</p> <p>In addition, the first IDS connector passing the IDSA certification was the TSI connector based on EDC.</p> <p>Also EDC is the only available IDS connector which has already implemented the dataspace protocol. Other initiatives will follow.</p>
Monitoring, Logging, Reporting, Audit	Monitoring and Logging	ELK (Elastic, Logstash, Kibana)	Reliably and securely take data from any source, in any format, then search, analyze, and visualize.	https://www.elastic.co/	<ul style="list-style-type: none"> • License: EL v2 / SSPL • Community Support: Largest community in the industry https://www.elastic.co/fr/community • Documentation Available: Yes https://www.elastic.co/docs • Extensibility: yes • Adoption by Business: Most adopted open sources logging /monitoring /reporting /auditing stack in the world. 	The ELK stack is an industry standard for log management and data analysis due to its scalability and powerful features. Elasticsearch handles large volumes of data with real-time search and analytics, Logstash processes and ingests data from various sources, and Kibana provides intuitive visualizations and reporting. Being open-source, it benefits from a large community, continuous improvements, and extensive plugins. Security features like TLS encryption, role-based access control, and audit logging ensure data protection, making ELK a reliable and versatile solution for diverse use cases.
Access control & trust	Commons	HashiCorp Vault	Used by Keycloak, EJBCA, Spring Cloud Gateway, and when access to stored credentials is needed by a Java Backend.	https://www.vaultproject.io/	<ul style="list-style-type: none"> • License: Business Source License (BSL) for Open Source • Community Support: High, with active forums and GitHub 	HashiCorp Vault is a secrets management and encryption platform that securely stores, manages, and encrypts sensitive data such as passwords, API keys, and certificates. It provides secure access, auditing, and revocation of secrets across distributed infrastructure, applications, and services, enabling secure development, deployment, and operation of modern systems.

					<p>discussions. https://discuss.hashicorp.com/c/vault/</p> <ul style="list-style-type: none"> • Documentation Available: Yes, extensive documentation covering all aspects of setup, usage, and integration. https://developer.hashicorp.com/vault/docs • Extensibility: Yes, supports plugins and integrations with major cloud platforms and authentications on systems. • Adoption by Business: Widely adopted globally, especially in industries with strict compliance and security requirements. 	
Access control & trust	Commons	MinIO	MinIO is a high-performance, S3 compatible object store. It is built for large scale AI/ML, data lake and database workloads. It is software-defined and runs on any cloud or on-premises infrastructure.	https://min.io/	<ul style="list-style-type: none"> • License: GNU AGPL v3 • Community Support: High, with active GitHub issues, a strong community forum, and Slack channels. https://slack.min.io/ • Documentation Available: Yes, comprehensive and detailed documentation covering deployment, configuration, and API usage. https://min.io/docs/ • Extensibility: Yes, supports integrations with multiple cloud platforms, Kubernetes, and third-party tools for storage and analytics. • Adoption by Business: Growing adoption worldwide, particularly in data-driven industries leveraging object storage for modern workloads like AI/ML, big data, and cloud-native applications. 	Min.io is an open-source, Amazon S3-compatible, distributed object storage server for cloud-native and edge computing applications. It provides a highly available, scalable, and performant storage solution with features like erasure coding, bitrot protection, and encryption, making it suitable for a wide range of use cases, from dev to production.
Access control & trust	Commons	PostgreSQL	used by Keycloak, EJBCA, Spring Cloud Gateway, and when a DB is needed by a Java Backend	https://www.postgresql.org		The World's Most Advanced Open-Source Relational Database

				postgresql.org/	<ul style="list-style-type: none"> • License: PostgreSQL License • Community Support: Very high https://www.postgresql.org/community/ • Documentation Available: Yes, documentation is wide and focuses on every aspect of the Database https://www.postgresql.org/docs/ • Extensibility: yes • Adoption by Business: Spread adoption around the world 	
Access control & trust	Common Identity provider	EJBCA	One of the world's most popular PKIs, EJBCA gives you time-proven flexibility and robustness. Unlike other open-source certificate authority and PKI solutions, EJBCA is platform-independent and can be scaled up and down to match your needs.	https://www.ejbcna.org/	<ul style="list-style-type: none"> • License: LGPL 2.1 • Community Support: Huge community based on the mailing list, github, forums, and slack channel • Documentation Available: Documentation is wide and helps understand how to use and interact with the tool https://docs.keyfactor.com/ejbcna/latest/ • Extensibility: yes using REST API • Adoption by Business: 	The most mature (23 years), used, and rich in features, java-based PKI solution in the open-source panorama.
Access control & trust	Commons Authorisation	Spring Cloud Gateway	It is a spring project that provides libraries for building an API Gateway on top of Spring WebFlux or Spring WebMVC. Spring Cloud Gateway aims to provide a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as: security, monitoring/metrics, and resiliency.	https://spring.io/projects/spring-cloud-gateway	<ul style="list-style-type: none"> • License: Apache 2.0 • Community Support: Huge community based upon the spring framework community https://spring.io/community • Documentation Available: Documentation is wide and focuses on every aspect of the framework https://docs.spring.io/spring-cloud-gateway/reference/ • Extensibility: yes very high • Adoption by Business: Spread adoption around the world 	Based upon the best java-based backed framework in the world (spring) it also offers a reactive implementation that ensures the maximum level of resiliency and extensibility.
Message Broker	Commons	Apache Kafka	Apache Kafka is an open-source distributed event streaming platform designed for building real-time data pipelines and streaming applications. It serves as a high-throughput, fault-tolerant, and horizontally scalable platform that can handle large volumes of data and stream events in real-time. Kafka uses a publish-subscribe model and durable storage for storing and	https://kafka.apache.org/	<ul style="list-style-type: none"> • License: Apache 2.0 • Community Support: Huge 	Apache Kafka's role as a message broker offers several advantages for handling asynchronous events and message-based communication within distributed systems: Scalability: Kafka's distributed architecture allows for horizontal scaling, enabling high throughput and low latency message processing even under heavy loads.

			<p>processing streams of records.</p> <p>Message Brokerage: In addition to its streaming capabilities, Kafka can effectively serve as a message broker, facilitating communication between different components of a system through the asynchronous exchange of messages. It provides features like message queuing, topic partitioning, and consumer group management, making it suitable for implementing a decoupled, event-driven architecture.</p>		<ul style="list-style-type: none"> Community: 29K stars on github https://kafka.apache.org/ Documentation: Available: yes - https://kafka.apache.org/documentation/ Extensibility: yes Adoption by Business: Spread adoption around the world 	<p>Durability: Messages are stored durably in Kafka, providing fault tolerance and preventing data loss in case of system failures.</p> <p>Reliability: Kafka ensures reliable delivery of messages to consumers through features like message retention and configurable acknowledgment settings.</p> <p>Decoupling: By decoupling producers and consumers through topics, Kafka enables loosely coupled communication between system components, improving flexibility and resilience.</p> <p>Real-time Processing: Kafka's ability to process and react to events in real-time makes it suitable for use cases requiring low-latency messaging, stream processing, and complex event-driven architectures.</p>
Cache	Commons	Redis	<p>Redis Cache is an in-memory data structure store widely used as a caching solution to enhance the performance of applications. By storing frequently accessed data in memory, Redis enables faster data retrieval compared to disk-based databases. It supports a variety of data types such as strings, lists, sets, and hashes, making it versatile for different caching needs. Redis is known for its high throughput, low latency, and scalability, often used for caching web pages, session management, real-time analytics, and message brokering. It also supports persistence, replication, and automatic failover for reliability.</p>	https://redis.io/	<ul style="list-style-type: none"> License: Redis Source Available License 2.0 (RSALv2) Community Support: high Documentation: Available: high Extensibility: yes Adoption by Business: Spread adoption around the world 	<p>Redis Cache offers several advantages for improving application performance and scalability.</p> <p>Performance: As an in-memory data store, Redis delivers extremely low-latency and high-throughput data retrieval, significantly boosting application speed.</p> <p>Scalability: Redis supports horizontal scaling through clustering and partitioning, allowing it to handle large datasets and heavy traffic efficiently.</p> <p>Flexibility: With support for various data structures such as strings, lists, sets, and hashes, Redis can handle diverse caching and real-time data processing use cases.</p> <p>Persistence and Reliability: Redis offers optional persistence mechanisms like snapshots and append-only files, ensuring durability, while replication and automatic failover provide high availability and fault tolerance.</p> <p>Integration: Redis integrates easily with various programming languages and frameworks, making it a popular choice for developers seeking an efficient, easy-to-deploy caching solution.</p>

The following table links the OSS components to their architecture documentation and installation guide:

OSS	Architecture Documentation	Installation Guide
XFSC Signer	https://gitlab.eclipse.org/eclipse/xfsc/tsa/signer	https://gitlab.eclipse.org/eclipse/xfsc/tsa/signer/-/blob/main/deployment/helm/README.md
XFSC Federated Catalogue	https://gaia-x.gitlab.io/data-infrastructure-federation-services/cat/architecture-document/architecture/catalogue-architecture.html	https://gitlab.eclipse.org/eclipse/xfsc/cat/fc-service/-/wikis/Installation%20&%20Configuration%20Guide
HashiCorp Vault	https://developer.hashicorp.com/vault/docs/internals/architecture	https://developer.hashicorp.com/vault/docs/install
Keycloak	https://www.keycloak.org/docs/latest/authorization_services/index.html#/overview_architecture	https://www.keycloak.org/guides
EJBCA	https://doc.primekey.com/ejbca/ejbca-introduction/ejbca-architecture/internal-architecture	https://docs.keyfactor.com/ejbca/latest/ejbca-installation
Crossplane	https://docs.google.com/document/d/1whncqdUeU2cATGEJhHzXWC9xdK29Er45NJeoomxebo/edit#heading=h.annq8ww6da48	https://docs.crossplane.io/latest/software/install/
ArgoCD	https://argo-cd.readthedocs.io/en/stable/operator-manual/architecture/	https://argo-cd.readthedocs.io/en/stable/operator-manual/installation/
Cloud-init	https://cloudinit.readthedocs.io/en/latest/	https://cloudinit.readthedocs.io/en/latest/index.html
Gitea	https://docs.gitea.com/category/installation	https://docs.gitea.com/
Apache Kafka	https://kafka.apache.org/documentation/	https://kafka.apache.org/quickstart
Redis	https://redis.io/learn/howtos/quick-start	https://redis.io/docs/latest/operate/oss_and_stack/install/install-redis/
SD (GX-Trust Framework)	N/A	https://code.europa.eu/simpl/simpl-open/development/data1/sdtooling-validation-api-be#installation
XFSC SD Tooling	https://gitlab.eclipse.org/eclipse/xfsc/self-description-tooling	https://gitlab.eclipse.org/eclipse/xfsc/self-description-tooling/sd-creation-wizard-api
XFSC OCM	https://gitlab.eclipse.org/eclipse/xfsc/organisational-credential-manager-w-stack/architecture-documentation	https://gitlab.eclipse.org/eclipse/xfsc/organisational-credential-manager-w-stack/deployment

EDC	https://eclipse-edc.github.io/documentation/	https://eclipse-edc.github.io/documentation/for-adopters/distributions-deployment-operations/
ELK (Elastic, Logstash, Kibana)	https://www.elastic.co/docs	https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html
MinIO	https://min.io/docs/minio/container/operations/concepts/architecture.html	https://min.io/docs/minio/container/operations/installation.html
PostgreSQL	https://www.postgresql.org/docs/current/tutorial-arch.html	https://www.postgresql.org/docs/current/install-binaries.html
Spring Cloud Gateway	https://cloud.spring.io/spring-cloud-gateway/reference/html/#gateway-how-it-works	https://cloud.spring.io/spring-cloud-gateway/reference/html/

Detailed Technical Specifications

This section presents technical implementation details that are particularly relevant for contributing to Simpl-Open and/or implementing it in a data space.

Identification, Authentication & Authorisation

The IAA 2-Tier approach in Simpl-Open is already described in the **Data Spaces Concepts** section of the **Simpl-Open High Level Overview**.

Because of the 2-Tier approach, the components are grouped into Tier 1 and Tier 2.

Tier 1 IAA Components

Tier 1 is meant to be under the control of the governance of the organization that became a Participant of a Dataspace, its components are local to the participant agent and are dedicated to enabling and controlling the access of the organization's end users to the resources/functionalities offered by the Simpl-Open agent and are:

Identification and Authentication

The component responsible for identification and authentication is the **Tier 1 Authentication Provider** realised using an extended version of Keycloak (OpenID Connect Identity Provider) integrated with the **User & Roles** component.

User & Roles

The User and Roles component is used to define roles used by the **Authorisation Tier 1**, manage roles assignment of **Tier 1 Authentication Provider** end users, and assign identity attributes to roles(described in Identity Attributes and *User Roles* sections below)

Authorisation Tier 1

This component manages permissions, determining what actions each end user is authorized to perform on a specific Agent resource. It plays a critical role in maintaining system security by ensuring that only the necessary users have limited access to specific functions, realised through an **API Gateway**, more specifically **Spring Cloud Gateway**, and relies on **Tier1 Authentication Provider** to retrieve roles of authenticated end users to enforce **RBAC (Role Based Access Control) policies** to authorize or deny the access to the requested agent resource.

RBAC policies will be applied to check if the end user has the authorisation to access the requested agent resource/functionality based upon its assigned roles.

Tier 1 Credential

The tier 1 credential consists of an OpenID Connect(OAuth 2.0) **AccessToken** issued by the **Tier 1 Authentication Provider**, in the form of a JWT([rfc7519](#)) that contains standard **claims** extended with the following four custom claims:

Client Roles

The client roles is an array containing the list of roles assigned to the end user through the functionalities of the **User & Roles** component:

```
client-roles : [ "NOTARY", "ONBOARDER_M" ]
```

this will also be included in every tier 1 access token with the claim name "**client-roles**" of the JWT([rfc7519](#))

Participant ID

The participant ID is the unique and immutable ID used to identify the participant in the tier 2 IAA process. It is represented by a **GUID** formatted as shown in the following example:

```
participant_id : "02309243-2f77-456a-a1db-d8e8bb006f74"
```

this will also be included in every tier 1 access token with the claim name "**participant_id**" of the JWT([rfc7519](#))

Note that the participant ID will never change in time.

Credential ID

The credential ID is the unique ID used to identify the current credential participant in the tier 2 IAA process. It is represented by the **HASH** of the **Public key** used to issue the *Participant Credential*(see [ACV Dynamic - BP 03A - Onboard a Participant](#)) formatted as in [Subresource integrity W3C Recommendation](#) as shown in the following example:

```
credential_id : "sha384-Li9vy3DqF8tnTxziaAJuML3ky+er10rcgNR/VqsVpcw+ThHmYcwiB1pbOxEbzJr7"
```

this will also be included in every tier 1 access token with the claim name "**credential_id**" of the JWT([rfc7519](#))

Note that the credential ID will change in time: e.g. when a credential is compromised a new issuance of credentials must occur, requesting a new keypair creation.

Identity Attributes

Participant identity attributes are used to enable the specification of access to a subset of functionalities for a participant. In the context of Tier 2 communication, the presence of Identity Attributes ensures ABAC compliance. Specifically, services provided by dataspace participants to other participants can be protected by one or more Attributes.

A subset of those attributes can be assigned to Tier 1 roles(see Tier 1 User Roles) meaning that every end user belonging to this role owns it, and is represented as in the following example;

```
identity_attributes : [ "DATA_CONSUMER", "DATA_ACCESS_LEVEL1" ]
```

this will also be included in every tier 1 access token with the claim name "**identity_attributes**" of the JWT([rfc7519](#))

Tier 1 User Roles

Tier 1 roles are the core elements on which the RBAC policies are enforced and are also used by the participant governance to assign a subset of Participant Identity Attributes(see Identity Attributes) to its end users.

Here is the updated list of Roles that are used inside Simpl-Open:

Human Readable Role Name	Role Value	Description	Predefined	Participant Type	Assigned Identity Attributes	Id Component
Tier 2 authorization manager	T2IAA_M	In the Dataspace Governance Authority is the one who is in charge of defining and changing the onboarding procedure itself, like setting up the mandatory documents and the rules that will be followed by the onboarding process.	true	Governance Authority		IAA-ONB-FE IAA-ONB-BE
Tier 2 authorization operator	NOTARY	tier 2 authorisation operator, the one who is in charge of taking care of onboarding requests and follow their process. It will ask for further documents, it will comment on the onboarding requests, and reject/approve the requests	true	Governance Authority		IAA-ONB-FE IAA-ONB-BE
Tier 2 setup administration role	ONBOARDER_M	tier 2 setup administrator role, the one who is in charge of finalising the tier 2 setup of an agent installation.	true	All Participant		IAA-U&R-FE IAA-U&R-BE
Tier 2 identity attributes manager	IATTR_M	This role is present only in the Dataspace Governance Authority and its duties are to cover the whole lifecycle of Identity Attributes,	true	Governance Authority		IAA-SAP-FE IAA-SAP-BE

		from the creation and management to the assignment to participants				
Tier 1 user and role manager	T1UAR_M	Tier 1 user and roles manager. In the Dataspace Governance Authority, this role will manage local roles and dataspace identity attributes (defining them and assigning them to participant types + defining their assignability). In any dataspace participant, this role will manage local roles and identity attributes assignment to local roles	true	All Participant		IAA-U&R-FE IAA-U&R-BE
Applicant Representative	APPLICANT	end user responsible for onboarding an applicant dataspace participant who signup the public dataspace onboarding site to manage the onboarding request. His/Her primary scope is to create an onboarding request and react on the Tier 2 authorization operator (NOTARY) interaction to get the onboarding request approved	true	Governance Authority		IAA-ONB-FE IAA-ONB-BE
	Ro-MU-CA	Role defined in XFSC Federated Catalogue: Catalogue Administrator	true	Governance Authority		
	Ro-MU-A	Role defined in XFSC Federated Catalogue: Participant Administrator	true	Providers	DATA_PROVIDER_PUBLISHER APP_PROVIDER_PUBLISHER INFRA_PROVIDER_PUBLISHER	
	Ro-SD-A	Role defined in XFSC Federated Catalogue: Self-Description Administrator	true	Governance Authority		
	Ro-Pa-A	Role defined in XFSC Federated Catalogue: Participant User Administrator	true	Providers	DATA_PROVIDER_PUBLISHER APP_PROVIDER_PUBLISHER INFRA_PROVIDER_PUBLISHER	
Researcher	RESEARCHER	Researcher who is able to access research only datasets	false	Consumer	ACCESS_LEVEL_MEDIUM	
SD Publisher	SD_PUBLISHER	Role defined for the user who is responsible for creating and publishing the self-description on the catalogue	true	Providers	DATA_PROVIDER_PUBLISHER APP_PROVIDER_PUBLISHER INFRA_PROVIDER_PUBLISHER	
SD Consumer	SD_CONSUMER	Tier-1 Role for Consumer	true	Consumer		

Tier 2 IAA Components

Tier 2 is meant to be under the control of the Dataspace Governance Authority and is used by all participant agents to ensure secured and encrypted communications(see *Encryption* and *Guaranteed Authenticity/Integrity* sections below), its components are both centralised (in the Authority Agent) and decentralised (local to all agents)

Centralised

Identity Provider Federation

This component includes functionalities about identity information and Tier 2 credential creation, validation, and management.

Starting from the onboarding process, the Identity Provider will be used for:

- Create the credential: when an applicant participant is onboarded by approving its onboarding request, a Tier 2 credential is created by the identity provider. The participant installs the credential within its own agent.
- Validate the credential: the identity provider verifies the received identity Tier 2 credentials.
- Management: during the lifecycle of a credential, it can be either renewed or revoked by the Dataspace Governance Authority.

Security Attribute Provider Federation

To implement ABAC policies, which are used in agent-to-agent communications, a set of valid and known Identity Attributes are needed and will be assigned to each dataspace participant by the Governance Authority.

The Security Attribute Provider component implements several functionalities:

- Identity Attributes management (create, delete, and modify identity attributes)
- Identity Attributes Participant assignment(both during the Onboarding and after)
- Temporary attestation of the participant's identity attributes in the form of a *signed ephemeral proof*

Decentralised

Tier 2 Authentication Provider

This component is responsible for keeping the Tier 2 Credential received during the onboarding process and implements all Tier 2 Identification and Authentication functionalities such as:

- Keep safely store the participant agent Tier 2 Credential and its keypair
- Check and Validate any Tier 2 credentials coming from other participant agents during the mTLS Authentication against the **Identity Provider Federation**.
- Check and Validate the ephemeral proof received from other participant agents after the successful mTLS Authentication process.
- Check and Validate the Tier 1 credential forwarded by other participant agents against the ephemeral proof(that contains also the caller **Tier 1 Authentication Provider** public key)
- Request ephemeral proof to the **Security Attribute Provider Federation** to be used in secured communications with other participant agents

Authorisation Tier 2

This component is realised through an **API Gateway**, more specifically **Spring Cloud Gateway**, and relies on the **Tier 2 Authentication Provider** to check Tier 2 credentials and ephemeral proof received during the mTLS Authentication process to enforce **ABAC(Attribute Based Access Control) policies** to authorize or deny access to the requested agent resource.

ABAC policies will be enforced in any agent-to-agent communication, by verifying whether the requestor's attributes are permitted to access the requested resource, and if needed the enforcement of ABAC policies can be done also in both Tier 1 and Tier 2 credentials(to check if the identity attribute is also present in the Tier 1 credential used by the end user of the caller participant agent)

Tier 2 Credential

The Tier 2 credential has the form of an X509 Certificate and is issued by a Certificate Authority embedded in the **Identity Provider Federation**.

Identity Attributes

Identity attributes are the most powerful and versatile tool at the disposal of the Dataspace Governance Authority to "design" the governance and the rules in the interactions between Dataspace participants. Some attributes are built in SIMPL Open(**Predefined = true**) and cannot be modified/removed.

Two important properties can be used in the definition of Identity attributes:

Assignable: if **true** means that any governance of a Participant that receives this identity attribute can assign it to any Tier 1 roles to then give it to its end users

IsRight: if **true** means that the identity attribute should be considered as a special centralised right.

Here is the updated list of Identity Attributes that are used inside Simpl-Open:

Human Readable Attribute Name	Identity Attribute Value	Participant Type	Description	Predefined	Assignable	IsRight	Id Component
Consumer	CONSUMER	Consumer	Identity attribute used for tagging an end user able to act as a user of a data consumer participant	true	false	false	

Data Provider	DATA_PROVIDER	Data Provider	Identity attribute used to tag the data provider	true	false	false	
Application Provider	APP_PROVIDER	Application Provider	Identity attribute used to tag the application provider	true	false	false	
Infrastructure Provider	INFRA_PROVIDER	Infrastructure Provider	Identity attribute used to tag the infrastructure provider	true	false	false	
Data Provider Publisher	DATA_PROVIDER_PUBLISHER	Data Provider	Identity attribute needed for publishing Data Catalogue	true	true	true	
Application Provider Publisher	APP_PROVIDER_PUBLISHER	Application Provider	Identity attribute needed for publishing Application Catalogue	true	true	true	
Infrastructure Provider Publisher	INFRA_PROVIDER_PUBLISHER	Infrastructure Provider	Identity attribute needed for publishing Infrastructure	true	true	true	
Basic Access Level	ACCESS_LEVEL_BASIC	Consumer	Basic Access Level	false	true	true	
Medium Access Level	ACCESS_LEVEL_MEDIUM	Consumer	Medium Access Level	false	true	true	
Full Access Level	ACCESS_LEVEL_FULL	Consumer	Full Access Level	false	true	true	

Encryption

In mTLS (mutual Transport Layer Security) communication, **encryption of in-transit data** ensures that the information exchanged between a client and a server is protected from interception or tampering. This encryption is achieved through the following process:

1. **TLS Handshake:** Both the client and server initiate a TLS handshake, during which they exchange public keys and agree on encryption algorithms.
2. **Mutual Authentication:** Unlike regular TLS, in mTLS both the client and server authenticate each other by exchanging digital certificates, confirming the identity of both parties.
3. **Symmetric Encryption:** After authentication, a symmetric encryption key is established and used to encrypt all subsequent data transmitted between the client and server.

Through this process, **data in transit is securely encrypted**, preventing unauthorized access or modification, while ensuring that both the client and server are trusted entities.

Guaranteed Authenticity / Integrity

Supports the measures in place to ensure end-to-end data integrity, such that Simpl-Open agents can validate the authenticity of the delivered information.

This capability is achieved by implementing mTLS communication between agents, ensuring that communication can be established only between trusted and known participants from the Authority.

The Governance Authority during the onboarding processes creates unique Identity Credentials for each participant of the Dataspace. Then the participant uses the credential during the mTLS communication.

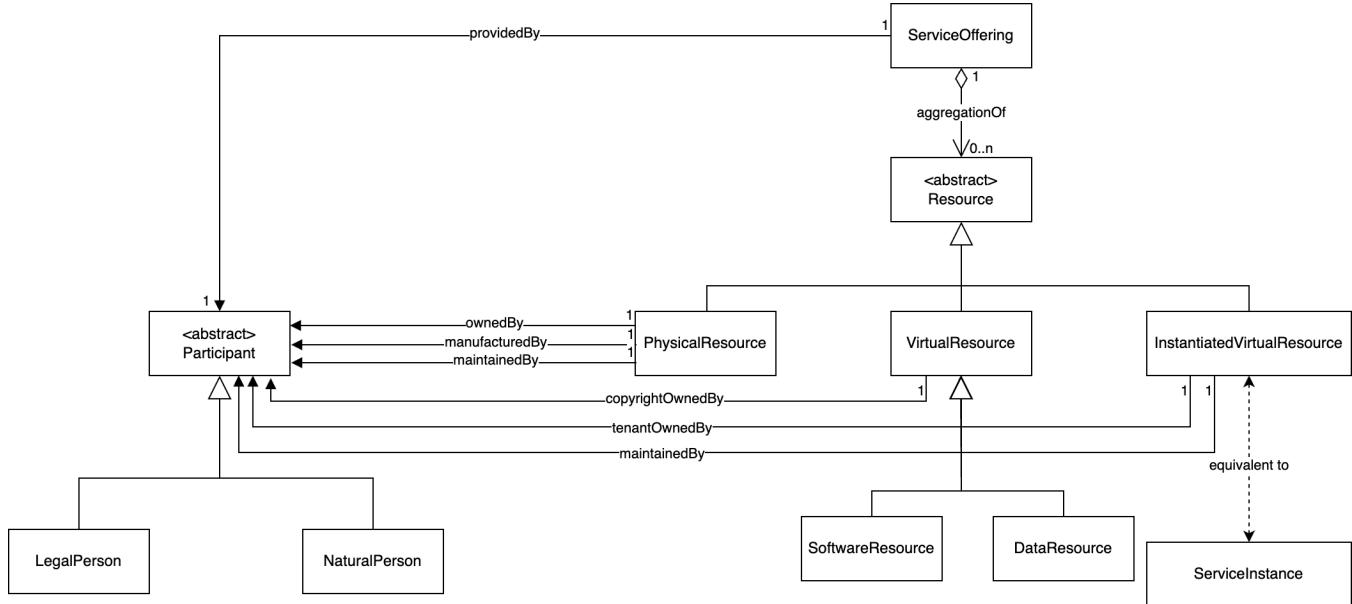
Self Descriptions

The metadata will be described as self-descriptions. These are described in this section.

In the sub-section Self-Description Tooling the tools to create self-descriptions are introduced and the flow of the different steps to be considered are visualized. The SD Schema Creator enables customized schemas for each data space. In Schema Definition Properties we enlist the proposed attributes any SIMPL data space should utilise. The Validation of Syntax and schema can be looked up in SD Tooling Syntax Validation & Schema Validation.

The structure of Self Descriptions should be based on the [GAIA-X Trustframework](#). There are already GAIA-X powered data spaces providing such a SD. This way the created SD can be easily reused and be enhanced by the special requirements of each sectoral data space.

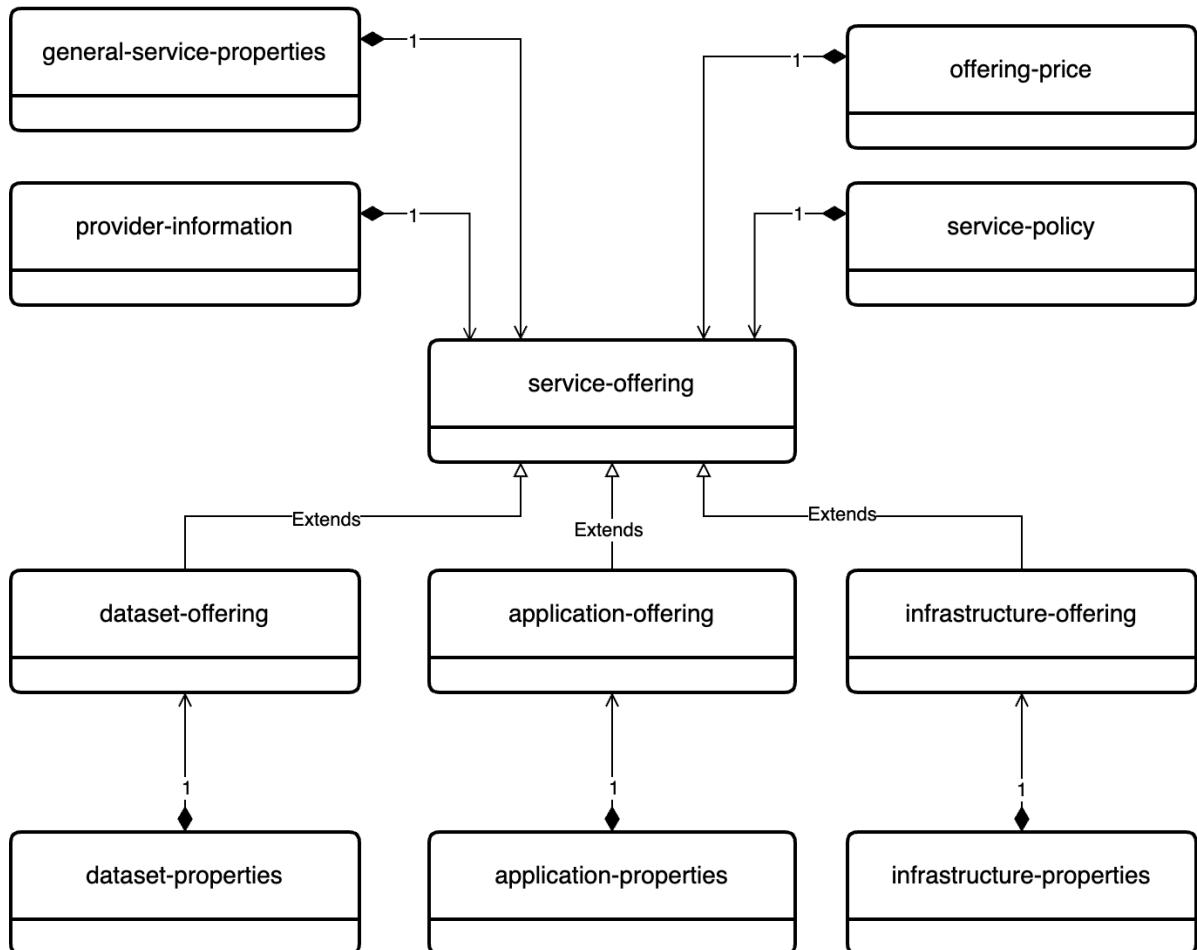
Base Entities and their relationship due to GAIA-X Trustframework



The description of the attributes for each Entity is described in the [GX-Trustframework](#) document and the respective chapters.

Proposed enhancements based on SIMPL requirements related to ServiceOffering from GAIA-X Trustframework

Service Offering Structure [Proposal]



Mapping of SIMPL required attributes to SD Entities and attributes

i Suggestions for enhancing the SD with Data Types and Constraints:

In order to allow ServiceProvider reusing their Self Descriptions according to GAIA-X Trustframework, we suggest mapping the attributes to mandatory attributes defined by GAIA-X Trustframework and enhance the SD with SIMPL entities for additional mandatory attributes due to SIMPL needs.

Data Offering:

Simpl Attribute	Entity	Attribute	Cardinality	Mandatory / Recommended	Data Type	Constraint	Comment
Unique identifier	service-offering	id	1	Mandatory	xsd:string		The id of the ServiceOffering, usually referring to a DID. Set automatically.
Name	service-offering	name	1	Mandatory	xsd:string	sh:maxLength 255	A human readable name of the service offering
Description	service-offering	description	1	Mandatory	xsd:string	sh:maxLength 1000	a short description of the service offering

Location of the dataset (e.g. URL, handle)	service-offering	serviceAccessPoint	1	Mandatory	xsd:anyURI	sh:pattern "[http(s)?:\/\/(www\.)?a-zA-Z0-9@:_\+\-#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:_\+\-#=&/=]*)"	a list of Service Access Point which can be an endpoint as a mean to access and interact with the resource
Keywords	service-offering	keywords	0..16	Recommended	xsd:string	sh:maxLength 50	list of keywords
Language (of the metadata, like the title, description)	service-offering	inLanguage	1	Mandatory	xsd:string	sh:languageIn ("bg" "hr" "cs" "da" "nl" "en" "et" "fi" "fr" "de" "el" "hu" "ga" "it" "lv" "lt" "mt" "pl" "pt" "ro" "sk" "sl" "es" "sv")	The language of the content or performance or used in an action. Please use one of the language codes from the IETF BCP 47 standard . See also availableLanguage .
Version					xsd:string		The version of the self-description. Technical property, set automatically.
Creation date					xsd:dateTimeStamp		The first onboarding date. Technical property, set automatically.
Last update date					xsd:dateTimeStamp		The last update date. Technical property, set automatically.
SD Schema					xsd:string		Reference to the used Schema ID (and version). Technical property, set automatically.
Data Provider	provider-information	providedBy	1	Mandatory	xsd:string	sh:maxLength 255	Reference to Participant SD. To be Set automatically.
Contact point (who to contact in case of questions/issues)	provider-information	contact	1	Mandatory	xsd:string	sh:pattern "^\w+@\[(\w+]+\.\.)+[\w-]{2,4}\$"	email address of the contact point
License	offering-price	license	1..n	Mandatory	xsd:anyURI	sh:pattern "[http(s)?:\/\/(www\.)?a-zA-Z0-9@:_\+\-#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:_\+\-#=&/=]*)" sh:maxLength 255	A list of SPDX identifiers or URL to document
Price Type				Recommended	xsd:string	sh:in("free" "commercial")	Link to price in the future.
Price (free, under cost)	offering-price	price	1	Mandatory	xsd:decimal	sh:minInclusive 0	
Currency				Mandatory	xsd:string	sh:in("BGN" "EUR" "CZK" "DKK" "HUF" "PLN" "RON" "SEK")	
Access policy (to define who can access the dataset)	service-policy	policy	0..n	Recommended	xsd:string	sh:pattern "[\:\,\{\}\[\]\ \\\" \'.*\?\\\" \'.*\?' \[-\w\.]+"	a list of policy expressed using a DSL (e.g., Rego or ODRL) (access, control, throttling, usage, retention, ...)
Usage policy (to define how a dataset can be used)	service-policy	policy	0..n	Recommended	xsd:string	sh:pattern "[\:\,\{\}\[\]\ \\\" \'.*\?\\\" \'.*\?' \[-\w\.]+"	a list of policy expressed using a DSL (e.g., Rego or ODRL) (access, control, throttling, usage, retention, ...)
Compliance: Indicates compliance with relevant data protection regulations and standards.	service-policy	dataProtectionRegime	0..n	Recommended	xsd:string	sh:pattern "[\:\,\{\}\[\]\ \\\" \'.*\?\\\" \'.*\?' \[-\w\.]+"	
Provenance	dataset-properties	producedBy	1	Recommended	xsd:anyURI	sh:pattern "[http(s)?:\/\/\/"	a resolvable link to the participant self-

					(www\.)?a-zA-Z0-9@:%_\+\-#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+\-#=?&/=]*")	description legally enabling the data usage	
Format under which the data is distributed (e.g. csv, xml, ...)	dataset-properties		1	Mandatory	xsd:string		
Schema of the dataset, depends on the type of data for JSON it would be JSON Schema Description that states what fields the data has and the types.	dataset-properties	openAPI	0..n	Recommended	xsd:anyURI	sh:pattern "[((http(s)?):\/\/(www\.)?a-zA-Z0-9@:%_\+\-#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+\-#=?&/=]*")"	
Additional Information about the dataset					xsd:anyURI	sh:pattern "[((http(s)?):\/\/(www\.)?a-zA-Z0-9@:%_\+\-#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+\-#=?&/=]*")"	
Related datasets	dataset-properties		0..n	Recommended	xsd:string		
Target users	dataset-properties		0..n	Recommended	xsd:string		
Data Quality (to include metrics such as completeness, accuracy, timeliness and other)	dataset-properties		0..n	Recommended	xsd:string		
Encryption: Describes the encryption algorithms and keys used to secure the data.	dataset-properties		0..1	Recommended	xsd:string		
Anonymization/pseudonymization: Indicates whether sensitive information has been anonymized or pseudonymized to protect privacy.	dataset-properties		0..1	Recommended	xsd:string		
Contract template Name	TBD	termsAndConditions	1..n		xsd:string	Name of the template	
Contract template Hash	TBD	termsAndConditions			xsd:string	hash value of the linked document	
Contract template (human-readable) newly added	TBD	termsAndConditions	1..n		xsd:anyURI	sh:pattern "[((http(s)?):\/\/(www\.)?a-zA-Z0-9@:%_\+\-#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+\-#=?&/=]*")"	a resolvable link to the Terms and Conditions applying to that service.
Contract template Hash Algorithm	TBD	termsAndConditions			xsd:string		Hash Algorithm that has been used to create the hash value of the linked document
SLA template Name	TBD	termsAndConditions	1..n		xsd:string		Name of the template
SLA template Hash	TBD	termsAndConditions			xsd:string		hash value of the linked document
SLA template (human-readable) newly added	TBD	termsAndConditions	1..n		xsd:anyURI	sh:pattern "[((http(s)?):\/\/(www\.)?a-zA-Z0-9@:%_\+\-#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+\-#=?&/=]*")"	a resolvable link to the Terms and Conditions applying to that service.
SLA template Hash Algorithm	TBD	termsAndConditions			xsd:string		Hash Algorithm that has been used to create the hash value of the linked document
billing template Name	TBD	termsAndConditions	1..n		xsd:string		Name of the template
billing template Hash	TBD	termsAndConditions			xsd:string		hash value of the linked document
billing template (human-readable) newly added	TBD	termsAndConditions	1..n		xsd:anyURI	sh:pattern "[((http(s)?):\/\/(www\.)?a-zA-Z0-9@:%_\+\-#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+\-#=?&/=]*")"	a resolvable link to the billing template applying to that service.

						Z0-9@:%_\+.\~#?& /=/]*")"	
billing template Hash Algorithm	TBD	termsAndConditions			xsd:string		Hash Algorithm that has been used to create the hash value of the linked document

Infrastructure Offering:

Simple Attribute	Entity	Attribute	Cardinality	Mandatory / Recommended	Data Type	Constraint	Comment
Resource Type	infrastructure-properties		1	Mandatory	xsd:string	sh:in("vm" "container" "block_storage" "object_storage" "relational_db" "document_db")	
Region and availability zone	infrastructure-properties		1..n	Mandatory	xsd:string	sh:in("eu-west-1" "eu-west-2" "eu-west-3" "eu-central-1" "eu-north-1" "eu-south-1" "eu-south-2")	
Size and capacity	infrastructure-properties		0..1	Recommended	xsd:string	sh:pattern "\d+(\.\d+)?s?(B KB MB GB TB PB EB ZB YB)"	
Operating system and image	infrastructure-properties		0..1	Mandatory	xsd:string		
Network configuration	infrastructure-properties		0..1	Recommended	xsd:string		
Security settings (access control, security groups/firewalls, encryption)	infrastructure-properties		0..1	Mandatory	xsd:string		
Instance type	infrastructure-properties		0..1	Mandatory	xsd:string		
Storage type	infrastructure-properties		0..1	Mandatory	xsd:string		
Backup and redundancy	infrastructure-properties		0..1	Recommended	xsd:string	sh:in("full-backup" "incremental-backup" "differential-backup")	
Scalability options	infrastructure-properties		0..1	Recommended	xsd:string	sh:in("dynamic-scaling" "scheduled-scaling", "sharding")	
Monitoring and logging	infrastructure-properties		0..1	Recommended	xsd:string		
Tags and metadata	infrastructure-properties	keywords	0..16	Recommended	xsd:string	sh:maxLength 50	
External URL	infrastructure-properties		1	Mandatory	xsd:string	sh:maxLength 255	
Deployment script ID	infrastructure-properties		0..1	Mandatory	xsd:string		

termsAndCondition structure (defined by GAIA-X Trustframework)

Attribute	Cardinality	DataType	Comment
URL	1	xsd:string	a resolvable link to document
hash	1	xsd:string	SHA256 of the above document

dataAccountExport structure (defined by GAIA-X Trustframework)

The purpose is to enable the participant ordering the service to assess the feasibility to export its personal and non-personal data out of the service. This export shall cover account data e.g., account holder's billing information, information on the PII held - but also data provided previously to the service by the user.

Attribute	Cardinality	DataType	Comment
requestType	1	xsd:string	the mean to request data retrieval: API, email, webform, unregisteredLetter, registeredLetter, supportCenter
accessType	1	xsd:string	type of data support: digital, physical
formatType	1	xsd:string	type of Media Types (formerly known as MIME types) as defined by the IANA .

Quality Rules

For the MVP only *mandatory quality rules* are supported. A Quality Score can only be calculated for recommended quality rules thus this will also not be supported.

Mandatory quality rules are always enforced during the creation of a Self-Description (SD) for an offering, to ensure the data quality of the SD. A resource provider is not be able to publish an SD that is not complying with the mandatory quality rules.

Quality Rule Formalization

Quality rules are defined in the schema of the self-description (which are semantic RDF Graphs) and allow to express data types, constraints and conditions on those RDF Graphs. Thus, we want to use [SHACL](#) (Shape Constraint Language) Constraints as the formal notation to express quality rules.

The quality rules that can be defined for an SD property can be base on the data type and/or on a SHACL constraint. Example for Constraints:

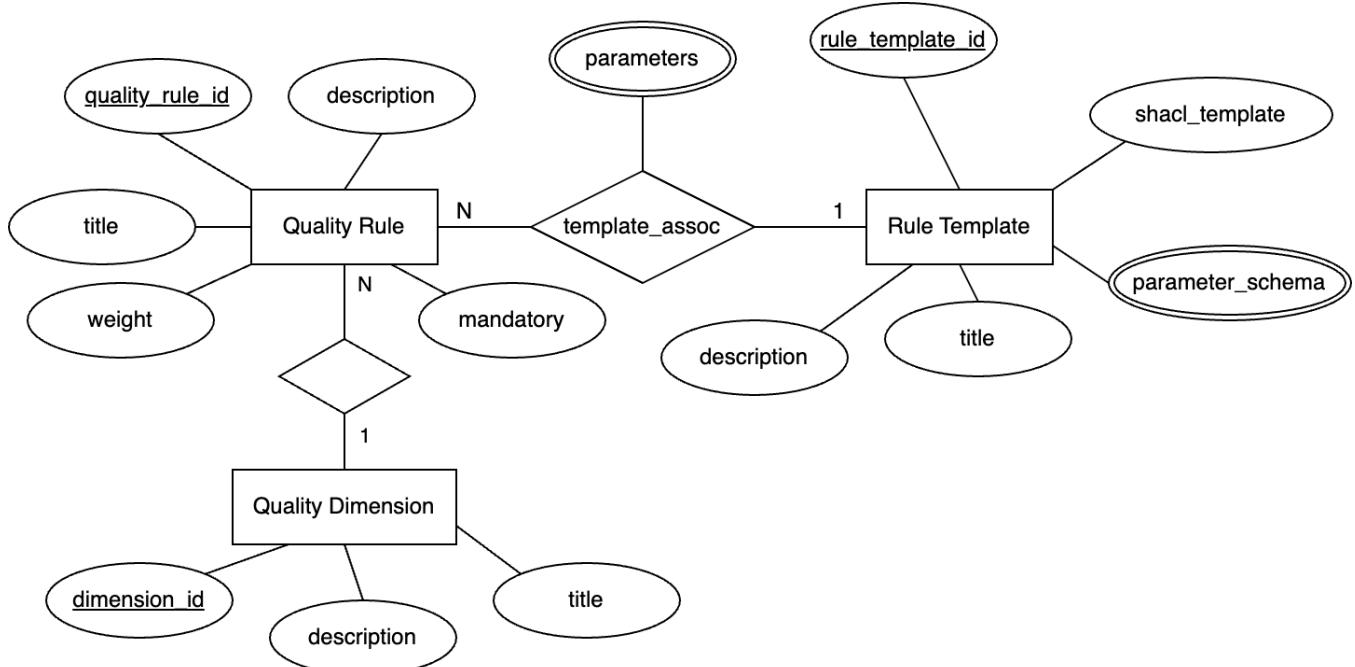
- Minimum or maximum length of a string value
- Value Ranges for Numbers
- Non-Negative Numbers
- Regular Expressions (Patterns)
- List of allowed values
- Constraints based on other properties

Data Model (initial)

To define the quality rules, we have three basic entities:

- Quality Rule: The (mandatory) quality rule. It is uniquely defined by an id and contains a textual description of the rule in clear text;
- Rule Template: The template for the formal definition of the rule. It contains besides the ID a field with SHACL template that are parameterised. The number of parameter and their type is defined in a parameter_schema, i.e., JSON Blob with the parameter and data types;
- Quality Dimension: The quality dimension used to group the quality rule for instance FAIR as example.

Each Quality Rule has exactly one Dimension and one Rule Template associated. The template_assoc also contains the concrete parameterization for the rule template.



Score Calculation

The score is calculated by dimension.

```

\delta_{r,s} = \begin{cases} 1 & \text{if the quality rule } r \text{ is fulfilled for the self-description } s \\ 0 & \text{else} \end{cases}
\text{score}(s, d) = \frac{\sum_{r \in \sim R_d} \delta_{r,s} * w_r}{\sum_{r \in \sim R_d} w_r} * 100
\text{is valid} \Leftrightarrow \forall d \in D: \text{score}(s, d) \geq \text{min\_score}_d

```

$$\delta_{r,s} = \begin{cases} 1 & \text{if the quality rule } r \text{ is fulfilled for the self-description } s \\ 0 & \text{else} \end{cases}$$

$$\text{score}(s, d) = \frac{\sum_{r \text{ in } R_d} \delta_{r,s} * w_r}{\sum_{r \text{ in } R_d} w_r} * 100$$

$$s \text{ is valid} \Leftrightarrow \forall d \in D : \text{score}(s, d) \geq \text{min_score}_d$$

The kronecker-delta is 1 if the quality rule r is fulfilled for the self-description s, else 0.

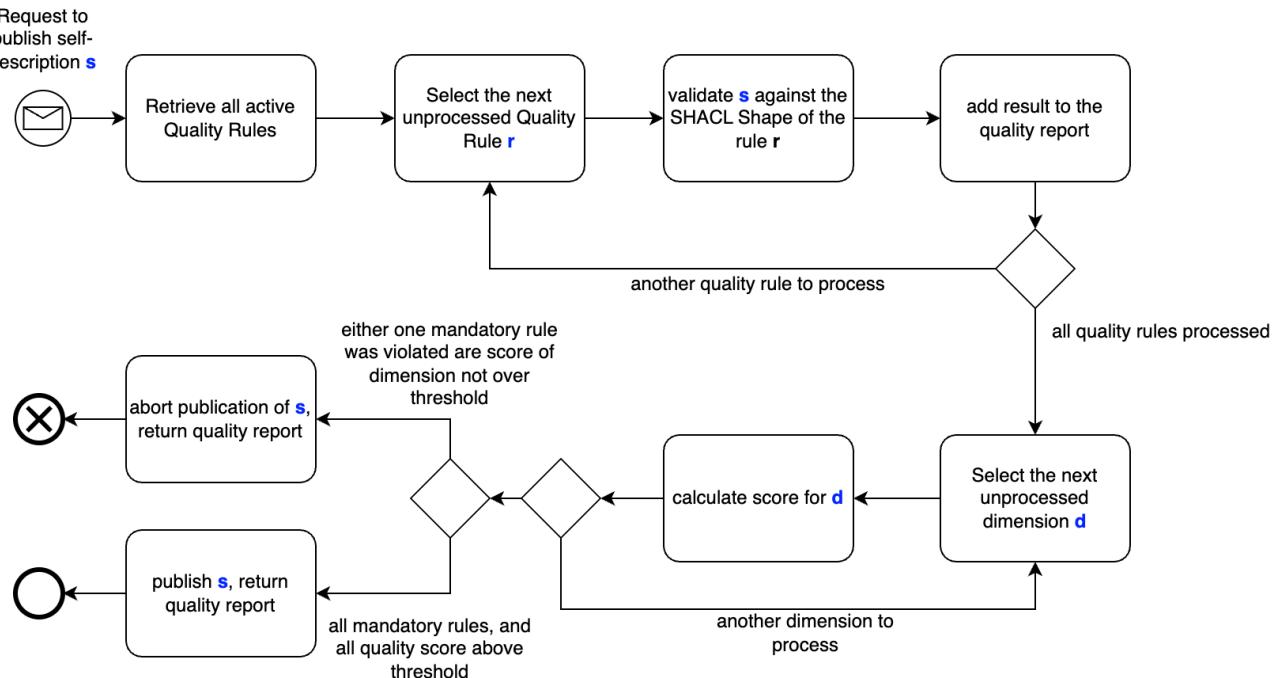
The score for a self-description s and quality dimension d is calculated by the sum over all the Quality rules r for the Dimension d (R_d) multiplied by their weight w. This is than normalized by the sum of all weights for the dimension. Because we want to have a value between 0-100 and not between 0-1 it is multiplied by 100.

A self-description is valid exactly if, for all quality dimension the score is greater than the specified threshold (min_score).

Calculation Process

The calculation of the score (and the validation of the rules) is done during the publication of the self-description **s** in the query mapper. First, we get all the quality rules that are active from the database (with the SHACL template associated). We loop over all the rules and validate them against the self-description **s**. The results are added to the quality report for **s**. After all the rules are processed, we iterate over all the quality dimensions **d**. For each dimension the score is calculated.

Next, we check if all the mandatory rules are fulfilled and if the score for each dimension is above the threshold defined. If this is not the case, we abort the publication and return the quality report to the provider. Else, the publication continues and the quality report is returned to the provider.



Self-Description Tooling

The self-description tooling consist of four different components that are all in their respective repositories:

1. SD Schema Creator: [SD Schemas](#)

This component creates the schemas that describe the form and content of the self-description. It is used by the Governance Authority to set the standard on the Self-Description. Technically it is done by a set of configuration files in the form of YAML-Documents. Those files are verified and

transformed into an ontology and SHACL Constraints that are used by the other components to create the wizards. The component is written in Python and we need to adjust at least the YAML Configuration for the POC.

2. SD Creation Wizard API: [SD Creation Wizard API](#)

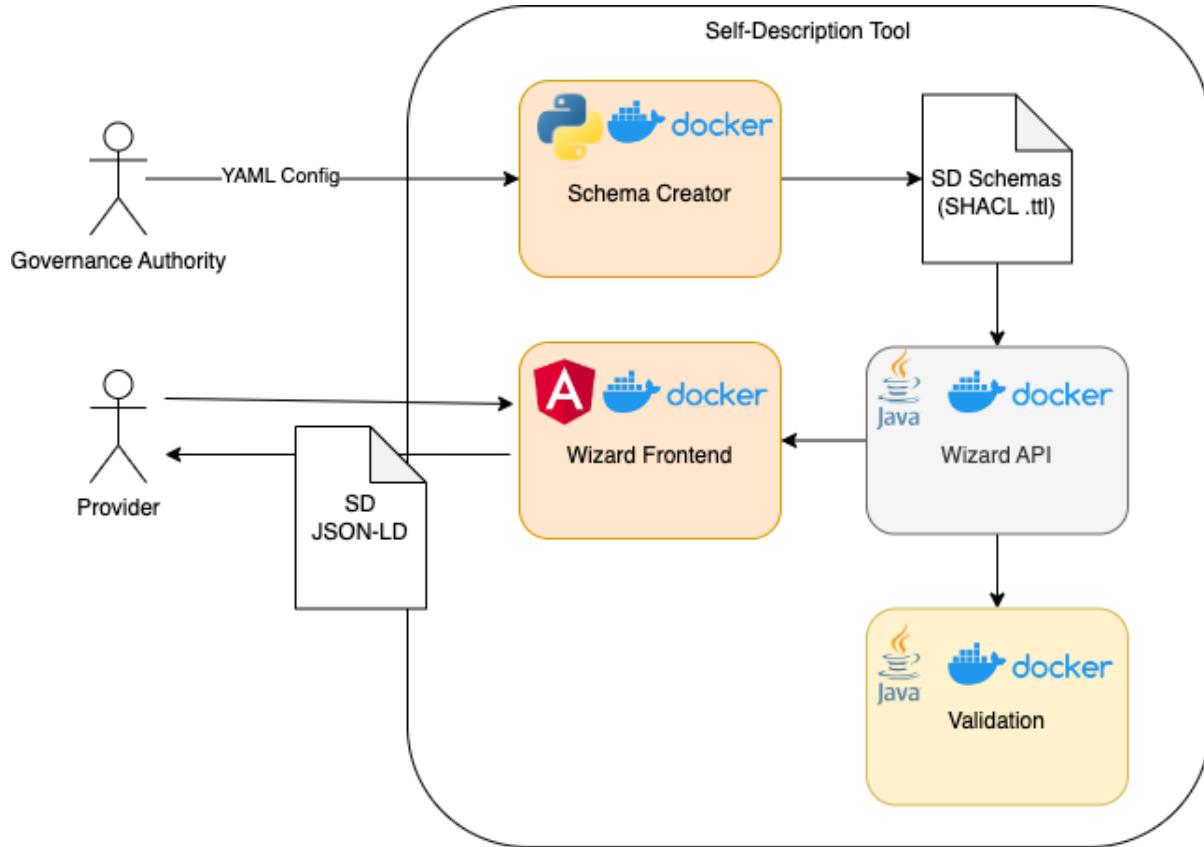
The main API project. Transform the SHACL-shapes from the SD Schema Creator into JSON forms that are used by the frontend to allow the provider to write new Self-Descriptions. The project is written in Java, currently we do not foresee changes needed for the POC.

3. SD Creation Wizard Frontend: [SD Creation Wizard Frontend](#)

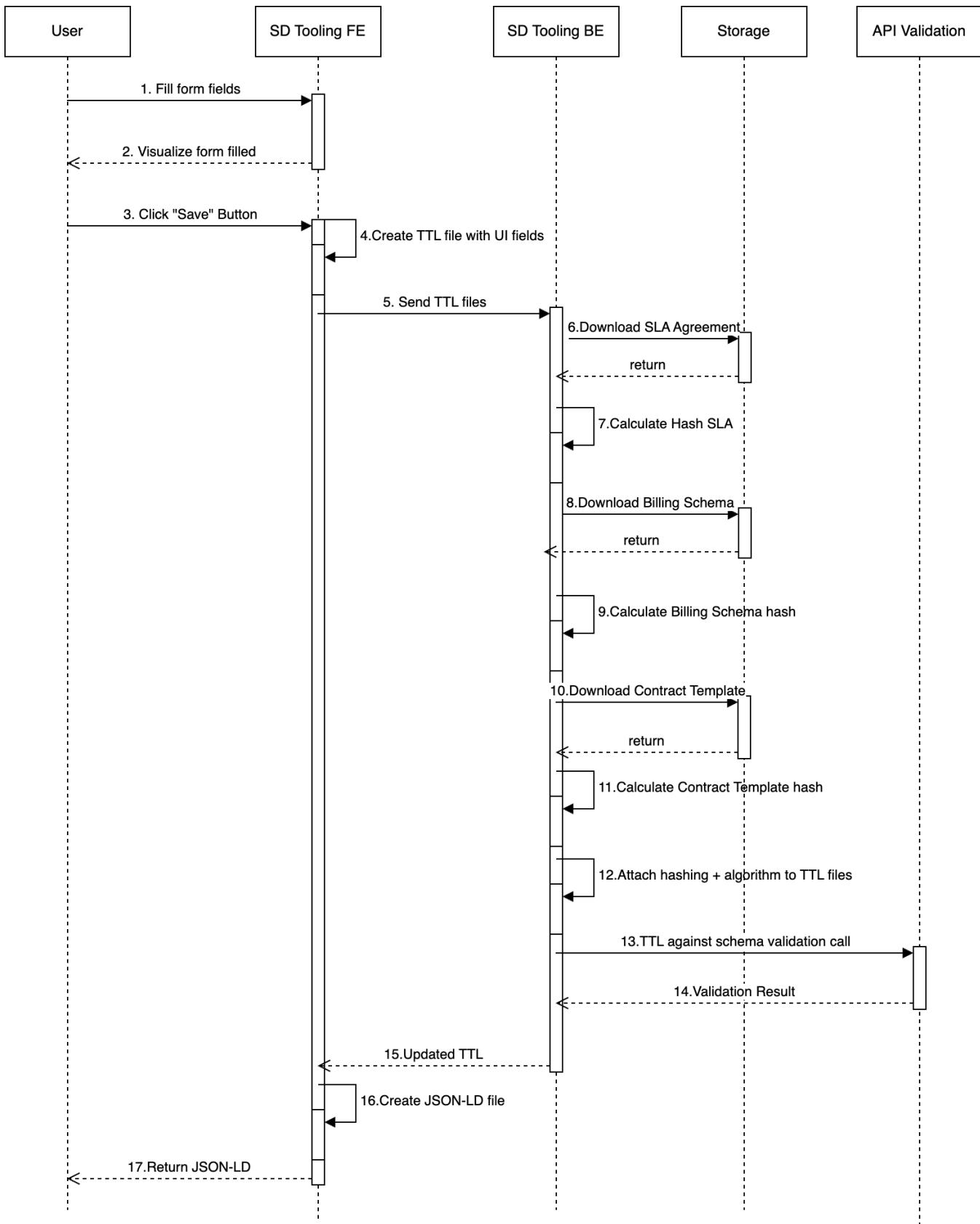
Frontend with the forms for the provider to create Self-Descriptions. Written in Angular and NodeJS. The result is a SD in the form of a JSON-LD document that can be uploaded to the catalogue.

4. SD Validation API: [SD Validation API](#)

Validation of the Self-Description against SHACL files. Might be used for the Quality Rule Validation. Currently outside of the POC. Written in Java.



[Flow Diagram](#)



1. Provider User select offering and fill form;
2. SD Tooling FE do first syntax validation based on the schema and return feedback to user;
3. User clicks on Save button;
4. SD Tooling FE create a TTL file with the values inserted on the form;
5. SD Tooling FE invoke an API to transmit TTL file created;
6. SD Tooling BE downloads SLA Agreements PDF file from the Storage. (The file is static and pre-loaded on the storage);

7. SD Tooling BE calculate hash with SHA-256 algorithm;
8. SD Tooling BE downloads Billing Schema PDF file from the Storage. (The file is static and pre-loaded on the storage);
9. SD Tooling BE calculate hash with SHA-256 algorithm;
10. SD Tooling BE downloads Contract Template PDF file from the Storage. (The file is static and pre-loaded on the storage);
11. SD Tooling BE calculate hash with SHA-256 algorithm;
12. SD Tooling BE attach hashing of three files and algorithm inside the TTL file;
13. SD Tooling BE invoke an API to Validation API tool for validation of graph TTL file against the schema;
14. SD Tooling BE receives Validation Result;
15. SD Tooling BE returns Updated TLL with the new fields (hash and algorithm);
16. SD Tooling FE generates JSON-LD file;
17. User can download JSON-LD file.

SD Schema Creator

Background

Self-Description in the context of DATA/APP are documents that describe the service offering (either Data, Application, or Infrastructure). The Schema of the Self-Description defines the format of the Self-Description, i.e. it is a description about what are the fields for the self-description, their data types and if they are mandatory or not.

Component Self-Description Schema Creator:

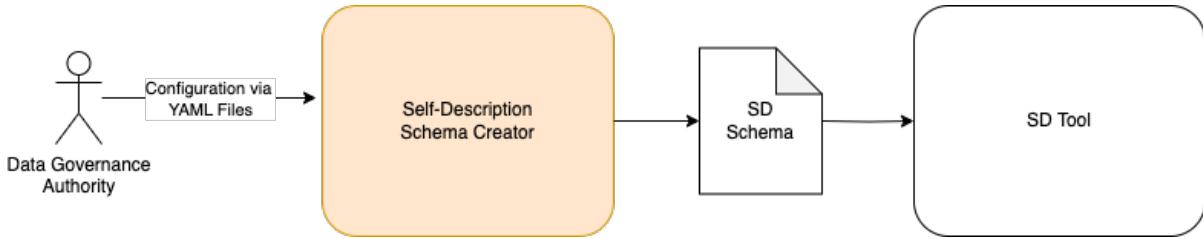
The Schema-Framework is a component that is able to generate the self-description schemas from configuration files. The idea is that from a simple configuration the schemas are generated and later used by the provider to write the self-description.

It should include validation of the schema files (syntax and semantic)

Basis of the implementation is the repository from GaiaX Context [sd-schemas](#)

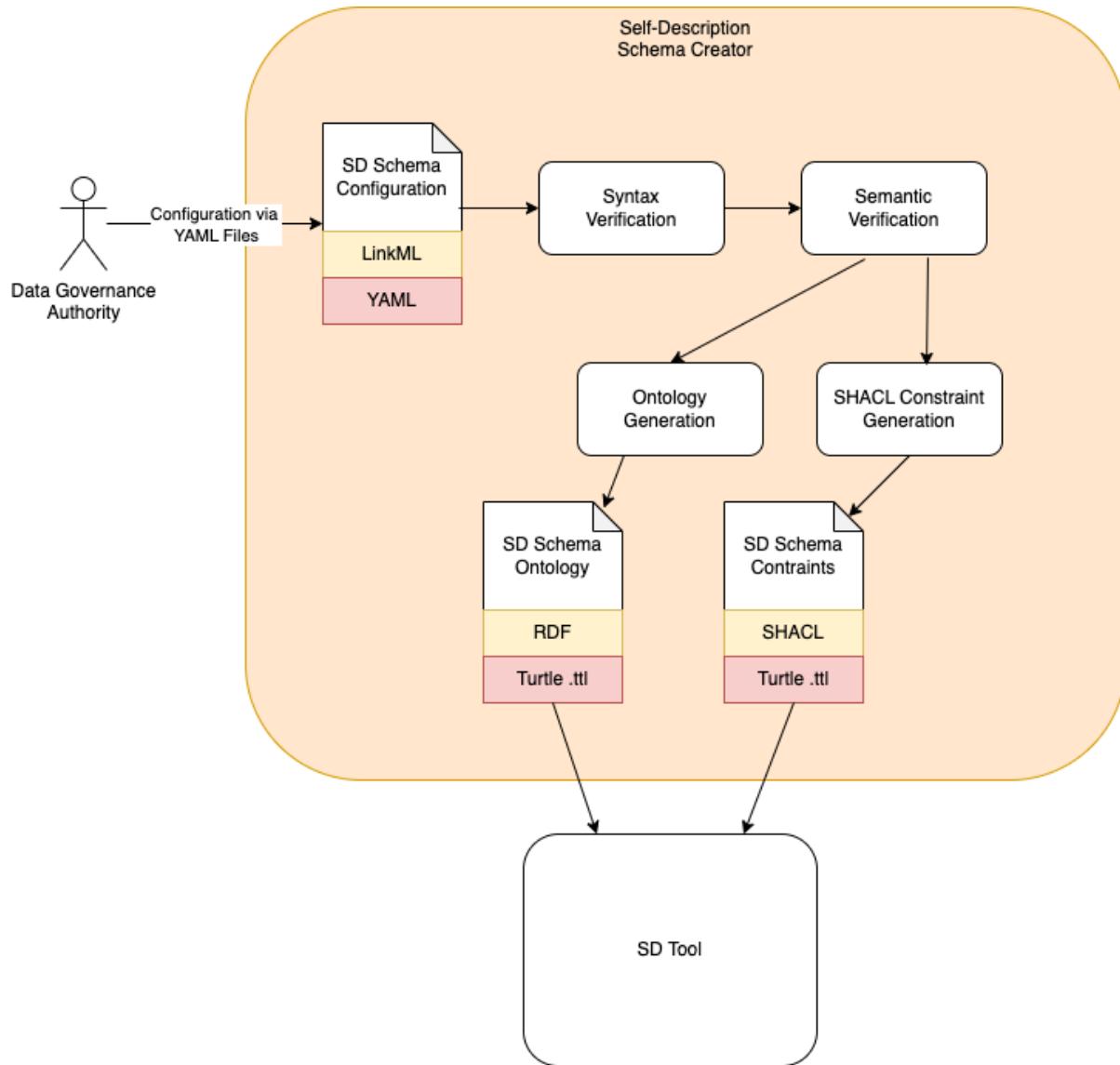
Context View

The main actor on the SD Schema Creator is the Data Governance Authority. They can configure the schema by changing the yaml files that define how the schemas for the different services should look like.



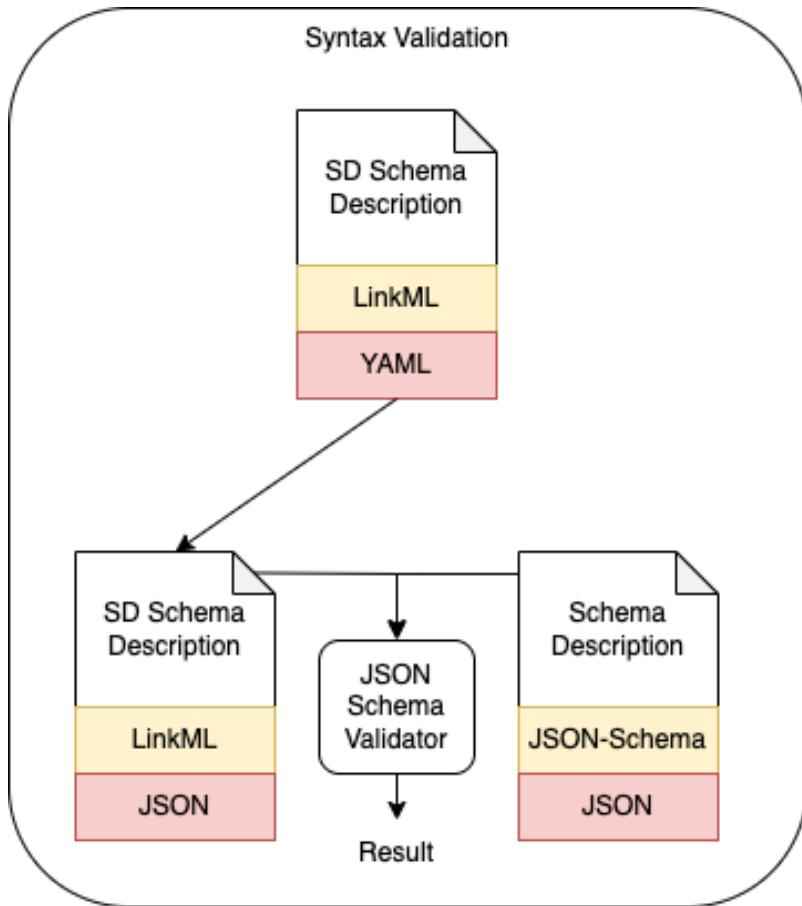
Component View

The input of the system is the SD Schema Configuration, the file uses the LinkML data model and is serialized as a YAML document. After the configuration is changed the process is triggered that first checks the syntax and the semantic. After the validation the configuration files are transformed into two different files that describe the semantic. One is an ontology, i.e. a formal representation of the knowledge which is used as a vocabulary for the SD Tool. The other are constraints in the form of SHACL-Shapes that are used as a template to build the forms in the SD Tool. Both semantic files are serialized as Turtle-files.



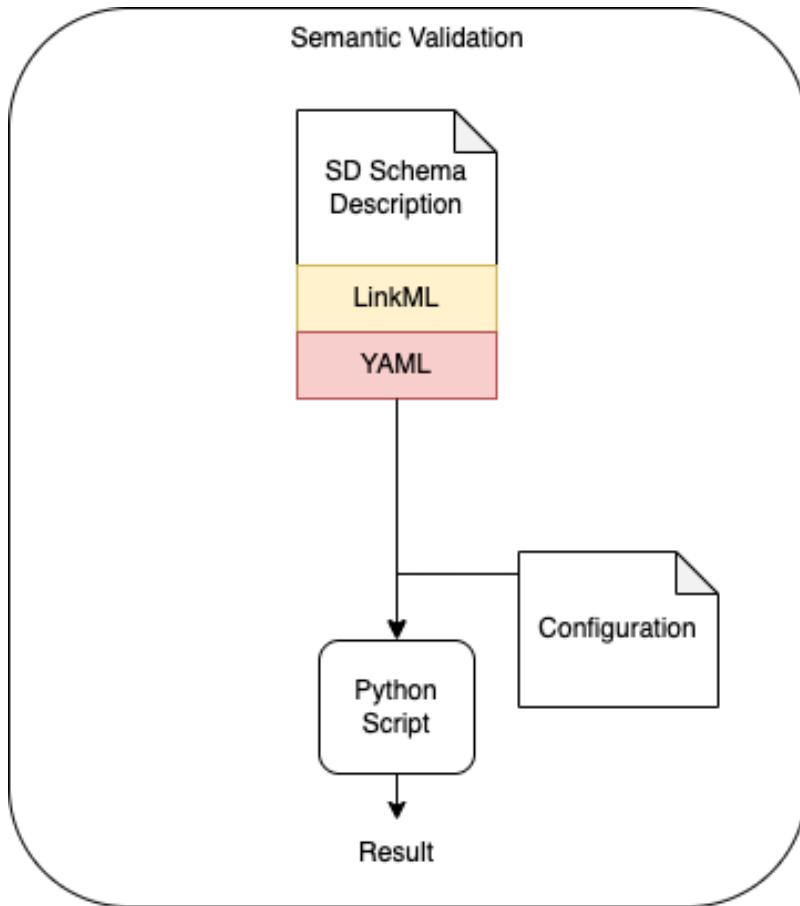
Syntax Validation

For YAML files there exist currently no standard for schema validation. To this end, we transform the SD Schema Description into a JSON Serialisation and use a JSON-Schema Description for the syntax validation. This JSON-Schema is written by the Data Governance Authority.



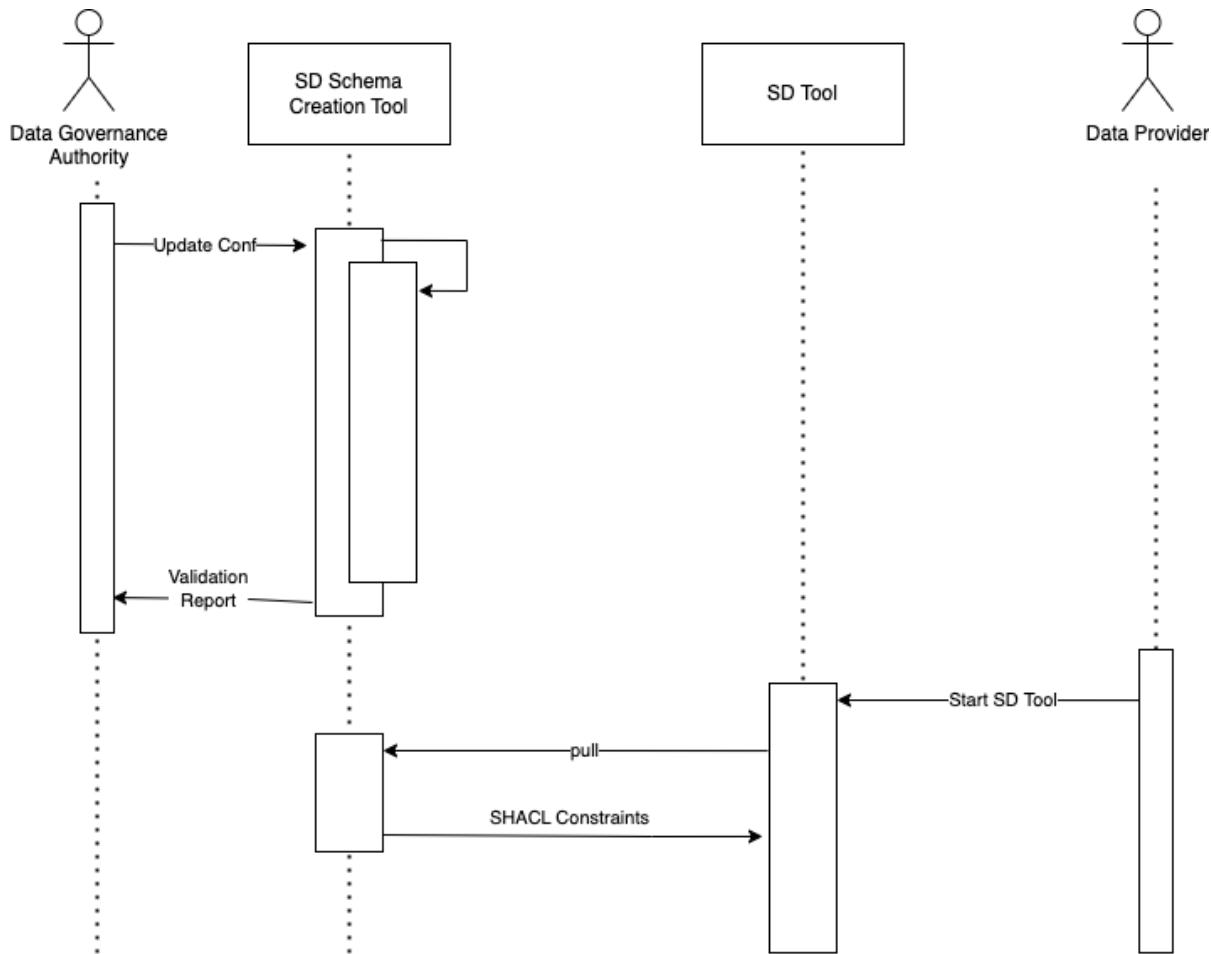
Semantic Validation

The Semantic Validation uses a python script which reads some configuration and guideline (for instance which fields are mandatory in the schema).



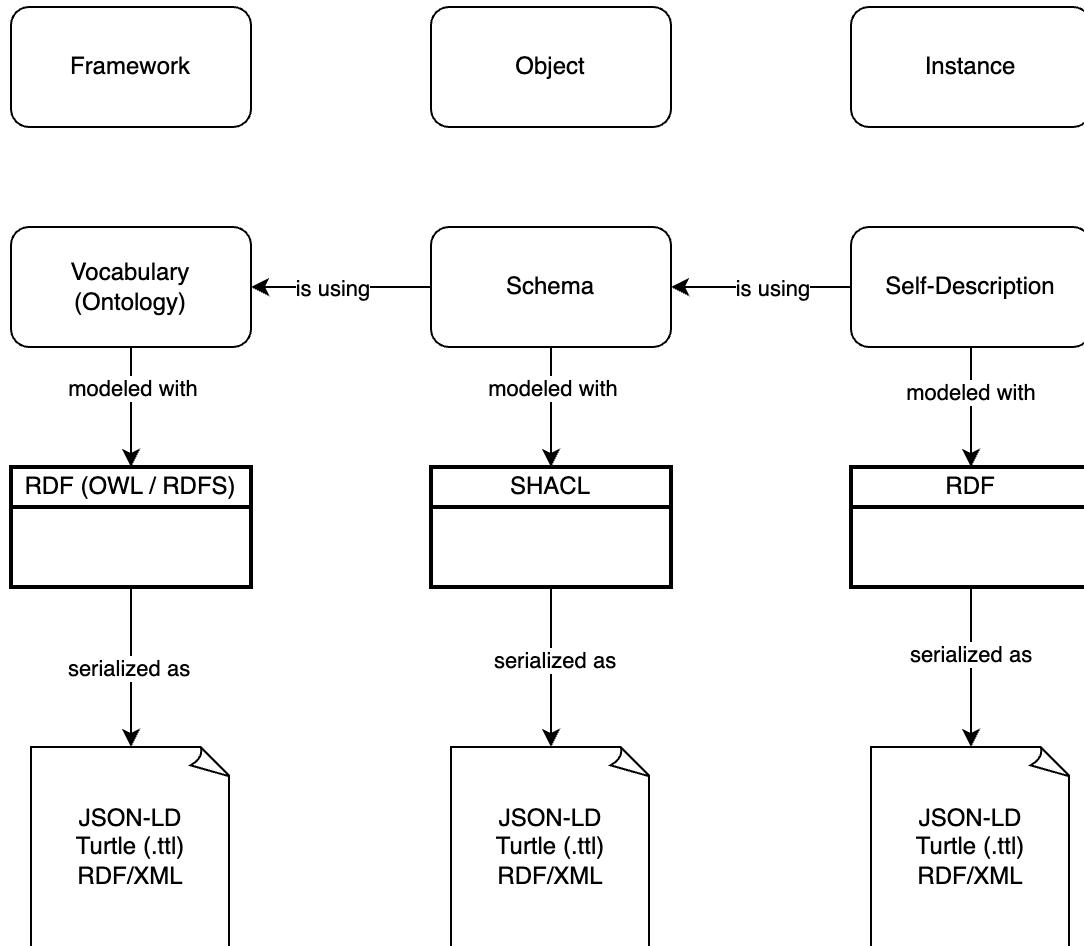
Runtime View

For the POC the System is simply deployed as a git-lab repository. A git-lab CI Pipeline starts if the configuration is changed by the data governance authority and generates the new files. If the Data Provider start the SD Tool, the SD-Tool pulls from the repository the current SHACL Constraints and Ontology.



SD Tooling Syntax Validation & Schema Validation

Vocabulary, Schema and Self-Descriptions



The vocabulary is a formal description of an ontology, representing knowledge and relationships between the terminologies, containing inference and integrity rules for reasoning.

A schema is describing a data object with constraints on the content, structure and meaning of a graph. These conditions may constrain the number of values that a property may have, the type of values, numeric ranges, string matching patterns or logical combinations of constraints.

The Self-Description is an instance of a schema object, meaning that values are assigned to the properties.

Syntax Validation

Syntax Validation during the process of creating the SD comprises the following:

- **Formatting:** Check if the file is malformed (e.g. missing brackets etc.);
- **Data Types:** Check for the correctly applied values according to data types. Allowed data Types according to [dataTypeAbbreviation.yaml](#).

The syntax validation for data types in the SD Frontend is based in the schema definition, which is the single point of truth.

The Syntax validation on the provider node is based on the schemas that are imposed by Simpl and are intended to guide the user to provide an error free Self-Description.

The Syntax validation on the Governance Authority Node ensures that only valid Self-Descriptions will be published to the catalogue.

Allowed Data Types

```

xsd:string: 'http://www.w3.org/2001/XMLSchema#string'
xsd:boolean: 'http://www.w3.org/2001/XMLSchema#boolean'
xsd:decimal: 'http://www.w3.org/2001/XMLSchema#decimal'
xsd:float: 'http://www.w3.org/2001/XMLSchema#float'
xsd:double: 'http://www.w3.org/2001/XMLSchema#double'
xsd:duration: 'http://www.w3.org/2001/XMLSchema#duration'
xsd:dateTime: 'http://www.w3.org/2001/XMLSchema#dateTime'
xsd:time: 'http://www.w3.org/2001/XMLSchema#time'
xsd:date: 'http://www.w3.org/2001/XMLSchema#date'
  
```

```

xsd:gYearMonth: 'http://www.w3.org/2001/XMLSchema#gYearMonth'
xsd:Day: 'http://www.w3.org/2001/XMLSchema#Day'
xsd:hexBinary: 'http://www.w3.org/2001/XMLSchema#hexBinary'
xsd:base64Binary: 'http://www.w3.org/2001/XMLSchema#base64Binary'
xsd:anyURI: 'http://www.w3.org/2001/XMLSchema#anyURI'
xsd:QName: 'http://www.w3.org/2001/XMLSchema#QName'
xsd:NOTATION: 'http://www.w3.org/2001/XMLSchema#NOTATION'
xsd:dateTimeStamp: 'http://www.w3.org/2001/XMLSchema#dateTimeStamp'
xsd:enum: 'http://www.w3.org/2001/XMLSchema#enum'
xsd:integer: 'http://www.w3.org/2001/XMLSchema#integer'
xsd:address: 'http://www.w3.org/2001/XMLSchema#address'
xsd:nonNegativeNumber: 'http://www.w3.org/2001/XMLSchema#nonNegativeNumber'
did:example: 'https://www.w3.org/TR/did-core/#example'
dct:location: 'http://dublincore.org/usage/terms/history/#Location-001'
trusted-cloud:meaningfulString: 'class=placeholder-from-dataTypeAbbreviation.yaml'

```

Semantic Validation

Semantic Validation during the process of creating the SD comprises:

- the verification of property patterns;
- data ranges;
- other constraints;
- the cardinality of the properties;
- the ontology/vocabulary compliance.

Examples:

- ValueRanges;
- Length;
- Pattern;
- Value Comparison;
- Memberships;
- Logical.

Constraints can be defined according to [Shapes Constraint Language](#)

Policies

For our context we want to define both access and usage policies for resources (Data, Application or Infrastructure)

We follow the definition of the Data Space Support Center:

- *Access Rules/Policy: define whether access to a resource is allowed or not.*
- *Usage Rules/Policy: define how a resource might or may not be used.*

Access control policies control the authorisation to access specific data while the data rights owner retains direct control over the data. Usage policies, including consent, regulate the permissible actions and behaviours related to the utilisation of the accessed data, which means keeping control of data even after the items have left the trust boundaries of the data owner.

<https://dssc.eu/space/BVE/357075567/Access+%26+Usage+Policies+Enforcement#Data-Space-Registry>

Following this definition the access policies are checked before the provider gives (at least partial) control over to the consumer. The usage policies describe the behavior after the consumer has access to the resource (Data, Application or Infrastructure).

Policy Language

We need a formal and machine-readable way to express and enforce the policies. We want to use Open Digital Rights Language (ODRL) to write both access and usage policies. <https://www.w3.org/TR/odrl-model/>

The key components of ODRL are:

Here are the key components of an ODRL usage policy:

1. **Asset:** The digital content or service to which the policy applies;
2. **Permissions:** Actions that are allowed with respect to the asset (e.g., read, download);
3. **Prohibitions:** Actions that are explicitly forbidden;
4. **Constraints:** Conditions or limitations that must be met for the permissions to apply (e.g., time restrictions);
5. **Duties:** Obligations that must be fulfilled by the user in order to exercise a permission (e.g., attribution, payment).

Different ways exist to serialize ODRL expressions we want to use JSON-LD for this part:

```
{
  "@context": "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Policy",
  "uid": "http://example.com/policy/123",
  "profile": "http://www.w3.org/ns/odrl/2/odrl.jsonld",
```

```

"permission": [
  {
    "target": "http://example.com/asset/image123",
    "action": "http://www.w3.org/ns/odrl/2/distribute",
    "constraint": [
      {
        "leftOperand": "http://www.w3.org/ns/odrl/2/purpose",
        "operator": "http://www.w3.org/ns/odrl/2/eq",
        "rightOperand": "http://www.example.com/vocab#nonCommercial"
      },
      {
        "leftOperand": "http://www.w3.org/ns/odrl/2/payAmount",
        "operator": "http://www.w3.org/ns/odrl/2/eq",
        "rightOperand": "0"
      }
    ],
    "duty": [
      {
        "action": "http://www.w3.org/ns/odrl/2/attribution"
      }
    ]
  }
],
"prohibition": [
  {
    "target": "http://example.com/asset/image123",
    "action": "http://www.w3.org/ns/odrl/2/modify"
  }
]
}

```

Access Policy

Here is an example for an access policy for a dataset provided by a data provider. The policy will specify who can access the data, under what conditions, and for how long.

Scenario:

- The dataset contains research data that can be accessed by different roles:
 - Researchers: Full access to the data for analysis;
 - Students: Limited access to anonymized data for study purposes;
 - External partners: Access to aggregated data for collaboration purposes.
- The access is granted for a specific period;
- The access is granted only for the geographic location of the EU.

We use different datasets for the full data, anonymized data and aggregated data.

```
{
  "@context": "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Policy",
  "uid": "http://example.com/policy/123",
  "profile": "http://www.w3.org/ns/odrl/2/odrl.jsonld",
  "target": "http://example.com/dataset/research123",
  "assigner": {
    "uid": "http://example.com/provider/dataProvider001",
    "role": "http://www.w3.org/ns/odrl/2/assigner"
  },
  "permission": [
    {
      "assignee": {
        "uid": "SECURITY_ATTRIBUTE",
        "role": "http://www.w3.org/ns/odrl/2/assignee"
      },
      "action": [
        { "name": "http://www.w3.org/ns/odrl/2/read" }
      ],
      "target": "http://example.com/dataset/research123/aggregated",
      "constraint": [
        {
          "leftOperand": "http://www.w3.org/ns/odrl/2/dateTime",
          "operator": "http://www.w3.org/ns/odrl/2/leq",
          "rightOperand": "2024-01-01T00:00:00Z"
        }
      ]
    }
  ]
}
```

```

        "rightOperand": "2024-12-31T23:59:59"
    },
    {
        "leftOperand": "http://www.w3.org/ns/odrl/2/dateTime",
        "operator": "http://www.w3.org/ns/odrl/2/eq",
        "rightOperand": "2024-01-01T00:00:00"
    },
    {
        "leftOperand": "http://www.w3.org/ns/odrl/2/spatial",
        "operator": "http://www.w3.org/ns/odrl/2/eq",
        "rightOperand": "http://www.geonames.org/external-partner-location"
    }
]
}
}
}

```

Minimal Access Policy

For the MVP we plan to support access policies with limited expressive power. It is possible to define two different actions

- <http://www.w3.org/ns/odrl/2/read>: the attribute holder is able to search for the dataset/application/infrastructure;
- <http://www.w3.org/ns/odrl/2/use>: The attribute holder can consume the dataset/application/infrastructure.

while **use** implies **read**.

For constraints we plan to support date time constraints that allow to specify when the policy should be valid.

{RESSOURCE_URI}, {POLICY_URI}, {PROVIDER_URI} are later automatically replaced with the correct URI. {SECURITY_ATTRIBUTE_URI} need to be specified but we provide a documentation with the available URI, as well as the action (read for searching and use for consumption, which implies read)

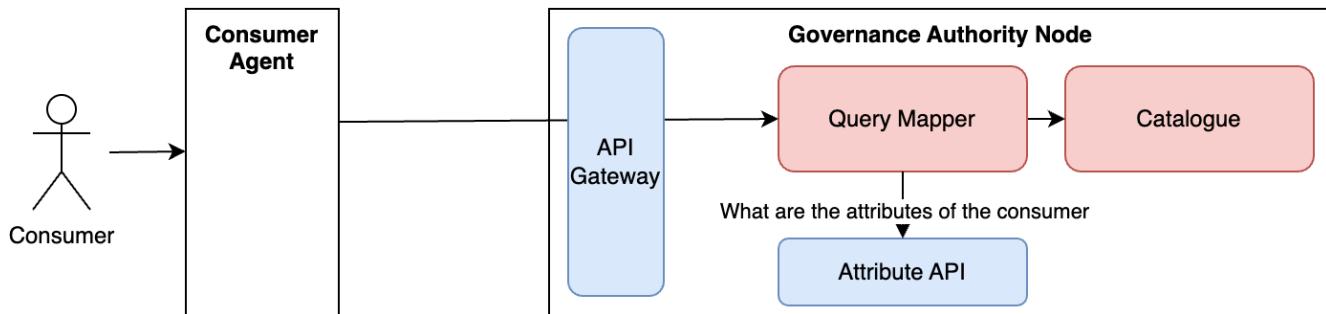
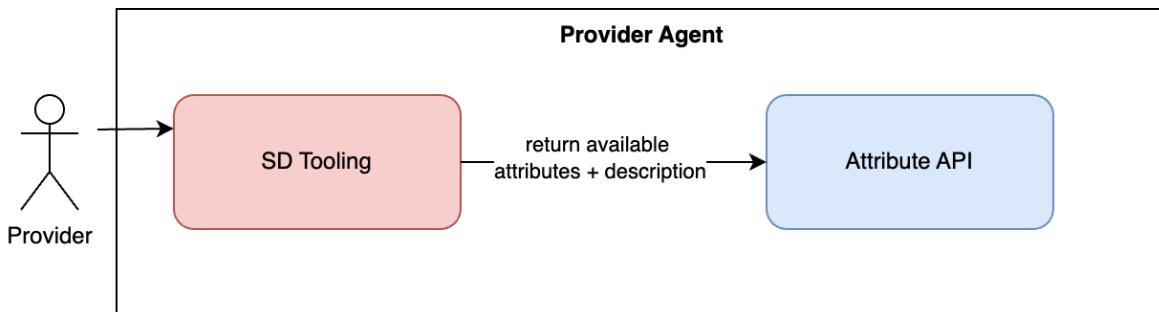
```

{
    "@context": "http://www.w3.org/ns/odrl.jsonld",
    "@type": "Policy",
    "uid": "{POLICY_URI}",
    "profile": "http://www.w3.org/ns/odrl/2/odrl.jsonld",
    "target": "{RESSOURCE_URI}",
    "assigner": {
        "uid": "{PROVIDER_URI}",
        "role": "http://www.w3.org/ns/odrl/2/assigner"
    },
    "permission": [
        {
            "assignee": {
                "uid": "{SECURITY_ATTRIBUTE_URI}",
                "role": "http://www.w3.org/ns/odrl/2/assignee"
            },
            "action": [
                { "name": "http://www.w3.org/ns/odrl/2/{read/use}" }
            ],
            "target": "{RESSOURCE_URI}",
            "constraint": [
                {
                    "leftOperand": "http://www.w3.org/ns/odrl/2/dateTime",
                    "operator": "http://www.w3.org/ns/odrl/2/leq",
                    "rightOperand": "2024-12-31T23:59:59"
                },
                {
                    "leftOperand": "http://www.w3.org/ns/odrl/2/dateTime",
                    "operator": "http://www.w3.org/ns/odrl/2/geq",
                    "rightOperand": "2024-01-01T00:00:00"
                }
            ]
        }
    ]
}

```

1. API to get all available attributes with description about the semantic. When will this be available and can we get static list so we can start developing.
 - a. Prioritizes before the MTLS;

- b. Availability not clear;
 - c. Provide a static list.
2. API to get the attributes of the searching consumer. For the use of filtering the results of the catalogue search:
- a. over the public key;
 - b. from the JWT, attributes are in the payload.
3. How to get Provider ID? While you use self-description? Can we somehow get the ID of the provider from an API to add this information to the Self-description:
- a. unique id of the agent is the public key, from the vault (HashiCorp/OCM) or the public endpoint;
 - b. self-description in long run of the participants.
4. Map Policy to ABAC (who is doing it?):
- a. ABAC only for first layer;
 - b. Second Layer with policy evaluation in EDC.



Usage Policy

The IDS Usage Control Language is based on ODRL: <https://international-data-spaces-association.github.io/DataspaceConnector/Documentation/v5/UsageControl>

The Usage Policy is part of the usage contract, as well es the Self-Description. It contains permissions, prohibitions and obligations.

Usage Policy Examples:

Allow the Usage of the Data

```
{
  "@context": "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Policy",
  "uid": "http://example.com/policy/usage/UsagePolicy001",
  "profile": "http://www.w3.org/ns/odrl/2/odrl.jsonld",
  "target": "http://example.com/dataset/TestData001",
  "action": "http://www.w3.org/ns/odrl/2/use",
  "assigner": {
    "uid": "http://example.com/provider/dataProvider001",
    "role": "http://www.w3.org/ns/odrl/2/assigner"
  },
  "permission": [
    {
      "assignee": {
        "uid": "http://example.com/roles/dataConsumer001",
        "role": "http://www.w3.org/ns/odrl/2/assignee"
      }
    }
  ]
}
```

```

        }
    ]
}
```

Use Data and Delete it After

```
{
  "@context": "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Policy",
  "uid": "http://example.com/policy/usage/UsagePolicy001",
  "profile": "http://www.w3.org/ns/odrl/2/odrl.jsonld",
  "target": "http://example.com/dataset/TestData001",
  "action": "http://www.w3.org/ns/odrl/2/use",
  "assigner": {
    "uid": "http://example.com/provider/dataProvider001",
    "role": "http://www.w3.org/ns/odrl/2/assigner"
  },
  "permission": [
    {
      "assignee": {
        "uid": "http://example.com/roles/consumer001",
        "role": "http://www.w3.org/ns/odrl/2/assignee"
      }
    ],
    "constraint": [
      {
        "leftOperand": "http://www.w3.org/ns/odrl/2/deletion",
        "operator": "http://www.w3.org/ns/odrl/2/eq",
        "rightOperand": "after_use"
      }
    ]
  }
}
```

Restricted Number of Usages

```
{
  "@context": "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Policy",
  "uid": "http://example.com/policy/usage/UsagePolicy001",
  "profile": "http://www.w3.org/ns/odrl/2/odrl.jsonld",
  "target": "http://example.com/dataset/TestData001",
  "action": "http://www.w3.org/ns/odrl/2/use",
  "assigner": {
    "uid": "http://example.com/provider/dataProvider001",
    "role": "http://www.w3.org/ns/odrl/2/assigner"
  },
  "permission": [
    {
      "assignee": {
        "uid": "http://example.com/roles/consumer001",
        "role": "http://www.w3.org/ns/odrl/2/assignee"
      }
    ],
    "constraint": [
      {
        "leftOperand": "http://www.w3.org/ns/odrl/2/count",
        "operator": "http://www.w3.org/ns/odrl/2/lteq",
        "rightOperand": "10"
      }
    ]
  }
}
```

Duration-restricted Data Usage

```
{
  "@context": "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Policy",
  "uid": "http://example.com/policy/usage/UsagePolicy001",
  "profile": "http://www.w3.org/ns/odrl/2/odrl.jsonld",
  "target": "http://example.com/dataset/TestData001",
  "action": "http://www.w3.org/ns/odrl/2/use",
  "assigner": {
    "uid": "http://example.com/provider/dataProvider001",
    "role": "http://www.w3.org/ns/odrl/2/assigner"
  },
  "permission": [
    {
      "assignee": {
        "uid": "http://example.com/roles/consumer001",
        "role": "http://www.w3.org/ns/odrl/2/assignee"
      }
    }
  ],
  "constraint": [
    {
      "leftOperand": "http://www.w3.org/ns/odrl/2/dateTime",
      "operator": "http://www.w3.org/ns/odrl/2/leq",
      "rightOperand": "2024-12-31T23:59:59"
    },
    {
      "leftOperand": "http://www.w3.org/ns/odrl/2/dateTime",
      "operator": "http://www.w3.org/ns/odrl/2/geq",
      "rightOperand": "2024-01-01T00:00:00"
    }
  ]
}
```

Extended Scenario

Another example for an extended usage policy for a dataset provided by a data provider. The policy will specify how a resource can be used once access has been granted.

A dataset contains sensitive health research data. The data provider wants to ensure that this data is used responsibly and in compliance with specific guidelines. The usage policy specifies the following:

1. The data can only be used for academic research purposes;
2. The data cannot be shared with third parties;
3. The data must be deleted after the research project is completed;
4. The data usage is monitored, and any breach of the policy will result in revocation of access.

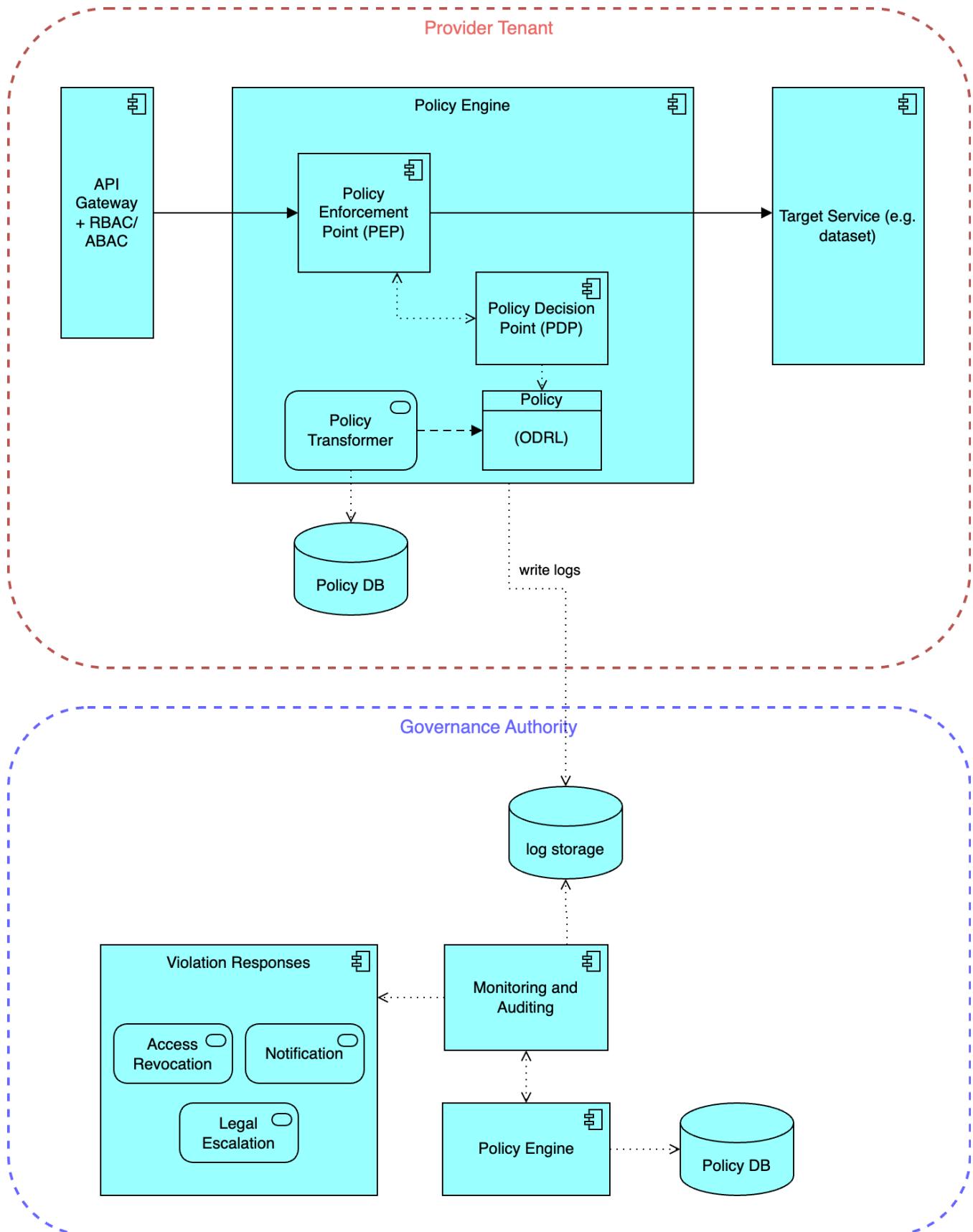
```
{
  "@context": "http://www.w3.org/ns/odrl.jsonld",
  "@type": "Policy",
  "uid": "http://example.com/policy/usage/001",
  "profile": "http://www.w3.org/ns/odrl/2/odrl.jsonld",
  "target": "http://example.com/dataset/health_research123",
  "assigner": {
    "uid": "http://example.com/provider/dataProvider001",
    "role": "http://www.w3.org/ns/odrl/2/assigner"
  },
  "permission": [
    {
      "assignee": {
        "uid": "http://example.com/roles/researcher",
        "role": "http://www.w3.org/ns/odrl/2/assignee"
      },
      "action": [
        { "name": "http://www.w3.org/ns/odrl/2/use" }
      ],
      "constraint": [
        {
          "leftOperand": "http://www.w3.org/ns/odrl/2/purpose",
          "operator": "http://www.w3.org/ns/odrl/2/eq",
          "rightOperand": "http://example.com/purpose/academic_research"
        },
        {
          "leftOperand": "http://www.w3.org/ns/odrl/2/purpose",
          "operator": "http://www.w3.org/ns/odrl/2/eq",
          "rightOperand": "http://example.com/purpose/medical_research"
        }
      ]
    }
  ]
}
```

```
{  
    "leftOperand": "http://www.w3.org/ns/odrl/2/deletion",  
    "operator": "http://www.w3.org/ns/odrl/2/eq",  
    "rightOperand": "after_use"  
}  
]  
}  
]
```

Policy Enforcement



This section present draft content for a capability falling behind the scope of the MVP (December 2024) and will be completed at a later time.



Federated Catalogue

SIMPL is using the [XFSC Federated Catalogue](#) as Catalogue for Data, Apps and Infrastructure (see architecture document of XFSC Federated Catalogue).

The Federated Catalogue is not a monolithic application. It consists of multiple components, to reuse existing technology and to allow scaling. Those components can be deployed individually (see section Deployment View).

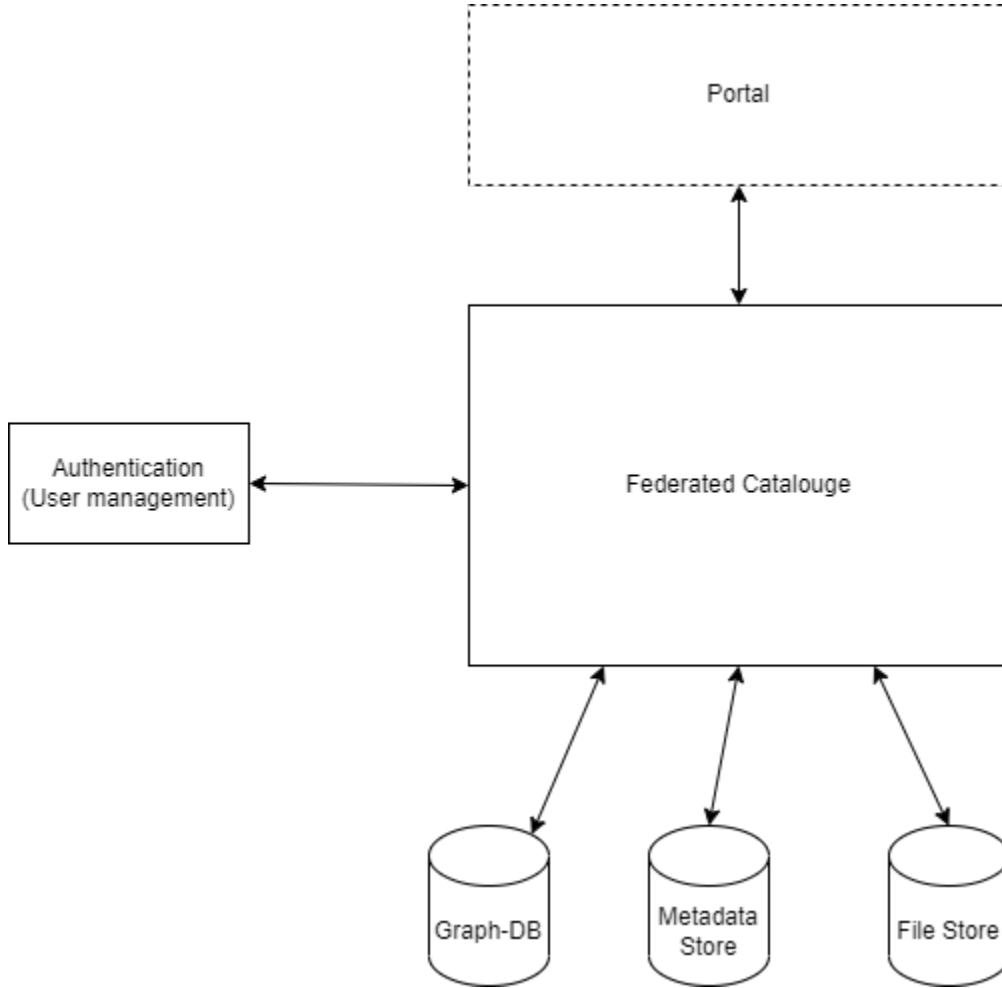


Figure 1. Components of the Catalogue

The components are

Name	Responsibility
Catalogue	Main component, implementing the core catalogue functionality.
Authentication	External component implementing the authentication flow and user management.
Graph-DB	Graph database, holding all claims contained in active Self-Descriptions. The Graph database is responsible for executing semantic search queries.
File Store	The File store is a blob storage. It holds the Self-Description files and the files for the Schemas. This includes historical versions of the Self-Descriptions and Schemas.
Metadata Store	Store for metadata on the Self-Descriptions, and Schemas stored in the File Store.

The architecture of the core component is described in the next sections.

Authentication

The authentication component is responsible for authenticating users. This is not a central component of the catalogue, as it will be implemented by Lot 1 "Authentication & Authorization" of the GXFS-DE project. For the catalogue implementation, a mock integration is shown, using common, off the shelf software that implements the OpenID Connect standard^[14].

The responsibilities of the authentication components are:

- Storage of Users;
- Storage of user roles for a Participant.

A user belongs to only one Participant, on whose behalf he or she acts (see specification section 2.4 for more details).

For the implementation, Keycloak will be used. It is widely used and also part of the implementation of other lots. Therefore, this integration of different lots is simplified. The user will get a JSON Web Token (JWT^[15]) with user claims and authorities, which is used to authenticate requests to the catalogue REST API.

An alternative implementation would be Lissi^[16]. It is not further considered, as it is not as mature as Keycloak.

Graph database

The graph database holds the claims of verified, active Self-Descriptions. Claims of Self-Descriptions that fail the verification are not added to the graph database. Claims of Deprecated, Expired or Revoked Self-Descriptions will be deleted from the Graph database.

The Graph Database can be considered as a kind of search index. The single source of truth are the active Self-Descriptions, stored in the File Store. This means at any point in time the Graph database can be rebuilt from scratch by reimporting the claims of the Self-Descriptions. This allows the following:

1. **Backup:** An explicit backup of the Graph database is not needed. Backing up the Self-Description files (located in the File Storage) and the metadata (located in the Metadata Store) is sufficient to allow the rebuild of the Graph Database;
2. **Scalability:** Querying the Graph database might be the most critical part regarding performance. Therefore, the Graph Database can be replicated in the future by multiple, independent instances. Since there are no strict consistency requirements, changes in the Graph can be applied independently. In the control flow, all write operations on Self-Descriptions pass the Metadata Store. Therefore, the consistency can be enforced by that database.

Generically returning the Self-Description files containing claims that influence query response is not possible. To get the relevant Self-Description files, the query to the Graph Database can be formulated to return the Gaia-X entity that is the *credentialSubject* of a Verifiable Credential. Then this can be used as a filter for the Self-Description endpoint, to download the Self-Description file.

Neo4j is used as implementation of the Graph database.

Limitation: queries to non-Enterprise Neo4j Graph database return empty record when no results found, rather than empty list.

When there is no data in the Graph database, i.e., no claims extracted from Self-Description, there is still a configuration node for the neosemantics module^[17], which enables Neo4j to support the RDF data model, which is required here. openCypher queries over all nodes without a WHERE clause or without specifying relationships always return this node, unless regular users are revoked access from the configuration node as follows:

```
DENY MATCH { * } ON GRAPH neo4j NODES _GraphConfig TO PUBLIC
```

However this revoke operation is only supported in Neo4j enterprise.^[18] We chose not to implement a workaround that involves query rewriting, as this may have harmful side effects.

File Store

The File Store is responsible to persist all file-based content submitted to the catalogue. These are Self-Descriptions and Schemas.

For the sake of simplicity, a folder in the file system is used as file store. For future scalability the file store can be simply realized using an Object Storage or Database.

Metadata Store

In the Metadata Store persists the metadata for the elements (Self-Descriptions, Schemas and Trust Anchors). It allows to efficiently identify the relevant files in the file storage, to process the incoming requests.

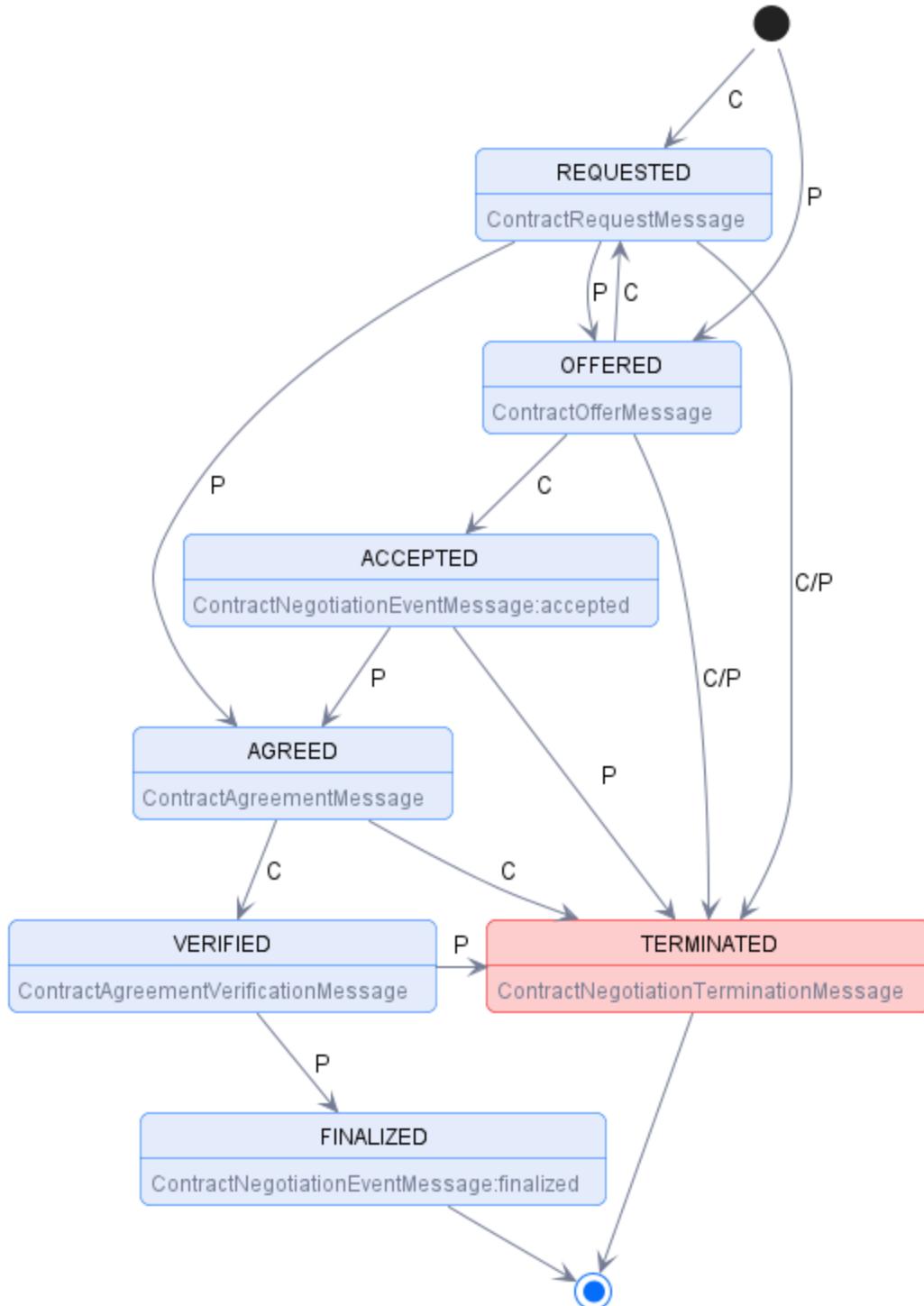
It is realized as relational database (e.g., PostgreSQL or MariaDB). Since all write requests are handled by the database, the transactional functionality guarantees the consistency of the data.

Todo: include the sections in our Wiki

- [Federated Catalogue - Verification of self-description \[WiP\]](#)
- [Architecture Decisions](#)

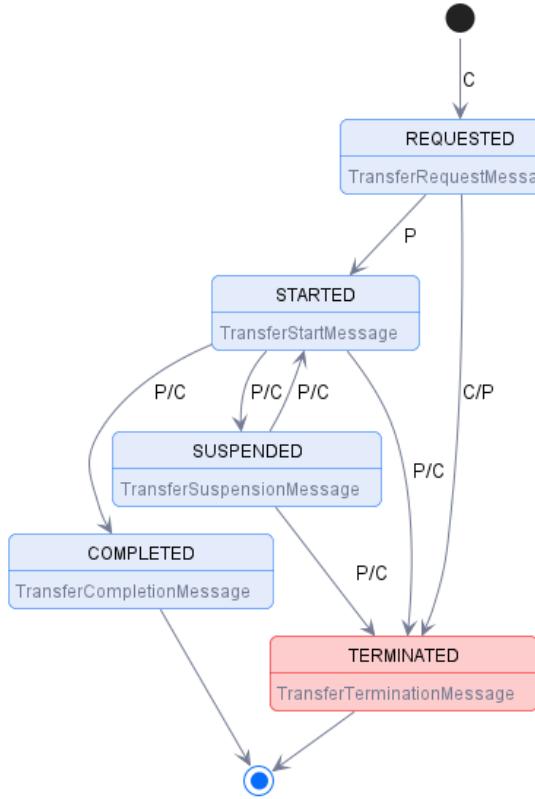
Contract

The state machine for a Contract Negotiation is visualised in the figure below:



Transitions marked with C indicate a message sent by the [Consumer](#), transitions marked with P indicate a [Provider](#) message. Terminal states are final; the state machine may not transition to another state. A new CN may be initiated if, for instance, the CN entered the TERMINATED state due to a network issue. The associated message types to switch into the mentioned states are denoted in the bottom part of each status box. For further information refer to the specification section [contract negotiation protocol](#).

After successful contract negotiation the Transfer Process can be invoked via the data plane. The state machine for the transfer process is shown in the diagram below:



Any implementation of Eclipse Dataspace Protocol must implement the state machines shown above where respective contract messages respectively Transfer messages induces switching of states.

In EDC connector the IDS dataspace protocol is implemented. Via State Transition Functions any specific actions can be triggered like invoking consent or contract managers. This is described in [Contract Negotiation Architecture](#).

Steps to be done for Contract Negotiations:

Pre-requisites a provider has to do to publish a service offer at a connector (using a provider connector):

1. Create an Asset on provider side;
2. Create a Policy on provider side;
3. Create a Contract definition on provider side.

Steps to be done on consumer side to request a service offer from a connector (using a consumer connector):

1. How to fetch catalogue on consumer side;
2. Negotiate a contract on consumer side;
3. Getting the contract agreement id.

These steps are described in [Transfer-01-negotiation](#).

After successful negotiation process the transfer process can be started.

Infrastructure Provisioning

In the first step, the Infrastructure Provider (or APP/Data Providers, upon their need) can use the Deployment Script Management to add their deployment scripts.

These scripts are Crossplane configuration files, that at the time of execution, will:

- A) Provision the infrastructure resources (VM, Container or Storage);
- B) Deploy Applications over the infrastructure resources (if needed);
- C) Load data sets or images on the infrastructure resource (if needed).

After adding the deployment script (via the available UI or the API), the DeploymentScriptID, which is a unique ID for that deployment script will be returned.

In the second step, at the time of creating infrastructure offerings (or bundles of app/data + infrastructure, as it would be required for use cases explained in BP 09A and BP 09B), the DeploymentScriptID is being added to the Self-Description.

In the third step, when the offer has been selected and successfully contracted, the Infrastructure Provisioner API (the same API that handles the addition/removal and modifications of the Deployment Scripts in step 1) is being called, and the DeploymentScriptID will be passed to that API for execution.

Therefore, that DeploymentScriptID is being validated, and if the validation is successful, will be executed. The infrastructure Provisioner Module will (as explained above):

- A) Provision the infrastructure resources (VM, Container or Storage);
 - B) Deploy Applications over the infrastructure resources (if needed);
 - C) Load data sets or images on the infrastructure resource (if needed).

And will share back the access data with the consumer.

The communication between the triggering module and the infrastructure provisioner is done via a message broker, to keep the process asynchronous.

Data Sharing

The data sharing between two participant agents is done via two connectors based on Eclipse [Dataspace Protocol](#) relying on the [IDSA Dataspace Protocol](#). Dataspace protocol is divided into two parts: First Contract Negotiation has to be invoked and after successful negotiation the Transfer process can be invoked. Contract Negotiation is done via [contract negotiation protocol](#) for the data exchange service based on the trust protocol defined by Gaia-x.

The proposed data transaction model scope is compliant to the EDC Dataspace Protocol.

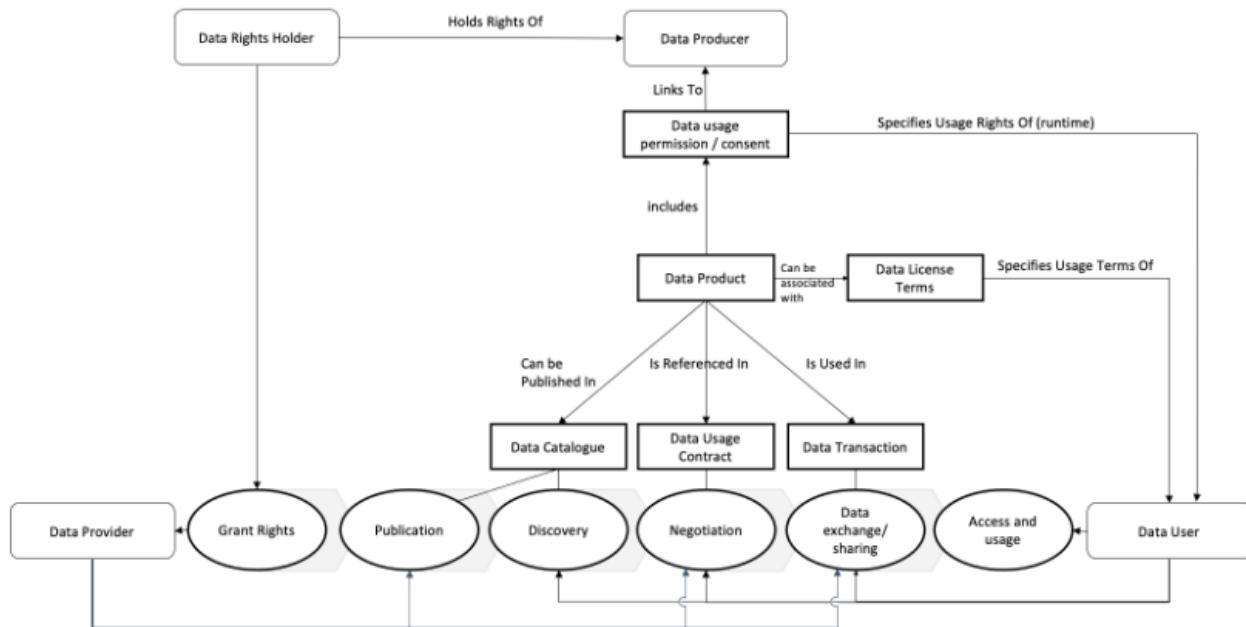
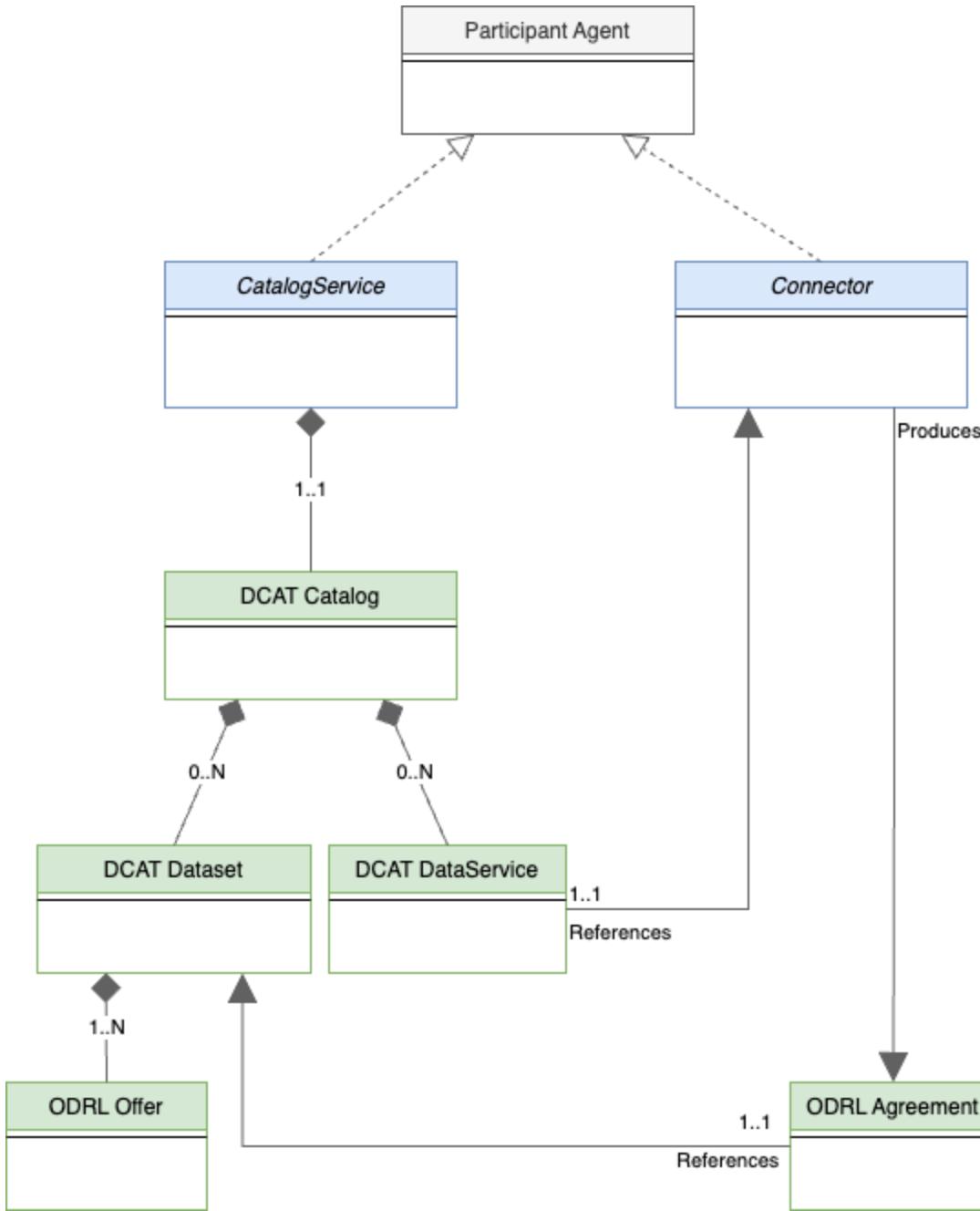


Figure 1: Scope of data transaction

The Information model of the dataspace model is described [here](#).



The figure sketches two implementations of a participant agent:

- A **Catalog Service** is a **Participant Agent** that makes a **DCAT Catalog** available to other **Participants** offering Data services and assets published by providers.
- A **Connector** is a **Participant Agent** (consumer) that performs **Contract Negotiation** and **Transfer Process** operations with another **Connector** aka participant agent of a Provider. An outcome of a **Contract Negotiation** may be the production of an **Agreement**, which is an **ODRL Agreement** defining the **Usage Policy** agreed to for a **Dataset**.

For further information refer to the specification section [model](#).

[EDC connector](#) has implemented the above-mentioned dataspace protocol as well as the depicted [data](#) and [control](#) planes and an additional [Management API](#). This management API is described in detail [here](#).

The Transfer process is described in [Transfer Process Architecture](#) and is implemented providing special extension to back-end systems.

In the context of the MVP, data orchestration refers to the data plane component responsible for the actual data transfer that takes place after a contract is established between the parties through the connector's control plane. This orchestration of data flow is a crucial step, as it translates contractual agreements into real actions for exchanging data between a source and a destination.

For the MVP release, this component will be implemented as an extension of the EDC connector, it has been depicted as an external entity in various diagrams and system architectures. This approach underscores the goal of creating an external and independent solution that is agnostic of the specific

connector used. Such independence is achievable as long as the connector supports the IDSA Dataspace Protocol, which is a key requirement to ensure interoperability within distributed ecosystems like sovereign data spaces or shared data infrastructures.

The primary role of this orchestrator is to serve as a bridge between the actual data source, located outside of the Simpl system, and the designated destination where the data is intended to flow. It ensures seamless connectivity between these two points, handling the complexities of transferring data across systems that may differ in protocols and technology. Additionally, the orchestrator is designed as a specific component tailored to each type of data source. This specialization allows to externalize the technical management of heterogeneous data sources that will be handled in the Simpl scenario, reducing complexity and promoting flexibility in the integration of various data ecosystems.

Steps to be done for transfer process:

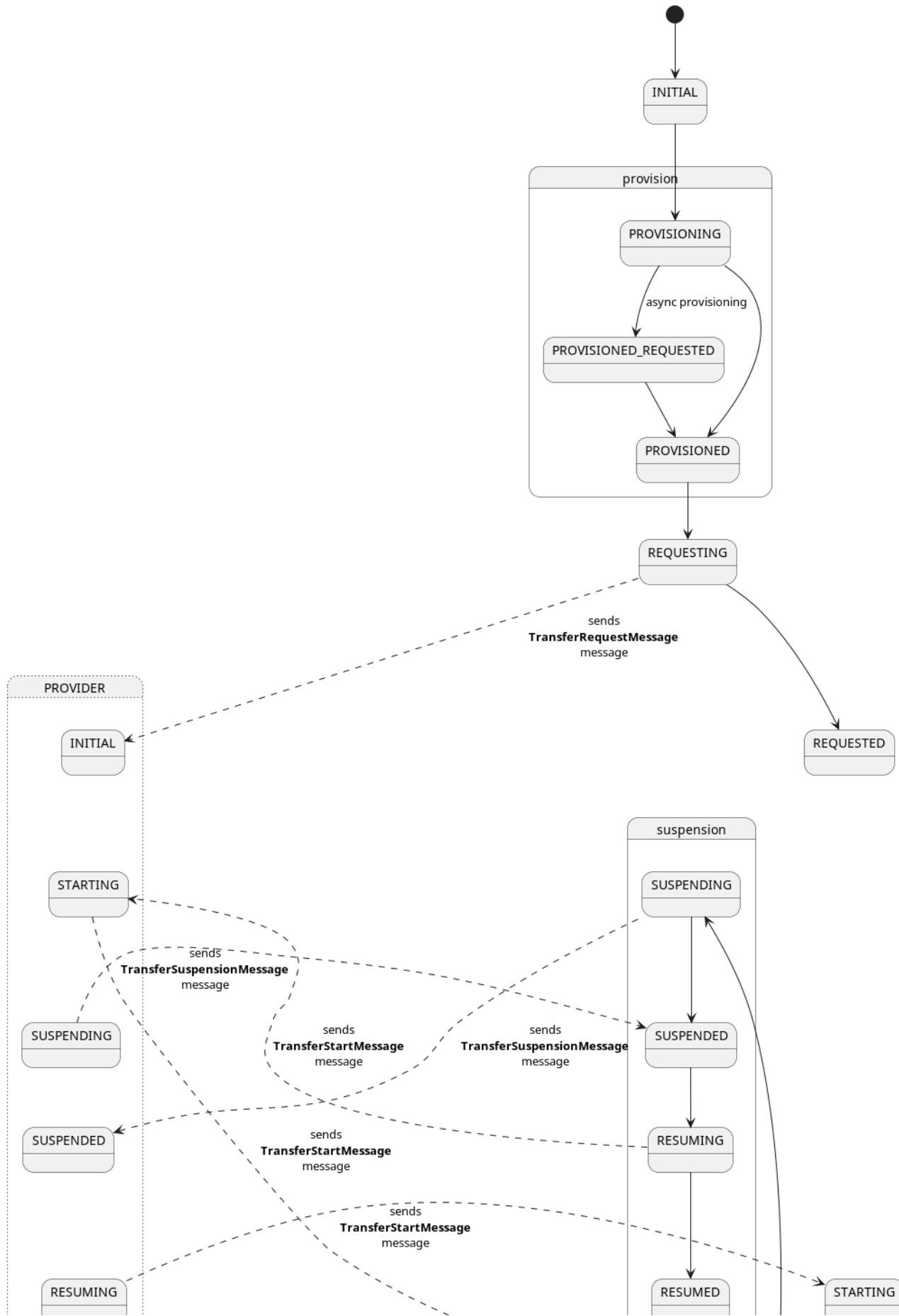
Either pull or push pattern can be used for transfer process:

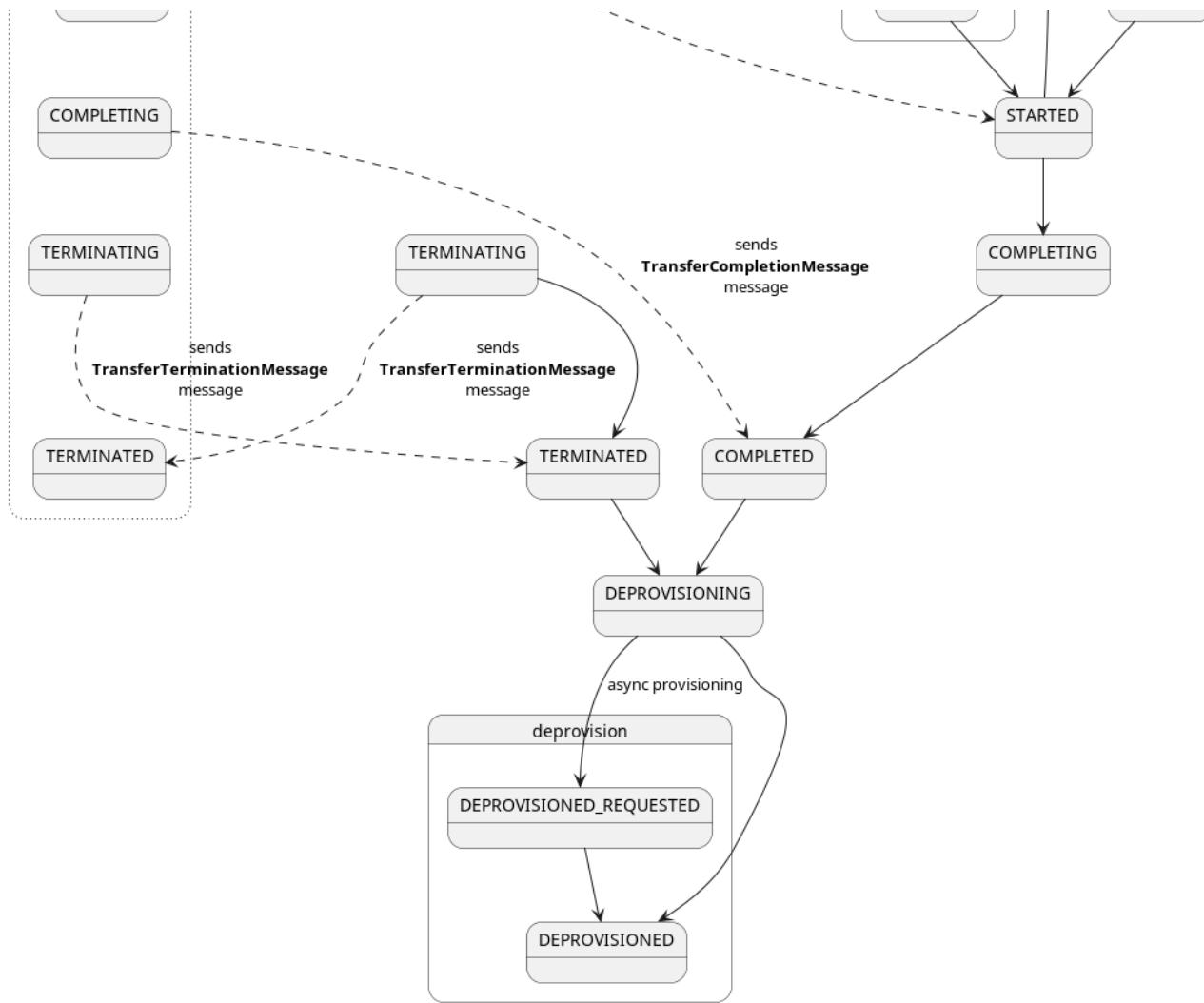
1. Consumer pull
2. Provider push

Description according to SAMPLES from EDC: <https://github.com/eclipse-edc/Samples/tree/main/transfer>

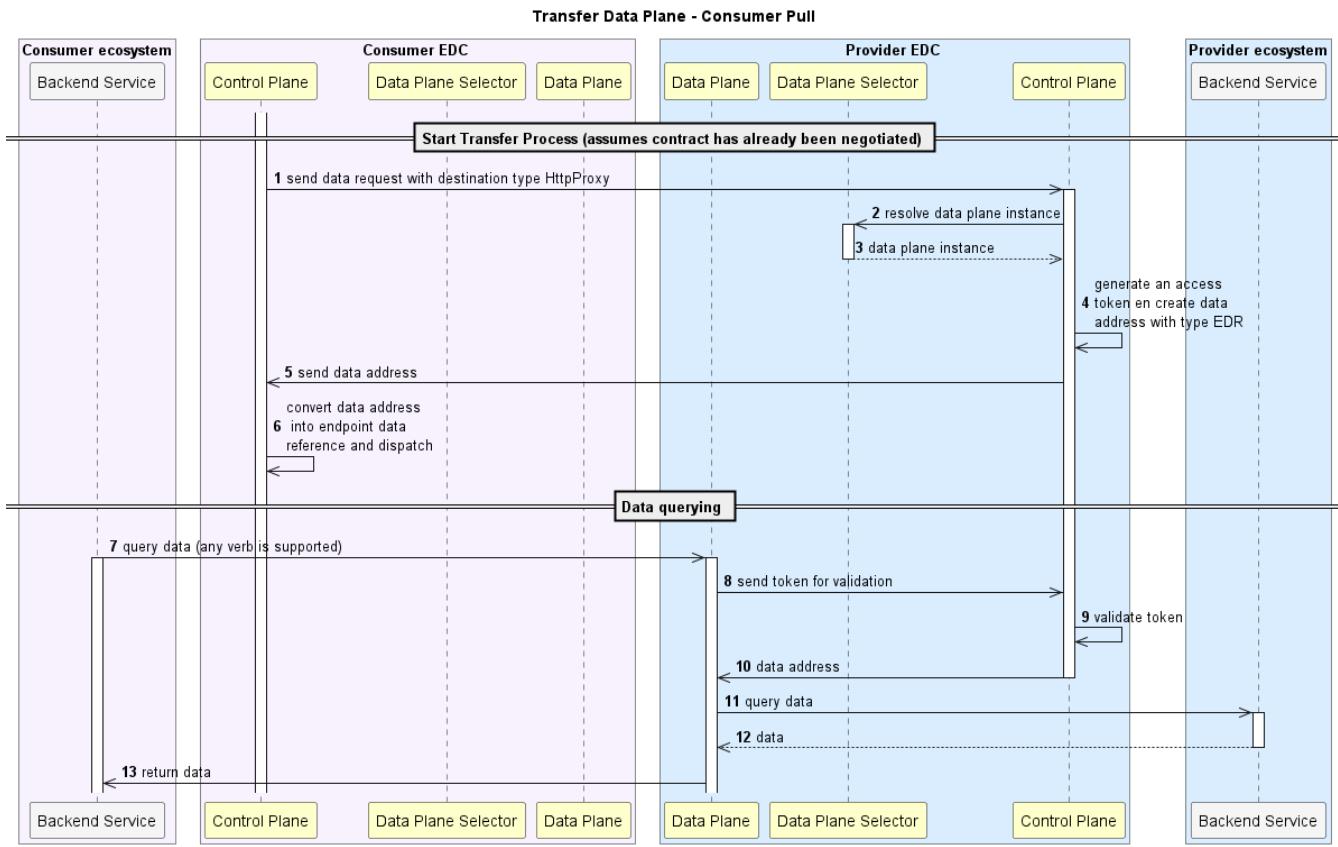
Consumer pull

Following diagram presents the state machine for this case:





Following diagram presents the sequence diagram for this case:



1. Provider and consumer agree to a contract (not displayed in the diagram);
2. Consumer initiates the transfer process by sending a `DataRequest` with destination type `HttpProxy`;
3. Provider Data Plane Selector is queried to find a suitable instance;
4. Provider Control Plane build a `DataAddress` which type `EDR`, whose:
 - endpoint corresponds to the public API of the selected Data Plane;
 - auth key is `Authorisation`;
 - auth code is a signed token generated by the Control Plane with claims;
 - dad containing the encrypted `DataAddress` of the actual data source (provider ecosystem);
 - cid claim containing the contract id.
5. This `DataAddress` is sent to the consumer Control Plane through DSP protocol;
6. Consumer Control Plane converts the `DataAddress` into a `EndpointDataReference` object and dispatch it through the `EndpointDataReferenceReceiverRegistry`.

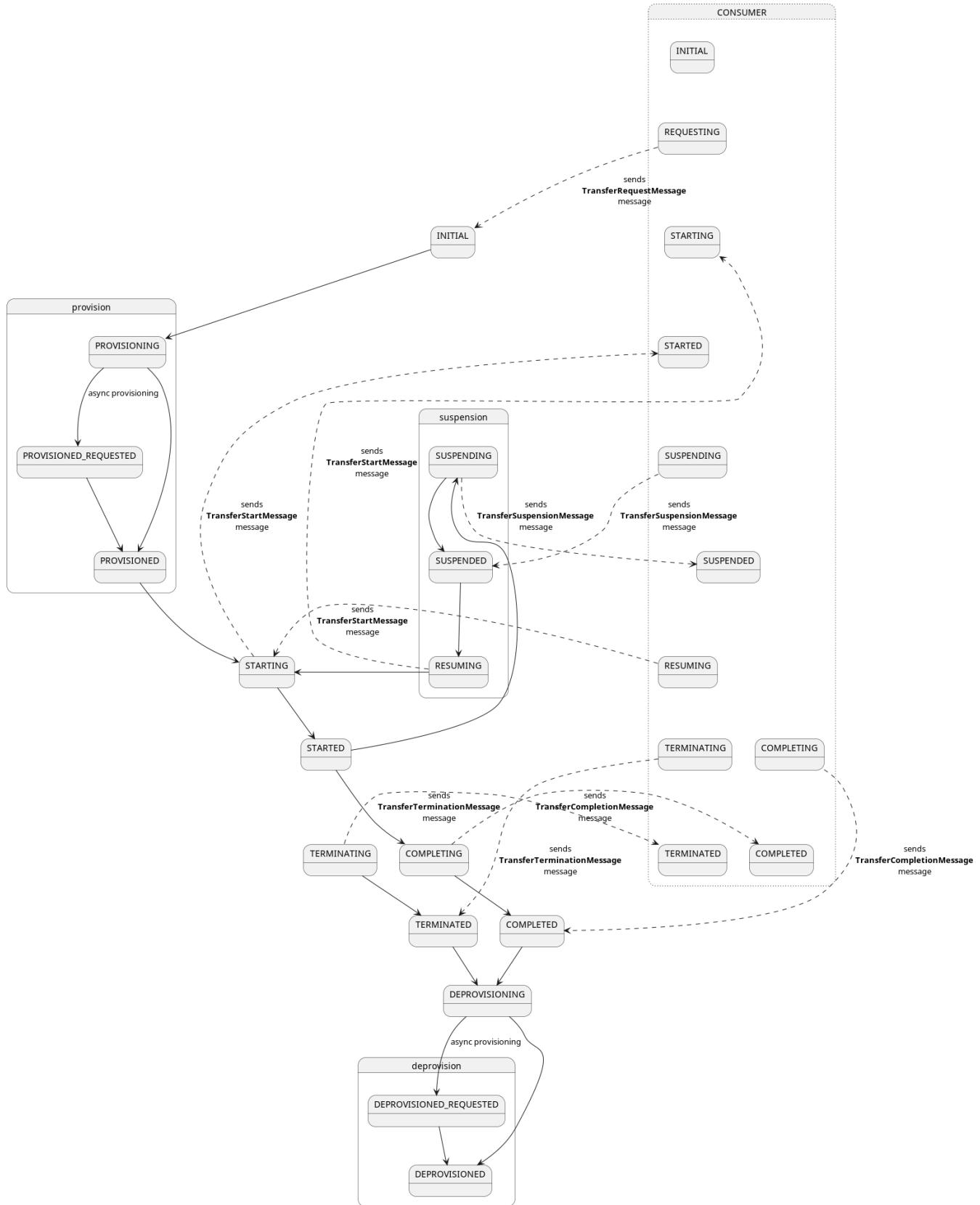
Once this process is completed, the consumer backend applications can use the received `EndpointDataReference` in order to query data from the provider Data Plane, by simply providing the provided token in the request header.

NOTE: For a Data Plane instance to be eligible for the Consumer Pull transfer, it must:

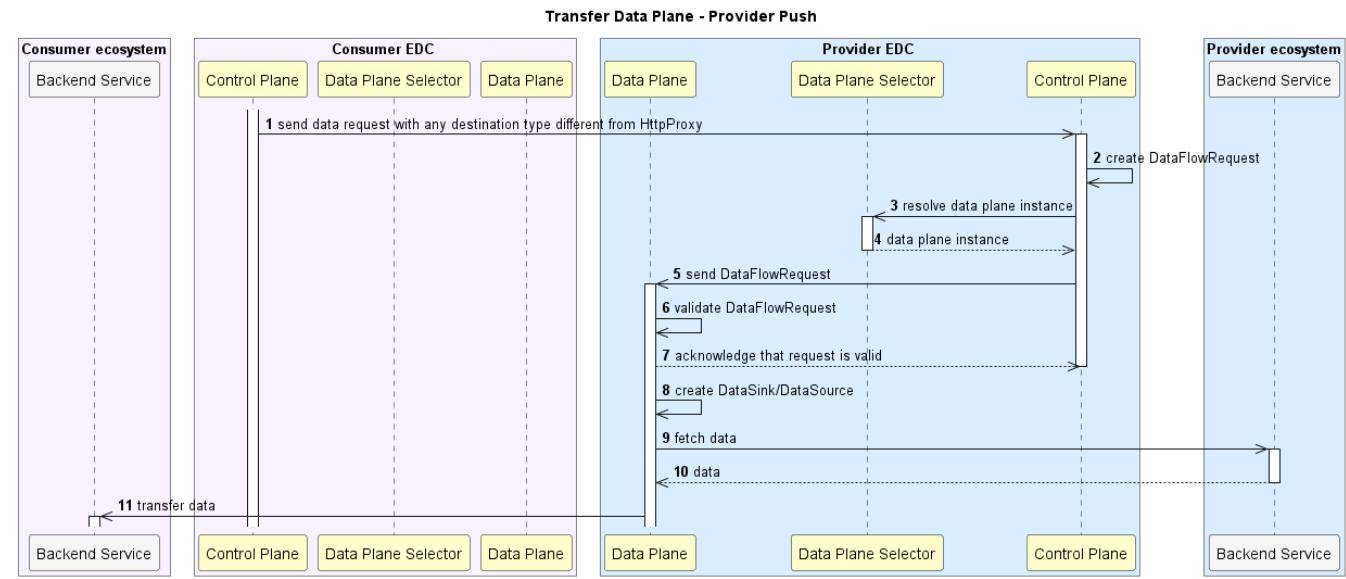
- contains `HttpProxy` in the `allowedDestTypes`;
- contain a property which key `publicApiUrl`, which contains the actual URL of the Data Plane public API.

Provider push

Following diagram presents the state machine for this case:



Following diagram presents the sequence diagram for this case:



1. Provider and consumer agree to a contract (not displayed in the diagram);
2. Consumer initiates the transfer process, i.e. sends `DataRequest` with any destination type other than `HttpProxy`;
3. Provider Control Plane retrieves the `DataAddress` of the actual data source and creates a `DataFlowRequest` based on the received `DataRequest` and this data address;
4. Provider Control Plane asks the selector which Data Plane instance can be used for this data transfer;
5. Selector returns an eligible Data Plane instance (if any);
6. Provider Control Plane sends the `DataFlowRequest` to the selected Data Plane instance through its control API (see `DataPlaneControlApi`);
7. Provider Data Plane validates the incoming request;
8. If request is valid, Provider Data Plane returns acknowledgement;
9. `DataPlaneManager` of the Provider Data Plane processes the request: it creates a `DataSource/DataSink` pair based on the source /destination data addresses;
10. Provider Data Plane fetches data from the actual data source (see `DataSource`);
11. Provider Data Plane pushes data to the consumer services (see `DataSink`).

Data Visualisation

For visualisation the component [Apache Superset](#) is chosen. Superset is a modern [data exploration and data visualization platform](#). It integrates well with a variety of data sources, and it is open source under the [Apache License](#). It comes out of the box with features to create a [dashboard](#) or how to [explore data](#). It also provides [Security Configurations](#). A REST API for user & role management can be enabled and even permissions can be customised. Superset's public [REST API](#) follows the [OpenAPI](#) specification and is documented [here](#).

The community also provides [automatic builds](#) for multi platforms and even prebuild docker builds from a [Superset Docker Hub repository](#).

Logging, Monitoring & Reporting

Types of logs - Reference model

The following table identifies the different types of logs that can be generated by an IT system together with their definition/description:

Grouping	Type of logs	Description
Business logs	Business logs	Record significant events or actions (related to steps within a business process or other functional use cases) that occur within a system, typically used for security, audit, and troubleshooting purposes.
Technical logs	Application logs	Record events and activities generated by an application during its runtime, typically used for troubleshooting, monitoring performance, and auditing activities within the application.
	Database logs	Record events and activities generated by a database (queries, transactions, schema changes), typically used for troubleshooting, ensuring data integrity and auditing access.
	System logs	Record events and activities generated by the operating system (OS) and system-level processes. These logs provide valuable information for monitoring system health, diagnosing issues, and ensuring security. System logs can include: low-level system events (kernel event, hardware error), system-level events (service startups/shutdowns/failure), authentication and authorisation events (login attempts, privilege escalation).
	Network logs	Record events and activities related to network traffic, devices, and communications within a network. These logs are essential for monitoring network health, diagnosing issues, and ensuring security. Network logs can include: firewall logs (allowed/denied connections, intrusion detection alerts, security policy violations), router and switch logs (device

		startups, interface status changes, routing protocol updates), DNS logs (queries/responses, cache activity, DNS server configuration changes and errors), proxy logs (user access, URL requests, content filtering, bandwidth usage) and network traffic logs (packet-level data, including source and destination IP addresses, port numbers, protocols, packet payloads).
	Security logs	Security logs are not a distinct type of log, they are a subset of all the other logs listed above, which allow to detect and respond to security incidents effectively. Ex: Intrusion detection alerts, security policy violations, anti-virus scans.
Infrastructure metrics	Infrastructure metrics	A metric is a piece of data that has a name, optional labels, and a value. It is not a log per-se, as they need to be retrieved by periodically scrapping an endpoint of the host system (pull instead of push paradigm). Once retrieved, the information is then persisted as a log.
Health check	Health check	A health check is a procedure that helps to determine if a component is functioning correctly or not. Just like infrastructure metrics, health checks are not logs per-se, it is an API exposed by each component to return a simple status on the health of the component, which is queried periodically.

Submission of a contract offer by a provider to a consumer

For the sake of simplicity, application, database, system, network and security logs are grouped under the more generic term of **Technical Logs**.

Use Cases and Types of Logs

Use case	Type of logs required	Type of metrics required	Description
Log and monitor business actions, mostly for audit purposes.	Business logs		A business log in this case represents a specific step in a business process that is relevant/meaningful to be tracked. E.g. Submission of an onboarding request.
Log and monitor consumption of a resource (infra/data/app) for various reasons (billing, audit, policy enforcement, regulations compliance ...).		Infrastructure metrics	Depending on the type of data or infrastructure resource that is being consumed, different metrics can be relevant: CPU, RAM, I/O, transfer speed, ...
	Technical logs		For application usage and for some data usage cases, application and database logs will give information on what is being done with the data/application.
Log and monitor the usage of a Simpl-Open agent (of its components) for the purposes of audit and troubleshooting.	Technical logs		All types of technical logs are relevant for troubleshooting purposes and some may also be relevant for audit.
	Business logs		A business log is generated for each incoming and outgoing operation at the boundaries of the agent (communication towards Tier 1 or Tier 2 users).
		Infrastructure metrics	Infrastructure metrics generated by the deployed components of the agent (CPU, RAM, Disk, ...).
Monitor the health of the Simpl-Open agent.		Health check	Health is not logged but only monitored (the monitoring queries each technical component in real time to get its health status).

Business Logging & Monitoring

Business logs are generated for each type of operation on the Simpl-Open agent:

1. For synchronous operations:
 - a. Request;
 - b. Response.
2. For asynchronous operations:
 - a. Request;
 - b. ACK of the request;
 - c. Callback;
 - d. ACK of the callback.

Business logs are generated in 2 places (in 2 different Elastic indexes):

1. The Tier 1 API Gateway for Human to Machine interactions;
2. The tier 2 API Gateway for M2M interactions.

Business logs contain the following fields:

- a. **Timestamp** - Date and time at which the log was created;
- b. **Origin** - Reference to the end-user (Tier I) or Simpl-Open agent (Tier II) that initiates the HTTP call;
- c. **Destination** - Reference to the end-user (Tier I) or Simpl-Open agent (Tier II) which is targeted by the HTTP call;

- d. **Business Operations** - Reference to the operation that is triggered (List to be defined);
- e. **Message type** - For both sync and async transactions, 4 types: request, request ACK, response, response ACK;
- f. **Correlation ID** - ID automatically generated by the first request in a transaction and is reused by the response and ACKs to correlate between them messages that are part of a same transaction.

Business operations reference list:

Business Process	Step in BP	Business Operation	Technical API call/response
BP 03A	1	Submission of an onboarding request by a provider or consumer	POST /public/onboarding-api/onboarding-request
	3 - Yes	Approval of an onboarding request by the Governance Authority	POST /private/onboarding-api/onboarding-request/*/approve
	3 - No	Rejection of an onboarding request by the Governance Authority	POST /private/onboarding-api/onboarding-request/*/reject
	10 - Yes	Confirmation of successful onboarding of a provider or consumer	POST /private/onboarding-api/onboarding-request/*/approve
	10 - No	Confirmation of failed onboarding of a provider or consumer	POST /private/onboarding-api/onboarding-request/*/reject
BP 05	4	Submission of a self-description to the catalogue by a provider	POST /api/sd/enrichAndValidate POST /api/sd/publish
	12 & 17	<u>Publication of a self-description to the catalogue by the Governance Authority</u>	
BP 06	1	Search in the catalogue	POST /api/search/v1/quick POST /api/search/v1/advanced
	6	<u>Submission of a resource's access request by a consumer</u>	
BP 07		Submission of a contract request by a consumer to a provider	POST /contract-negotiation/catalog POST /contract-negotiation/negotiate GET /contract-negotiation/negotiate POST /contractnegotiations
		Submission of a contract offer by a provider to a consumer	POST /contract-negotiation/catalog
		Acceptance of a contract offer by a consumer to a provider	N/A
		Submission of a contract agreement by a provider to a consumer	POST /contract/v1/credentials/agreements/{contractAgreementId}/definitions/{contractDefinitionId}
		Verification of a contract agreement by a consumer to a provider	POST /contractnegotiation/v1/signed/{contractAgreementId}/{signed} POST /contract/v1/credentials/agreements/{contractAgreementId}/definitions/{contractDefinitionId}
BP 08	1	Submission of an infrastructure resource request by a consumer	POST contract-consumption/transfer/start
	5	Completion of an infrastructure resource deployment by a provider	POST infrastructure/scripts/trigger
BP 09A	1	Submission of a request to transfer a data resource	POST contract-consumption/transfer/start
	6	Completion of a data resource transfer by a provider	POST contract-consumption/transfer/status/{id}
	6	<u>Completion of data resource transfer by a consumer</u>	
BP 09B	1	Submission of a request to load data/application on a provider infrastructure by a consumer	POST contract-consumption/transfer/start
	8	Confirmation of a data/application resource deployment	POST contract-consumption/transfer/status/{id}

Next to this predefined list of business operations, Simpl logs all incoming and outgoing requests between agents.

Technical implementation

Routes and ABAC/RBAC rules are loaded in the API Gateways through YAML files.

We will create a separate configuration YAML file that maps routes and specific parameters (e.g. HTTP 200 response code). For MVP, static configuration will be used. After MVP it is aimed to support hot config changes.

Resource Consumption Logging & Monitoring

Consumption of a resource (infrastructure/data/application) is logged and monitored for 2 main uses cases :

- Policy enforcement;
- Billing.

The following (sub-)processes are considered:

1. Data Consumption
 - a. Direct access to the dataset (BP 09A);
 - b. Data is accessible from an infrastructure tenant (BP TBD - possibly extension of 09A);
 - c. Data is accessible through a built-in application deployed on infrastructure tenant (BP 09B);
2. Infrastructure Consumption (BP 08).

For each of these scenarios, below Data Usage and Infrastructure Usage sections depict the applicable types of usage policy (which also drives billing) and how consumption can be monitored for each of them.

Data Usage

Direct access to the dataset

In this scenario, the data is shared directly between the provider and the consumer (outside of Simpl-Open) and as such no usage policy can be enforced (only "legal enforcement" possible). It corresponds to the "allow usage of data" and "use data and delete afterwards" policies.

This also implies that billing always happens as a one-time payment, upfront of the consumption (possible extension to BP 07).

There is thus nothing that Simpl-Open agent can log or monitor during consumption.

Data is accessible from an infrastructure tenant

In this scenario, the data provider shared the data on an infrastructure tenant provisioned by an infrastructure provider.

2 types of usage policy are considered, which can be technically enforced and billed:

1. Based on number of usages (e.g. access the data 3 times)
2. Based on duration (e.g. access the data for 7 days)

In both cases, policy enforcement and billing can be performed based on the logs from the storage.



Architecture assumptions:

- It is assumed that VM and containers always have an attached storage;
- It is assumed that Simpl-Open only supports natively S3-compliant storage but is extensible to support other storages (offering an API).

The logs (e.g. storage, bandwidth) are collected over HTTP through the S3 logging API.



The exact list of logs that will be collected by Simpl-Open and the mechanism to collect these logs are still to be defined based on what is offered by the S3 logging API.

Data is accessible through a built-in application deployed on infrastructure tenant

In this scenario, the data provider gives the consumer access to an application that offers restricted viewing (such as read only) or processing capabilities over the data resource. Only Scenario 1 is considered (a stand-alone application will be deployed on a dedicated infrastructure resource per consumer).

1 type of usage policy is considered, which can be technically enforced and billed:

1. Based on duration (e.g. access the data for 7 days)



Architecture assumptions:

- It is assumed that the application is always deployed and terminated together with the infrastructure resource as part of deployment script;
- It is assumed that Simpl-Open only supports natively applications deployed on Kubernetes but is extensible to support other platforms (offering an API).

In this case, monitoring the status of the underlying infrastructure resource is sufficient.

To do so, the following 2 options exist:

1. Collecting log files from the infra resource;
2. Collecting logs from the infrastructure provider API.

The first option could be more restrictive as it requires access to the infrastructure resource itself.

Simpl-Open therefore implements option 2 and collects logs through the kube-api exposed by the infrastructure provider.

Infrastructure Usage

2 types of usage policy are considered, which can be technically enforced and billed:

1. Based on duration (e.g. access to a VM for 7 days);
2. Based on resource utilisation (e.g. CPU, RAM, storage, bandwidth).

In the first case, monitoring the status of the infrastructure resource is sufficient and in the second case, it requires access to infrastructure metrics of the resource.

i Architecture assumptions:

- It is assumed that Simpl-Open only supports natively:
 - S3-compliant storage
 - Kubernetes containers platform
 - VMWare virtual machines

but is extensible to support other platforms (offering an API).

Both status of the resource and infrastructure metrics can be collected through the infrastructure provider APIs:

- S3 API for storage;
- kube-api for containers;
- VMWare API for VMs.

i The exact list of logs/metrics that will be collected by Simpl-Open and the mechanism to collect these logs are still to be defined based on what is offered by the APIs.

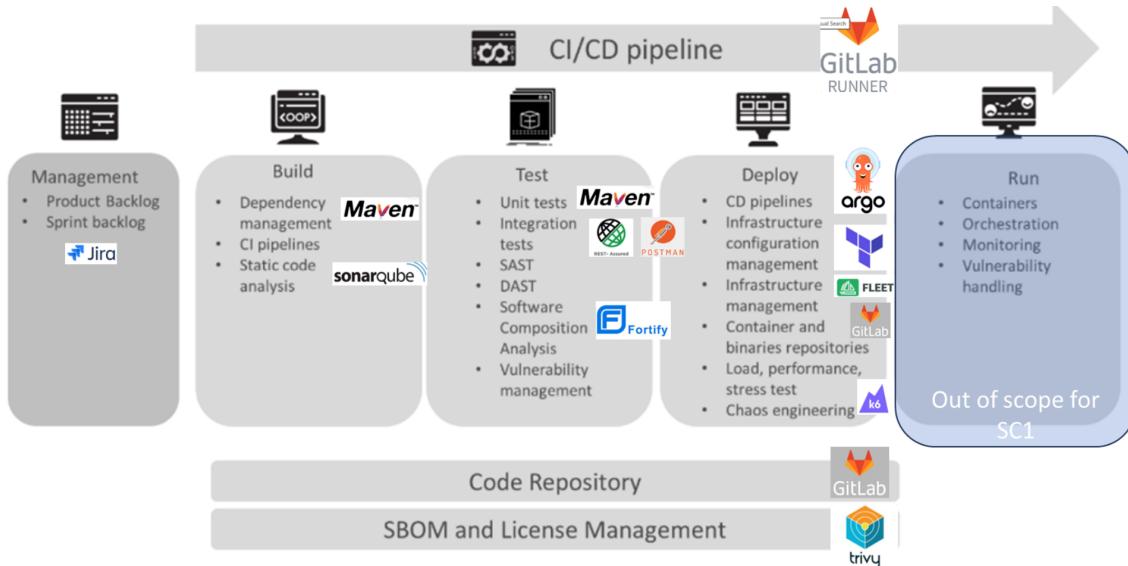
Reporting

i This section is only a placeholder for capabilities falling behind the scope of the MVP (December 2024) and will be completed at a later time.

DevSecOps Approach

This section gives an overview of the architecture for the DevSecOps tools and environments for Simpl-Open. The following diagram is taken over from Specific Contract 1 - Terms of reference

which provides an overall view of required DevSecOps approach completed with the relevant choices of tools/technologies in our implementation.



Overview

This architecture diagram below shows the main components of the DevSecOps toolchain used to comply with the above mentioned approach for the development of Simpl-Open.

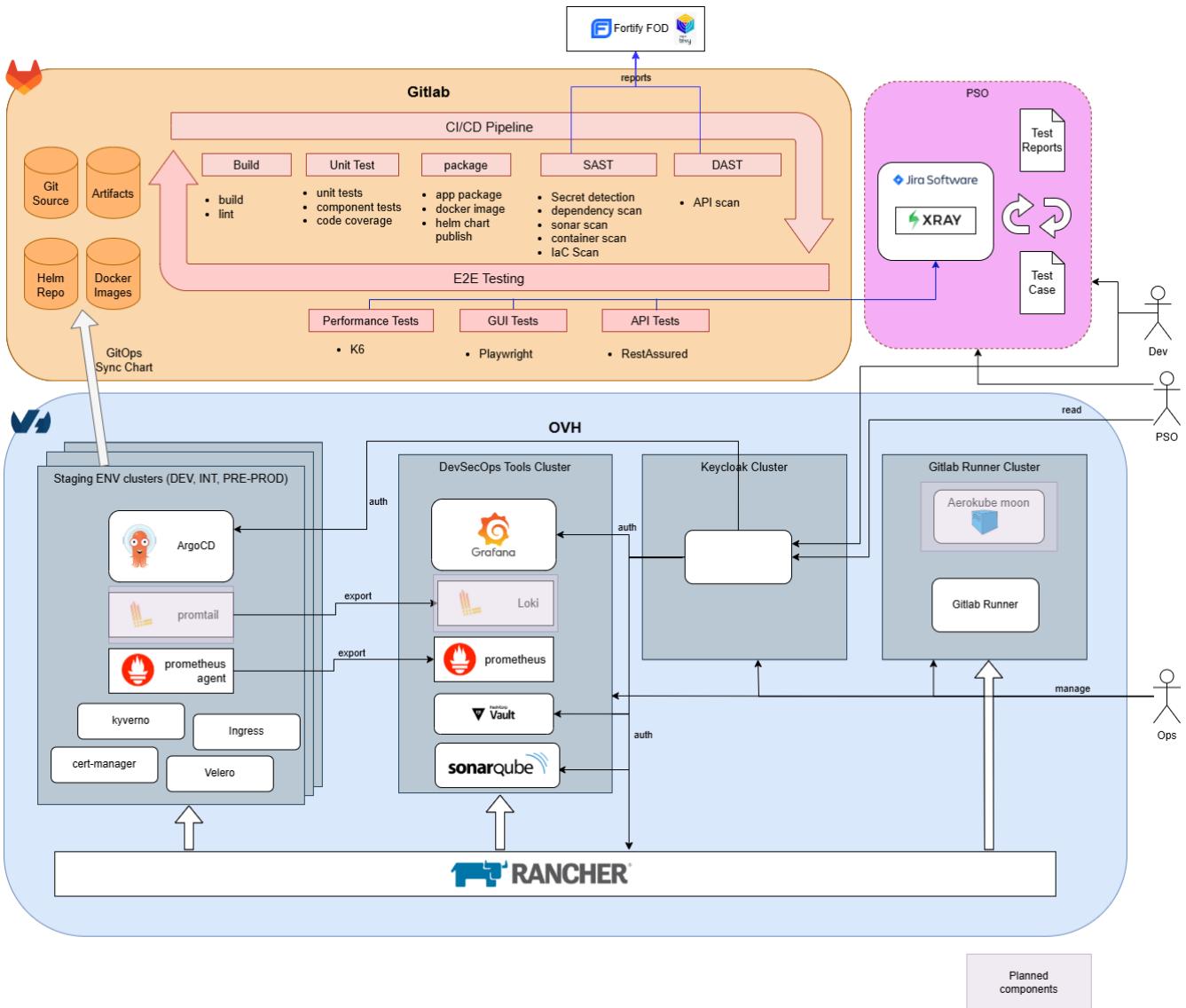
The central CI/CD pipeline to build and test the applications and components, as well as the code repositories, are on a GitLab instance on code.europe.eu.

OVH is used for the different Kubernetes clusters:

- Dedicated cluster with various namespaces for development;
- Dedicated cluster with various namespaces for integration;
- Dedicated cluster with various namespaces for end-to-end testing;
- Dedicated clusters for Keycloak identity management, Gitlab Runners, DevSecOps tools and DevSecOps tool testing (staging env for DevSecOps tools) .

For tickets, test cases and test reports are Jira is set up together with Xray for test management.

The diagram reflects the current status of the toolchain, with planned elements shown shaded.



Planning and Design of Clusters

The DevSecOps team provides the infrastructure resources to the development teams. It is responsible for the setup of the Kubernetes clusters on OVH and the management of those.

The cluster "Dev-components" is set up as the development environment. Each team gets isolated name space(s) to run their services.

To avoid vendor-lock-in, it is proposed to avoid using managed services on OVH like managed databases. This guarantees that the designed solution will also work on other cloud platforms without modifications needed.

The DevSecOps team manages the clusters via Rancher and provides access to the projects & namespaces to only the team members of the product streams.

The expected workload for each of the environments is estimated based on the input from the different development team and used for initial cluster sizing.

The table below shows the different stages and what they are used for.

Stage	Purpose	Data	Operations Level Agreement	Target User Group (responsible for deploy)	Release Management	Deployment Strategy (When are rel
Dev	<ul style="list-style-type: none"> The development stage 	<ul style="list-style-type: none"> Syntactic, m 	n/a	<ul style="list-style-type: none"> Developers of consortia members QA engineers for testing 	<ul style="list-style-type: none"> dev versions, e.g. 0.0.4-snapshot 	<ul style="list-style-type: none"> Daily and continuous builds from c and/or feature branch

	<p>is where new features and enhancements are developed and tested;</p> <ul style="list-style-type: none"> • Isolated Simpl product teams dev environment s (opt: connected to feature branches); • No involvement in commercial applications. 	<p>an ua lly cr ea te d da ta pe r de v te st to co nd uc t fu nc tio na l te sti ng of th e in div id ua l pr od uc t. (For fur th er de tail s se e Te st PI an)</p>			
Int	<ul style="list-style-type: none"> • The int stage is dedicated to comprehensive testing of features and fixes before they are promoted to release branch; • Used for functional testing of 	<ul style="list-style-type: none"> Sy nt he tic, m an ua lly cr ea te d da ta pe r de v te st to co nd uc t fu nc tio na l te 	n/a	<ul style="list-style-type: none"> • QA engineers for testing 	<ul style="list-style-type: none"> • release candidate versions • Manual sync for release candidate

	API endpoints and correct interaction between different components	string of individual product; may use same sanitized production data. These sanitization processes will ensure no sensitive information will be used.			
Pre-Prod	<ul style="list-style-type: none"> The pre-prod stage serves as a pre-production environment where the validation of the system's end-to-end workflows (End-to-end testing) 	<ul style="list-style-type: none"> Subsets of production data or representative datasets to simulate multiple environments. 	n/a	<ul style="list-style-type: none"> E2E testing team 	<ul style="list-style-type: none"> stable release versions <ul style="list-style-type: none"> manual sync for released versions Aimed to automate as much as feature maturity grows

<p>g) are taking place from user scenario point of view;</p> <ul style="list-style-type: none"> Also this stage is responsible for all non functional testing (load/stress testing, security testing by DAST) 	<p>e pr od uc tio n en vir on m en t ac cu rat ely;</p> <ul style="list-style-type: none"> Data will be saved or anony mously in ze d to protect sensitive information while preserving the integrity of the dataset. 			

Cluster Provisioning and Setup

Environment Onboarding Process

If a Dev-Team needs a new environment for any stage, they need to create an issue in the following Gitlab repo: Simpl/Operations/Environment-onboarding.

The DevSecOps team will create the environment with the default tool stack and grant access to it afterwards.

Process Description:

1. Create project in Rancher in the desired cluster (dev, int, ...);

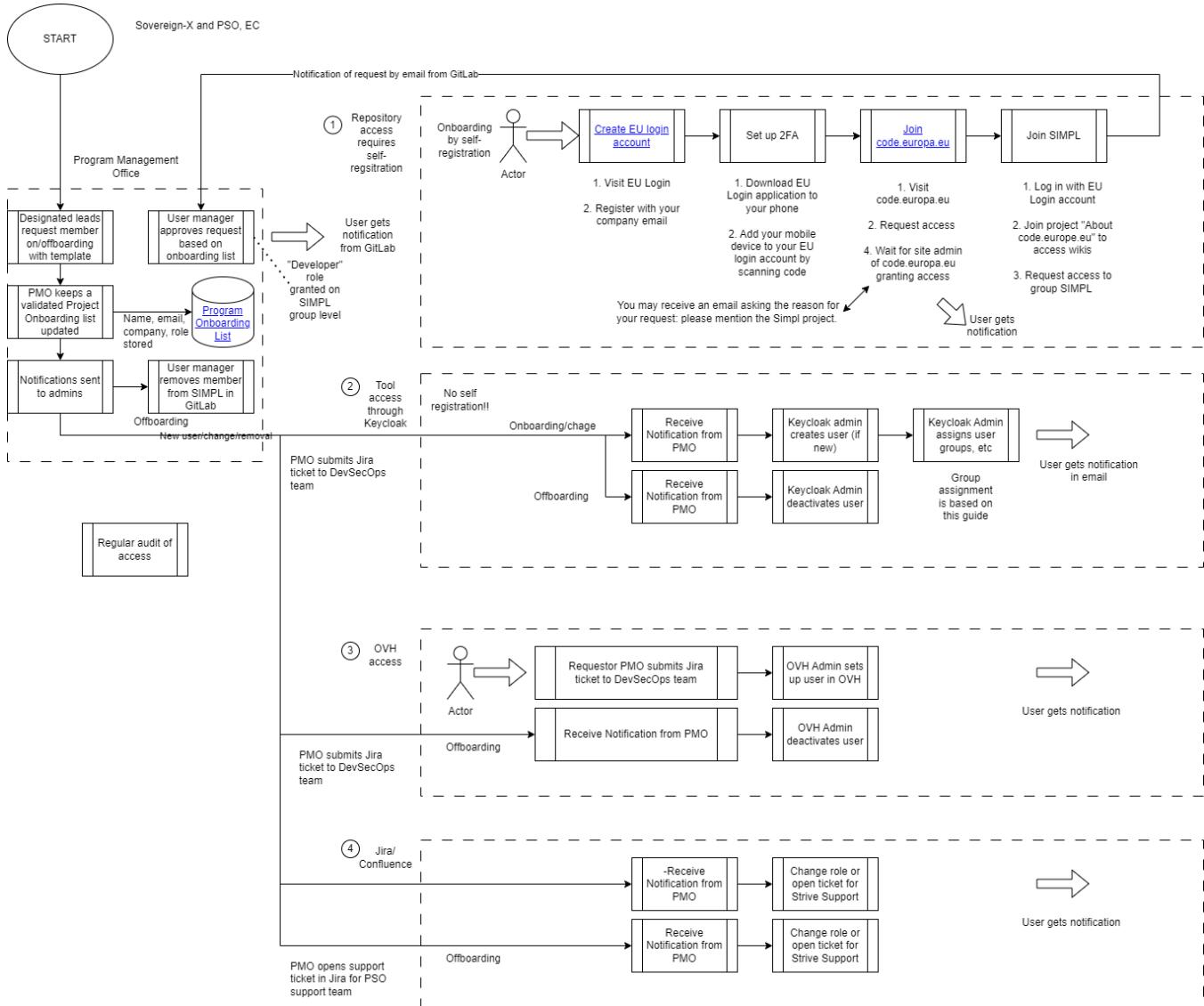
2. Deploy default toolstack (ingress etc.);
3. Create project in ArgoCD;
4. Grant access to Rancher project and ArgoCD project.

Security and Access Control

The following best practices are used to secure the environments.

Access Management

Access is granted / revoked based on the process described below.



Definition of Basic roles (as tracked in PMO master list)

User role name	Description
ADMIN	Role for the operation of the DevSecOps toolchain
PSO/ EC	Members of PSO accessing the DevSecOps toolchain for quality assurance purposes
DEVELOPER	Developers who will use the DevOps pipeline for development activities
LEAD DEVELOPER	Developer with code ownership and elevated security privileges
DEVELOPER OPS	Developers with elevated infrastructure privileges

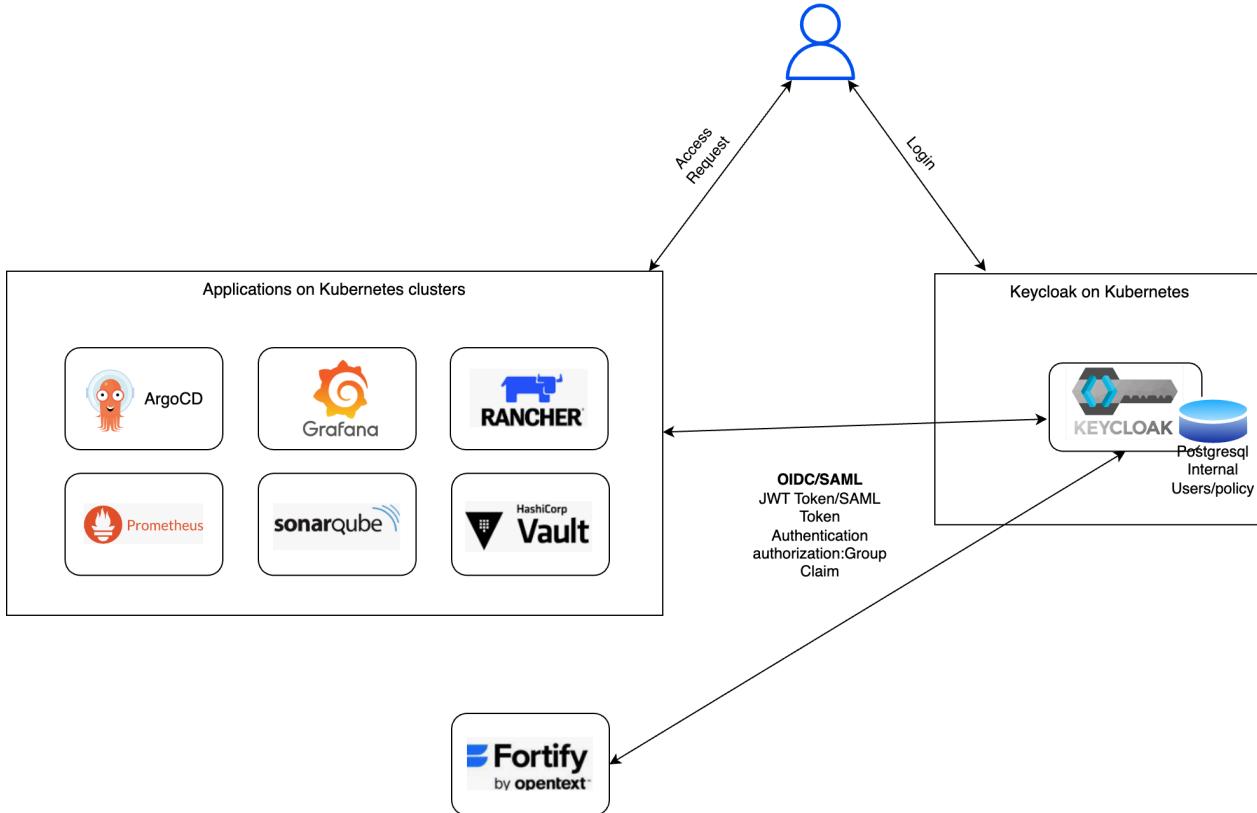
LEAD DEVELOPER OPS	Developers with code ownership and elevated security + infrastructure privileges
TESTER	Testers who will take part in the testing of developed code

Mapping of basic roles

This table shows the mapping between the basic roles and the internal roles within each tool.

Tool (with internal Roles)	code.europe.eu	Argo CD (ADMIN, DEV, READ-ONLY)	Rancher (ADMIN, Project Member, Read-Only)	Vault (ADMIN, DEV, DENY-ALL)	Fortify (Security Lead (Adm
Basic Role					
everyone on PMO master list (non-need for any specific DevSecOps role)	DEVELOPER on SIMPL group level (subject to self-registration)	n/a	n/a	n/a	n/a
ADMIN	MAINTAINER is set manually for DevSecOps team	ADMIN for ArgoCD	ADMIN	ADMIN	Security Lead (Admin)
PSO	DEVELOPER on SIMPL group level (subject to self-registration)	READ-ONLY	READ-ONLY	DENY-ALL	n/a
DEVELOPER	DEVELOPER on SIMPL group level (subject to self-registration)	Project Member (per project)	n/a	n/a	Developer (per application)
LEAD DEVELOPER	DEVELOPER on SIMPL group level CODEOWNER in repository (subject to self-registration)	Project Member (per project)	n/a	n/a	Lead Developer (includ (per application)
DeveloperOps	DEVELOPER on SIMPL group level (subject to self-registration)	Project Member (per project)	Project Member (per project)	DEV (per project)	Developer (per application)
TESTER	DEVELOPER on SIMPL group level (subject to self-registration)	READ-ONLY	n/a	n/a	Tester

Keycloak is used as central instance for the user management and providing the login mechanisms for the different tools:



Security Checks

Application security scans will be done by Fortify on Demand (FoD). The scope are the following:

- Static Application Security Test (SAST);
- Static Component Analysis (SCA);
- Container Scanning
- Dynamic Application Security Test (DAST).

SAST (Static Application Security Testing) is a method of testing the source code of an application for security vulnerabilities without executing the application. It's a type of white-box testing that analyzes the application's internal structures and logic on code level for flaws that might lead to security risks and vulnerabilities. Main advantages of using SAST:

- Enables detection and remediation of vulnerabilities early in the Software Development Lifecycle (SDLC), reducing costs and risks;
- Can analyze the entire codebase, providing a comprehensive security assessment;
- Helps the project to comply with security standards like OWASP, PCI DSS, and others

WHEN: SAST is performed on every commit/merge

WHAT: the source code is scanned

WHERE: during the development lifecycle

WHO: the pipeline is starting the analysis automatically

Fortify is integrated with the central component development pipeline which triggers Static Application Security Test (SAST) when new code is merged in the repository by the development or the integration teams. For feature branch a scan can be requested manually. Results of the scan is shown on the dashboard of Fortify. Developers can review the results of their components and handle identified vulnerabilities in the next version of the code. A quality gate set in Fortify must be met for the pipeline to merge the code to the main branch.

SCA (Software Composition Analysis) is a process used to identify and manage risks associated with the use of third-party and open-source software components in an application. It is a critical aspect of modern software development, as applications increasingly rely on external libraries and frameworks. Main advantages of using SCA:

- Protects the Simpl agent by identifying and addressing vulnerabilities in external components;
- Mitigates legal risks from improper use of open-source licenses;
- Automates tracking and reporting of third-party components.

WHEN: SCA is performed on every commit/merge

WHAT: *third party libraries are scanned*

WHERE: *during the development lifecycle*

WHO: *the pipeline is starting the analysis automatically*

SCA is integrated with the same approach as SAST using the debricked service of the Fortify online platform.

Container scanning refers to the process of analyzing and inspecting container images for security vulnerabilities, compliance issues, malware, and other potential risks before they are deployed in a production environment. Here's a brief overview:

- Purpose: The primary goal is to ensure that containers are free from known security vulnerabilities and adhere to organizational security policies. This helps in maintaining the integrity, confidentiality, and availability of applications running inside these containers.
- Components Scanned:
 - Base Images: Checking if the base images from which containers are built have any known vulnerabilities.
 - Dependencies: Examining all the software libraries and dependencies included within the image for vulnerabilities or outdated versions.
 - Configuration: Assessing the container's configuration files for potential security misconfigurations.

WHEN: *container scanning is performed in the development lifecycle at every branch and in every commit/merge*

WHAT: *the containers and images used are scanned*

WHERE: *during the development lifecycle*

WHO: *the pipeline is starting the analysis automatically*

DAST (Dynamic Application Security Testing) is a method used to identify security vulnerabilities in an application by analyzing it during runtime. It simulates attacks on a running application, typically from an external perspective, to uncover vulnerabilities that can be exploited in real-world scenarios. Main advantages of using DAST:

- Identifies vulnerabilities of the Simpl software as an attacker would exploit them
- Finds issues related to application logic, runtime behavior, and server configuration.

WHEN: *at the end of the development lifecycle, after e2e testing.*

WHAT: *DAST is performed on the Agents.*

WHERE: *DAST is performed in the pre-prod environment*

WHO: *the end2end team is responsible for configuring and running the scans*

DAST is implemented using the webinspect service of the Fortify online platform.

While SAST, SCA and Container Scanning are integrated in the component pipeline and used on component level, DAST will be triggered manually after deployment of the integrated Simpl agent in the pre-prod environment and the completion of preparational steps. This testing type may require the runtime of up to 2-3 days by Fortify. Once the testing is complete the Fortify dashboard will provide an overview of the results. Similarly to SAST and SCA developers can review the identified issues and act on them as necessary.

Kubernetes best practices

In order to set up a flexible and scalable environment for managing our containerized applications, Kubernetes has been identified as the best fit technology. Primary reasons for the choice are:

- vendor neutral platform;
- support of microservices infrastructure;
- autoscaling capabilities to handle growing and fluctuating workloads;
- support for DevSecOps;
- support of multi-tenant environments;
- high availability.

Main features of Kubernetes:

- **Network Policies:** Defined network policies [network policies](#) to control traffic within the cluster; OVH provides a default set of policies. Inside the Kubernetes cluster we use ingress and egress isolation for pod level according to the specific needs of the cluster.

- **Secret Management:** Usage of Kubernetes Secrets and/or Vault to store sensitive data securely. Secrets used in the pipeline are stored in an external tool like Vault;
- **Role-Based Access Control (RBAC):** Implemented RBAC to manage access to cluster resources based on user roles; Keycloak groups are mapped to Rancher projects to ensure proper isolation of namespaces. User roles are mapped to Keycloak groups/roles.
- **Service Accounts:** Usage of Service Accounts to authenticate and authorize pods;
- **Image Scanning:** The integrity and security of container images is verified before deployment in the pipeline; This is done by using Trivy/Fortify triggered by the pipeline.
- **Regular Updates and Patching:** To keep the Kubernetes distribution and components up-to-date with the latest security patches regular updates are done. Since Kubernetes is a managed service, updates are made available by OVH. Admins regularly check update options and decide to stay with current version or update.

Continuous Deployment and GitOps

This section outlines the implementation of continuous deployment (CD) and GitOps in Simpl-Open using GitLab CI/CD, Helm Charts, Argo CD, and multiple environments.

The goal is to automate the release management process, ensuring consistent and reliable deployments across various environments.

Architecture

The architecture consists of:

1. **GitLab:** The source code is managed on the GitLab instance at code.europa.eu. There also the CI/CD pipeline is used;
2. **Helm Charts:** Package managers for Kubernetes applications;
3. **Argo CD:** A continuous deployment tool for automating the application release process for the development, integration and pre-prod environments
4. **Fleet Management:** A K8 concept and tool to centrally manage DevSecOps tools, agents, components on the every clusters in the landscape.
5. **Multiple Environments:** The deployment is done into multiple environments.

GitFlow

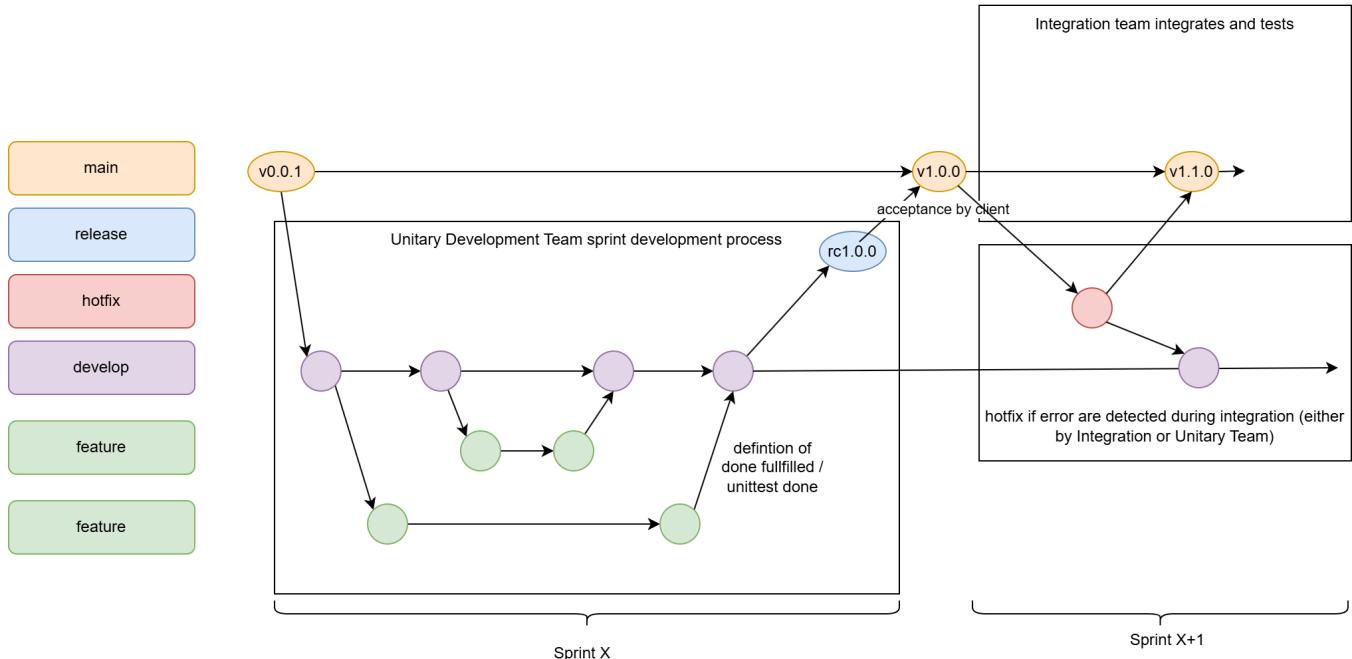
The project uses GitFlow, as a branching strategy for Git repositories designed to streamline collaboration and manage releases in software projects. Giflow has become widely adopted in software development workflows, especially for projects with regular release cycles.

The advantages of the Gitflow approach:

- Clearly defines branches for development, features, releases, and hotfixes;
- Makes it easier for teams to work on different features or issues simultaneously;
- Facilitates managing multiple releases and hotfixes.

Artefacts should be versioned according to the Semantic Versioning Concept.

The Gitflow approach in Simpl is depicted in the following diagram:



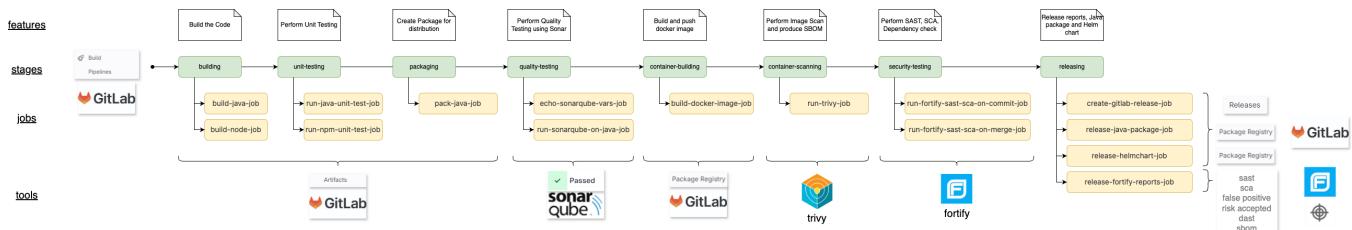
Explanation for the branches:

1. **main**: The main branch, which represents the production-ready code;
2. **develop**: The development branch, where new features are developed and tested;
3. **feature/***: Feature branches for specific tasks or fixes;
4. **release**: Release branch for release candidates;
5. **hotfix**: Forked from tags of the main branch used for urgent fixes.

In Gitlab (code.europa.eu) the main and develop branches are set up as protected. Merge to main can be initiated from develop, release and hotfix. Developers remove release and hotfix following the merge of the updated code to main. Develop is only allowed to merge code from feature/* (for instance feature/SIMPL-1234). Developers remove the feature branch after merging to develop.

CI/CD Pipeline

The pipeline is implemented using GitLab CI/CD. The are multiple steps included to ensure proper testing and security before the deployment:



Pipeline features:

- **Building the Code**: Compile the source code into executable binaries or artifacts;
- **Perform Unit Testing**: Execute automated tests to verify individual components of the codebase for correctness;
- **Create package for distribution**: Bundle the application into distributable formats like jar and publish to the GitLab artifacts;
- **Perform Quality Testing with Sonar**: Perform static code analysis to identify code quality issues and technical debt;
- **Build and Push Docker Image**: Create a Docker image from the application and push it to the GitLab Container registry;
- **Perform Image Scan and produce SBOM**: Scan the Docker image for vulnerabilities and compliance issues using Trivy, list all dependencies created with Trivy and Fortify;;
- **Perform SAST (Static Application Security Testing)/SCA (Software Composition Analysis)** : Analyse the source code for security vulnerabilities without executing the code using Fortify;
- **SCA (Software Composition Analysis)**: Identify and analyze open-source components (dependencies scanning) for known vulnerabilities using Trivy for image scanning and Fortify for dependencies;
- **Release reports, Java package and Helm Chart**: Update the version, package and release a Helm chart for Kubernetes deployments in the GitLab Package Registry;

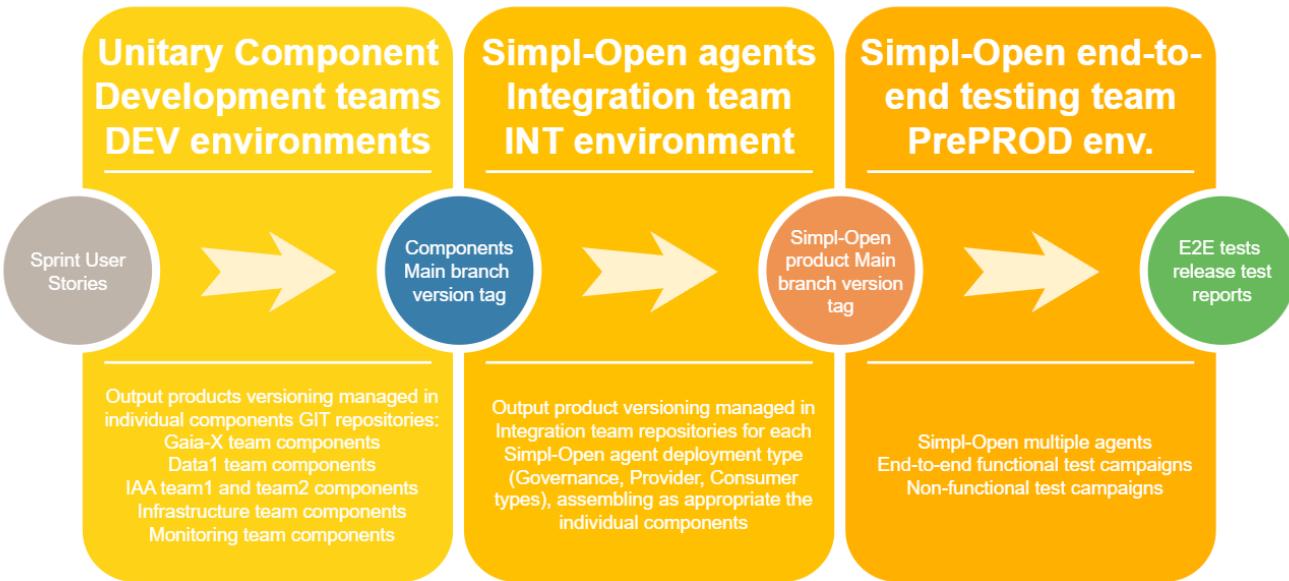
Pipeline runs can be tracked on the UI of Gitlab. Issues are indicated by the progress diagrams on the UI, details are provided by Gitlab based on the logs of the failing jobs.

Release Management

The release management process will be carried out on two distinct level (App of Apps concept):

- Unitary Component development
- Agent Components (integration and pre-prod)

The following diagram shows the overall process:



As shown in the diagram there are multiple stages with different environments:

1. **Development Environment:** The development environment is where new features and enhancements are developed and tested on component level;
2. **Integration Environment:** The integration environment is dedicated to integration activities and integration testing before they are promoted to pre-prod;
3. **Pre-Prod Environment:** The pre-prod environment serves as an environment where features are integrated and tested together as a cohesive unit, end to end. This is where load testing is taking place.

A Production environment is not planned for Simpl-Open, just for Simpl-Labs and Simple-Live.

As an overall concept, the release management process is automated using GitLab CI/CD and Argo CD:

1. **Prepare a Release:** A new release version is created following the GitFlow approach on GitLab;
2. **Build and Test:** The extended pipeline stages run to validate the release quality, including E2E testing and extensive security scanning;
3. **Deploy:** Argo CD deploys the release to the target environment;
4. **Verify:** Verify the deployment by running tests and monitoring application logs.

Helm Charts

Helm Charts are used to manage components and Kubernetes applications. Similarly Helm Charts define the application on Agent level.

1. **Chart Management:** Helm Charts are managed using GitLab CI/CD, allowing for automated updates and versioning;
2. **Deployment:** Helm Charts are deployed to the target environment using Argo CD.

Component development teams release their component by a Helm Chart. By the application of the App of Apps concept, on Agent level (App) we define and manage the individual components (Apps). This is ensured by the configuration of Helmcharts in a hierarchical manner (Agent level configuration overwrites component level configuration).

Benefits of the App of Apps Concept

- **Scalability:** Simplifies managing a large number of components in complex agents;
- **Centralized Management:** Enables a single point of control for all components;
- **Flexibility:** Supports managing components across multiple environments or clusters;
- **Modularity:** Each component remains independently manageable, facilitating updates and troubleshooting;
- **GitOps Alignment:** Integrates seamlessly with GitOps workflows for declarative management.

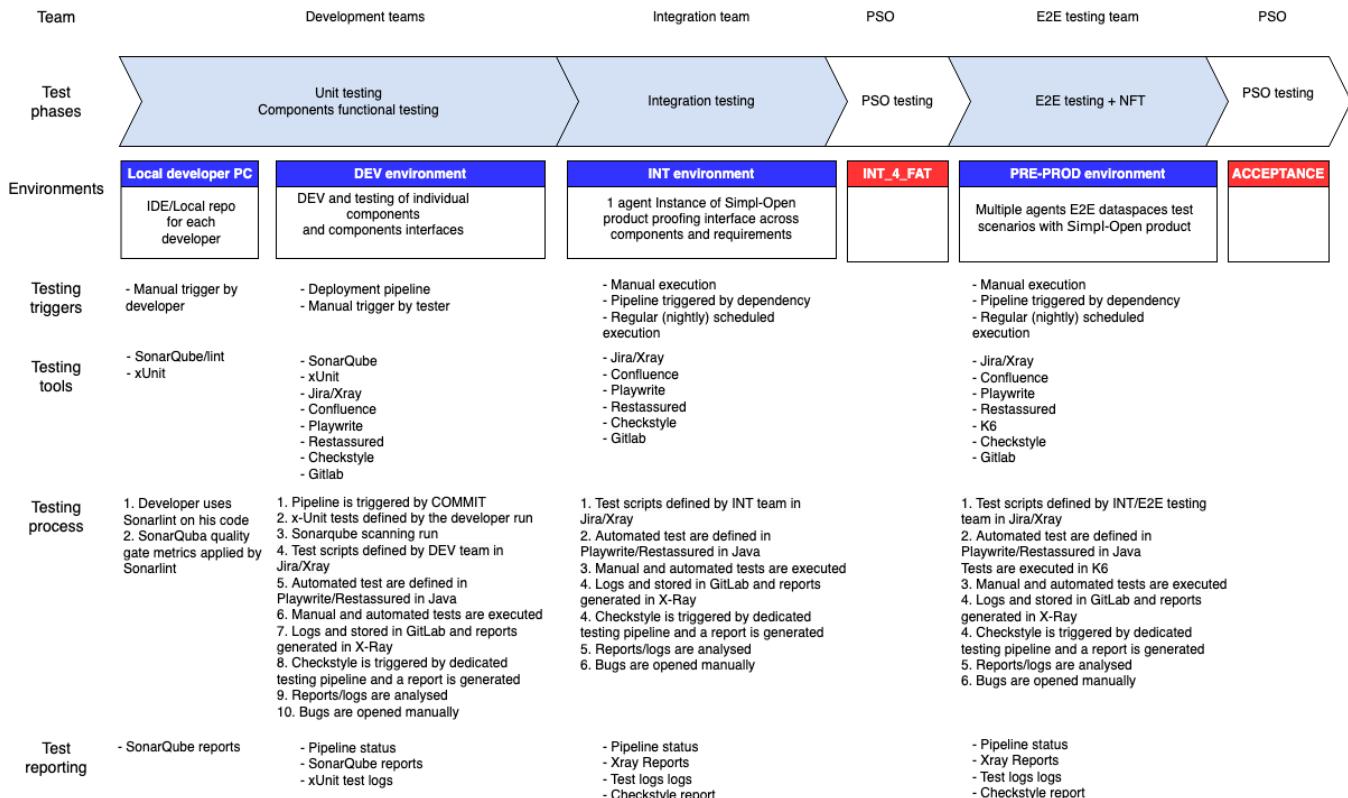
Argo CD

Argo CD is used to automate the deployment process. It is deployed in each environment to support the release process.

1. **Application Definition:** Define applications in Argo CD's configuration file (application.yaml);
2. **Source Code Management:** Argo CD manages automatic deployments. Deployments happen based on the following triggers: by repo source code change, package registry change and manually. Dev team have the freedom to configure this for their components.
3. **Deployment Strategies:** Choose deployment strategies for each application, such as rolling updates or blue-green deployments.

Testing

The testing process is separated into different phases which are shown in the diagram below. The full test process is described in the testing document.



Monitoring and Logging

Monitoring and logging tools are used to track application performance and detect issues:

1. **Prometheus:** A monitoring tool that collects metrics from the environments/ tools and stores them in a timeseries database; Deployed as agent on all clusters.
2. **Grafana:** A visualization tool that displays dashboards based on Prometheus data. Centrally deployed on the DevSecOps-tools cluster as a central instance.
3. **Promtail:** will be deployed on the clusters as an the agent to discover and gather logs.
4. **Loki:** will be deployed to centrally aggregate and manage collected logs.

Prometheus agents will be deployed to all Kubernetes clusters and connect to the central Prometheus instance (on the DevSecOps tool server) to consolidate metrics. Grafana, deployed on the DevSecOps cluster will use the data available for the Graphana central instance to visualize data.

Metrics data is retained for 1 months.

Currently Email based alerting mechanism is set up in Grafana to notify operators for events configured in the tool.

Further extension of the infrastructure will be done with the deployment of Promtail and Loki.

Backup and restore

On the Kubernetes clusters Velero is used for backing up and restoring persistent volumes . The tool is deployed on the following clusters:

Cluster	Backup policy
dev-components	WEEKLY, DAILY
devint-agents	WEEKLY, DAILY
devsecops-keycloak	NONE
devsecops-runners	WEEKLY
devsecops-tools	WEEKLY, DAILY
devsecops-toolstest	NONE
preprod-agents	WEEKLY, DAILY

Backup is done for the specific Namespaces configured for the process.

Data restoration is carried out with the same tool.

Deprovision of Environments

Deprovisioning refers to the process of removing or deleting resources that were initially provisioned. In this context, it involves removing the application from the Kubernetes clusters across different environments and deleting the infrastructure managed by Terraform.

For Simpl there are two steps for the deprovisioning: One for the application and the second for the overall infrastructure to shut down Simpl completely.

Step 1: Application Deprovisioning

The application deployed via ArgoCD can be removed by deleting the relevant application resource. This can be achieved using the ArgoCD CLI or the ArgoCD API.

Please note that this process has to repeated for every instance of the application running on different Kubernetes clusters within each environment.

Step 2: Infrastructure Deprovisioning

Terraform maintains an up-to-date state file that reflects the current state of the infrastructure. To deprovision the infrastructure, it needs to be destroyed using Terraform.

Terraform offers the `destroy` command to delete the infrastructure which was deployed by Terraform. The command compares the state file to the current infrastructure and removes everything that exists.

The command has to be executed for each of the environments separately. This can be done exclusively be the admins of the DevSecOps team for each environment.

If the infrastructure including the DevSecOps-Tools-Cluster is destroyed, also the managed applications like Keycloak, Vault etc. are deleted. Any necessary data must be backed up before this process is started.

Annexes

Annex 1 - Glossary of Terms

This section provides a comprehensive list of terms and definitions essential for understanding Simpl's terminology.

Term	Definition
(Simpl-Open) Agent	An agent is a deployed instance of Simpl-Open that serves as a local gateway towards the data space services.
Access Policy	Access policies are rules and conditions prepositioned as to how the provider wants to control the service. The access policy allows you to configure operations, permissions, access rules, and selection rules that restricts/allow certain access to a service.
Applicant	An "applicant" refers to an entity that submits an application or request to join a data space. This term is typically used in contexts where a selection or approval process is in place. Applicants are at the initial stage of engagement with the data space, seeking approval or acceptance which is based on the criteria and procedures set by the Data Space Governance Authority
Application Catalogue	Application providers publish their service descriptions to the application catalogue which can be queried and discovered by consumers.
Authentication provider federation	An Authentication Provider Federation is a collaborative arrangement between multiple authentication providers that enables users to authenticate seamlessly across different systems, organisations, or services using a unified and interoperable framework. This federation facilitates the sharing of authentication credentials and ensures a consistent and secure authentication experience across various platforms.
Capability	Capability refers to the inherent power, capacity, or potential of the data space to perform specific tasks or achieve certain outcomes. This could involve the ability to integrate, analyse, and share data across different systems, the capacity to handle large volumes of data, or the potential to support complex data-driven processes.
Catalogue	The catalogues enable the discovery of services within the data space.
Consent	'Consent' of the data subject means any freely given, specific, informed and unambiguous indication of the data subject's wishes by which he or she, by a statement or by a clear affirmative action, signifies agreement to the processing of personal data relating to him or her; (GDPR Art. 4(11)) ¹
Contract Offering	A contract offering is a promise made by the Provider to the Consumer (the offeree) to provide a service under certain conditions, quality KPI for a certain agreed price. Such an offering could, for example, consist of: <ul style="list-style-type: none">• SLA Agreement that guarantees the quality of service;• A License agreement on which terms and conditions the service can be consumed;• A Billing schema on how the service is invoiced and the pricing.
CRUD interface	A set of backend API and fronted UIs that realise the Create, Read, Update and Delete operations on a specific entity
Data access policy	A data policy defined by the data rights holder for the access of their shared data in a data space. ¹ Explanatory text: A data access policy that provides operational guidance to a data provider for deciding whether to process or reject a request for providing access to specific data. Data access policies are created and maintained by the data rights holders.
Data Catalogue	Data providers publish their service descriptions to the data catalogue which can be queried and discovered by consumers.
Data policy	A set of rules, working instructions, preferences and other guidance to ensure that data is obtained, stored, retrieved, and manipulated consistently with the standards set by the governance framework or data rights holders. ¹ Explanatory text: Data policies govern aspects of data management within or between data spaces, such as access, usage, security, and hosting.
Data space	A distributed system defined by a governance framework that enables secure and trustworthy data, application, and infrastructure transactions between participants while supporting trust and sovereignty. A data space is implemented by one or more infrastructures and enables one or more use cases. ¹
Data space	A party that has committed to the governance framework of a particular data space and may have one or more roles in it.

participant	
Data usage contract	An agreement between a data rights holder or data provider, and a data recipient specifying the terms and conditions of a data exchange, which may refer to specific data policies. ¹ Explanatory text: The term data usage contract can also be used for usage contracts related to data services.
Digital signer role	Digital signer role is an extended role (in Tier 1) that identifies the end users who are able to digitally sign in on behalf of a participant.
End user	An end user is either: <ul style="list-style-type: none"> • a real person (human actor) that interacts with the Simpl-Open agent mostly through the UI; • an IT system (machine actor) that interacts with the Simpl-Open agent through APIs.
Feature	A feature is a distinctive attribute or characteristic of the data space platform that contributes to its overall functionality. Features are specific aspects or components designed to provide particular services or tools to users, enhancing the user experience or enabling specific operations.
Functionality	Functionality refers to the range of operations and tasks that the data space is designed or expected to perform. It encompasses the practical uses and applications of the data space, ensuring that it serves its intended purpose effectively. Functionality includes both the basic operations (e.g., data storage, data querying) and more complex operations (e.g., automated data processing, real-time data analytics) that the data space may support.
Identity Attribute	Identity Attributes are the characteristics or values of a Simpl-Open agent (e.g. "Data Provider Publisher" or "Basic Access Level") involved in an access event. Identity Attributes are used to enforce ABAC (attributes-based access control) in the agent-to-agent communication (tier 2).
Identity Authority (component)	The identity authority is a potential module of the Governance Authority and is responsible for establishing, maintaining, and verifying the identity of participants within that data space.
Identity provider federation	An Identity Provider Federation is a collaborative arrangement between multiple identity providers that allows for the sharing and mutual recognition of user identities across different organisations, domains, or services. This federation enables users to use a single set of credentials to access multiple systems or services, thereby simplifying the authentication process and enhancing user convenience while maintaining security and trust.
Infrastructure Catalogue	Infrastructure providers publish their service descriptions to the infrastructure catalogue which can be queried and discovered by consumers.
Onboarding request (request for onboarding)	An onboarding request is a formal application submitted by an entity (the applicant) to join a data space.
Ontology	An ontology is a formalised and structured knowledge within a specific domain. It includes the concepts (also called the vocabulary) as well as the relationships between the concepts. For the use of the automatic validation, the ontology should be provided in an RDF Schema (RDFS) or Web Ontology Language (OWL) specifications.
Participant	A "participant" is an entity that has successfully passed the approval or acceptance in the application process and is now actively involved or engaged in the services offered by the applied data space. Participants are either governance authority, data providers, application providers, infrastructure providers or consumers.
Policy	Policies define rules that Providers want to enforce in order to control the access and usage of their resources.
Quality Rules	Quality rules can be defined as a set of guidelines, standards, or criteria used to assess and ensure the quality of a self-description. Explanatory text: <i>Data quality rules are a formal and structured definition of required quality of the resource description. These rules include assessing completeness, consistency, correctness, and other quality aspects, e.g., defined in ISO 25012. The checks go beyond semantic conformance, such as identifying inconsistencies between data instances, detecting redundant information, or verifying data integrity. The data quality rules can be classified into mandatory rules and recommended rules. Mandatory rules need to be fulfilled completely for all instances, e.g., the creation date should never be after the update date. For the recommended rule it is possible to define a threshold on how many instances need to fulfil the rule, e.g., it is preferable to provide 4 keywords.</i>

Representative	A "representative" refers to the human end user that takes actions on behalf of a participant, such as the: consumer, provider, or governance authority. The representative is special type of end user that represents the participant in the data space.
Resource description	Resource description refers to the metadata records that describe various resources, allowing identification, searchability, access management, preservation, and prediction of resource behaviour. These resources can include data, applications, or infrastructure.
Role Attribute	A role attribute refers to a specific property or characteristic assigned to a user role within a system. A role defines a set of permissions or access rights that a user has within the software, and role attributes provide additional details or constraints that customise or refine these roles.
Security attribute provider federation	A Security Attribute Provider Federation is a collaborative network of multiple security attribute providers that enables the sharing and management of security-related attributes across different systems, organisations, or services. This federation facilitates a unified approach to attribute management, enhancing security, interoperability, and user experience in various environments.
Self-description (Technical term for 'resource description')	<p>A metadata record that providers use to describe themselves, their data product offerings, and the resources and services their data products are composed of.¹</p> <p>Explanatory text:</p> <p>The representation of these metadata must be comprehensible for the data space enabling services that manage self-descriptions through their lifecycle, as well as for data users and any software that assists them.</p>
Semantic validation	<p>Semantic validation involves verifying the data types and some constraints (like mandatory, cardinality, consistent with a vocabulary) for the different properties in the self-description.</p> <p>Example: Ensuring that a date of birth field contains a valid date that is in the past and not in the future.</p>
Simpl-Open	Open-Source Smart Middleware Platform for Cloud-to-Edge Federations and Data Spaces.
Syntax validation	<p>Syntax validation refers to the process of checking if the resource description conforms to the defined syntactical rules and formats. It ensures that the input is correctly structured according to the required grammar and patterns.</p> <p>Example: Ensuring that a JSON file is properly formatted with matching brackets and correct key-value pairs.</p>
Usage Contract	<p>A usage contract is the signed agreement between a provider and a consumer, that stipulates the type of services a provider offers to the consumer and includes the conditions and policies the service, provider, and consumer needs to adhere to.</p> <p>The usage contract is formed and signed by both the Provider & Consumer during the contract negotiation process. The usage contract is afterwards used as an immutable credential by the Provider and the Consumer.</p>
Usage Policy	<p>Usage Policies are policies defined by the provider for the usage of their resource in a data space. The policies regulate the permissible actions and behaviors related to the utilisation of the accessed data/application/infrastructure.</p> <p>A usage policy defines what actions can be undertaken on a resource by what consumers and under what constraints.</p>
Vocabulary Provider (component)	The vocabulary provider defines the ontologies and vocabularies that are standardised in the data space.

¹<https://dssc.eu/space/Glossary/176553985/DSSC+Glossary+%7C+Version+2.0+%7C+September+2023>

Annex 2 - Mapping between functional requirements and components

While L2 requirements are mapped to functional requirements through the use of components in Jira, the below table provides an extract from this mapping.

Requirement ID	Summary	Component/s
SIMPL-6122	Data Visualization	Data Transfer, Infrastructure Management

SIMPL-6109	Access policy enforcement	EDC Connector
SIMPL-6100	Requesting an infrastructure resource	Infrastructure Management
SIMPL-5396	Request a data resource	Data Space Connector, Data Transfer
SIMPL-4889	Publishing self-description	Federated Catalogue, Resource Offering Editor
SIMPL-4497	Returning query results	Federated Catalogue, Search
SIMPL-4495	Filter search result based on access policy	Federated Catalogue, Search
SIMPL-4422	Monitoring Simpl-Open agent infrastructure technical logs	Observability
SIMPL-4417	Automated deployment of Simpl-Open pre-configured monitoring dashboard	Observability
SIMPL-3995	Define the onboarding process documentation	Onboarding
SIMPL-3886	Monitoring Simpl business logs	Observability
SIMPL-3381	The Usage Contract Agreement stored in machine readable format	Contract Management
SIMPL-3370	Usage contract signature	Contract Management
SIMPL-3363	Contract negotiation protocol	Contract Management, Data Space Connector
SIMPL-2949	Simpl shall log all business actions in the central logs repository	Observability
SIMPL-2946	Log Simpl agent infrastructure metrics	Observability
SIMPL-2945	Store technical logs of the infrastructure on which Simpl-Open is deployed in a log repository	Observability
SIMPL-2941	Simpl shall store technical logs of agent (software) components in a log repository	Observability
SIMPL-2921	Monitoring Simpl-Open agent infrastructure metrics	Observability
SIMPL-2916	Pre-configured monitoring dashboard	Observability
SIMPL-1789	Integration with Cloud APIs through Crossplane	Infrastructure Management
SIMPL-1784	Data sharing	Data Space Connector, Data Transfer
SIMPL-1748	End user authentication process - api	IAA
SIMPL-1745	Roles management operations	IAA
SIMPL-1739	Triggering Mechanism	Data Space Connector, Federated Catalogue, Infrastructure Management

SIMPL-1738	Infrastructure Specific Features	Infrastructure Management
SIMPL-1734	Advance search - Search parameters compliant with constraints and vocabularies	Schema Management, Vocabulary Management
SIMPL-1720	Search Results Limitation	Search
SIMPL-1719	Advanced Search	Federated Catalogue, Search
SIMPL-1715	Access policy definition	Resource Offering Editor
SIMPL-1705	Uploading self-description	Federated Catalogue, Resource Offering Editor
SIMPL-1704	Creating self-description	Resource Offering Editor
SIMPL-1699	Syntax Validation	Federated Catalogue, Resource Offering Editor, Schema Management
SIMPL-1698	Validation of a resource description - feedback to the provider	Federated Catalogue, Resource Offering Editor, Schema Management
SIMPL-1696	Mandatory quality rules	Federated Catalogue
SIMPL-1689	Users and roles configuration	Onboarding, IAA
SIMPL-1687	Credentials installation and review - status and information	Onboarding, IAA
SIMPL-1686	Credentials installation and review - services	Onboarding, IAA
SIMPL-1684	Credential request - tracking by participant	Onboarding, IAA
SIMPL-1683	Credential creation	Onboarding, IAA
SIMPL-1682	Create credential request	Onboarding, IAA
SIMPL-1681	Attribute selection	Onboarding, IAA
SIMPL-1679	Onboarding requests - rejection support	Onboarding, IAA
SIMPL-1677	Onboarding requests - manual approval support	Onboarding, IAA
SIMPL-1676	Onboarding requests - verification support	Onboarding, IAA
SIMPL-1674	Onboarding request - tracking by applicant	Onboarding, IAA
SIMPL-1673	Register onboarding application	Onboarding
SIMPL-1672	View the onboarding process documentation and initiate the onboarding	Onboarding
SIMPL-1654	Participants detail operations - services	Onboarding, IAA

SIMPL-1653	Participants detail operations - workflow	Onboarding, IAA
SIMPL-1650	Participants list	IAA
SIMPL-1616	Authentication between participant agents	IAA
SIMPL-1615	Ensure ABAC compliance	IAA
SIMPL-1614	Controlling communication between participants	IAA
SIMPL-1613	Tier 2 attributes management - services	Onboarding, IAA
SIMPL-1612	Tier 2 attributes management - workflow	IAA
SIMPL-514	Assign Contract Template	Contract Management, Resource Offering Editor
SIMPL-503	Access policy publication	Resource Offering Editor
SIMPL-500	Semantic Validation	Federated Catalogue, Schema Management, Vocabulary Management
SIMPL-469	Quick Search	Federated Catalogue, Search
SIMPL-415	Enforce usage policies	Contract Management, Data Space Connector
SIMPL-409	Assign usage policy	Resource Offering Editor
SIMPL-402	Create usage policy	Resource Offering Editor

Annex 3 - Non-Functional Requirements

This section provides an initial list of requirements stemming from the Simpl-Open tender specifications and which could drive elicitation of Simpl-Open non-functional requirements.

Req ID	CFN.001
Short Title	Interoperability with existing solutions
Description	Simpl shall provide the mechanisms to interoperate with existing solutions, such as the EOSC middleware functionalities, CEF BBs, specific data space solutions, and other artefacts.
Priority	M
Related architectural component(s)	Multiple
Source	Preparatory Study "Final Business Requirements" Framework Contract
Baseline technologies	API
Baseline component	New
Req ID	CFN.006
Short Title	Business continuity and disaster recovery
Description	Simpl shall offer the appropriate mechanisms to ensure its business continuity. Based on the monitored

	values of the status of the services where it is deployed, as well as the health of its components at all times. Simpl shall take the appropriate actions to provide (semi-automatic) self-healing mechanisms. In the event that automatic self-healing mechanisms cannot be launched, an alarm shall be raised.
Priority	M
Related architectural component(s)	Orchestration
Source	Preparatory Study "Final Business Requirements"
Baseline technologies	AI techniques to detect anomalies and concept drifts. Decision Support Systems for the decision-making on the self-healing mechanisms are to be applied.
Baseline component	Orchestrator (Kubernetes, OpenNebula or similar)

Req ID	CFN.009
Short Title	Infrastructure agnostic
Description	Simpl shall be deployable on multiple types of deployment models (public, private, hybrid) and providers (e.g., OVH, hyperscalers, other providers).
Priority	M
Related architectural component(s)	Distributed execution – Infrastructure management
Source	Preparatory Study "Final Business Requirements" Framework Contract
Baseline technologies	IaC (e.g., Ansible, Terraform)
Baseline component	-

Req ID	CFN.020
Short Title	Encryption mechanisms
Description	Simpl shall encrypt data at rest and data in motion following state of the art encryption protocols.
Priority	M
Related architectural component(s)	Security – Encryption
Source	Preparatory study, "Final Business Requirements"
Baseline technologies	-
Baseline component	-

Annex 4 - Architecture Patterns

Microservices Integration Patterns

The following diagram describes at a high level how the different services of Simpl-Open integrate with each other within an agent but also between agents.

This architecture presents a combination of both API-driven and message-driven patterns.

To fulfill the specific requirements of Simpl-Open to have 2 tiers IAA, 2 distinct API Gateways co-exist to handle cross-cutting concerns:

1. The Tier I API Gateway is used to integrate and secure communications with the Tier I security provider from the participant and also with the User Interfaces (microfrontends) which are part of the agent itself;
2. The Tier II API Gateway is used to integrate and secure communications with agents from other participants (or Governance Authority).

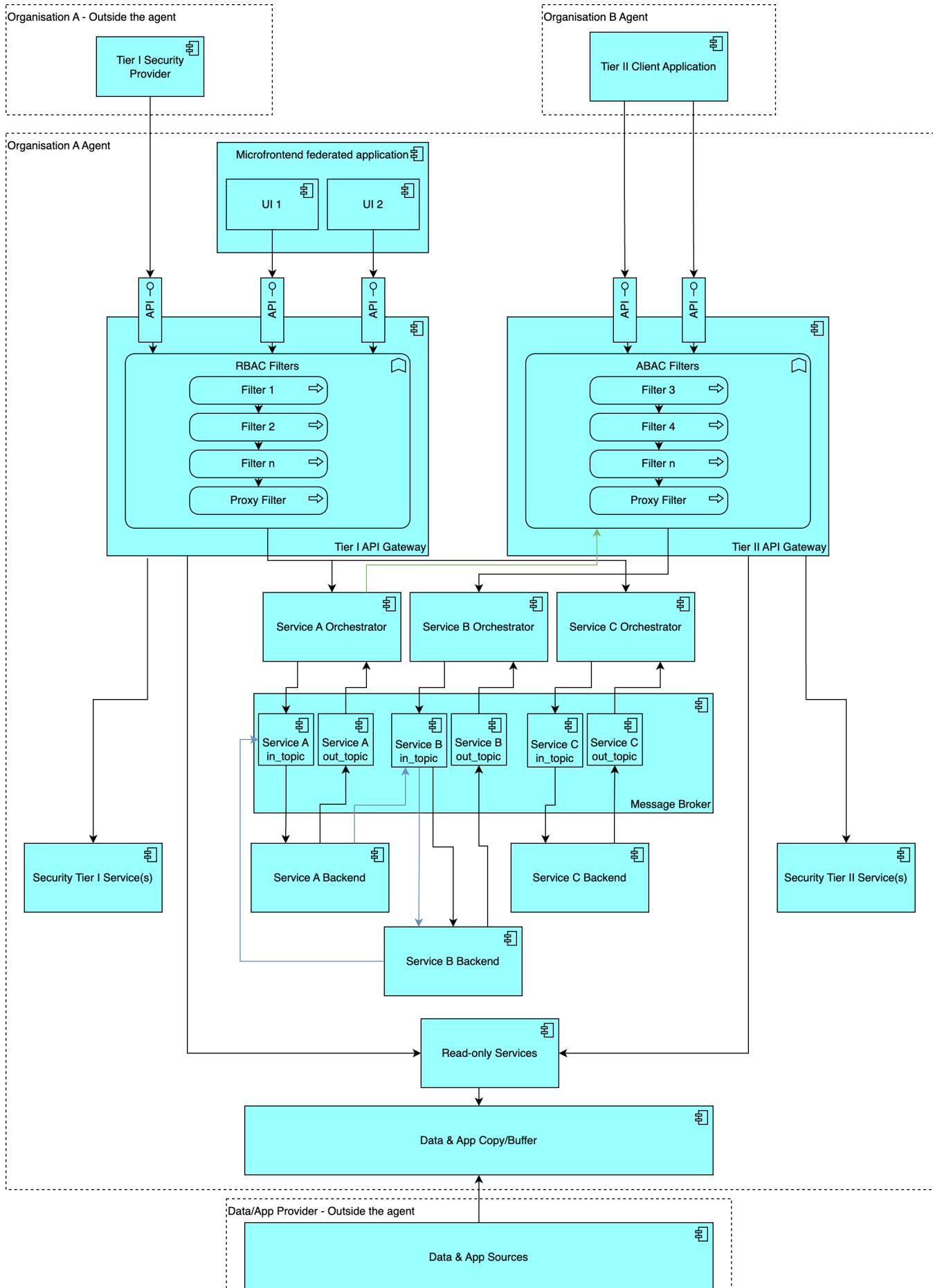
Both API Gateways will expose only synchronous HTTP endpoints to be called by their respective clients. When a request is received, the API Gateway will apply a set of filters which implement the so-called "cross-cutting concerns" (e.g. authorisation, service mesh, circuit breaker - to be further detailed later on). The last filter in the chain is a Proxy filter which redirects the request to the appropriate backend service which can handle it.

Backend services of 3 types exist:

1. Security related services which integrate synchronously over HTTP in a RESTful architecture;

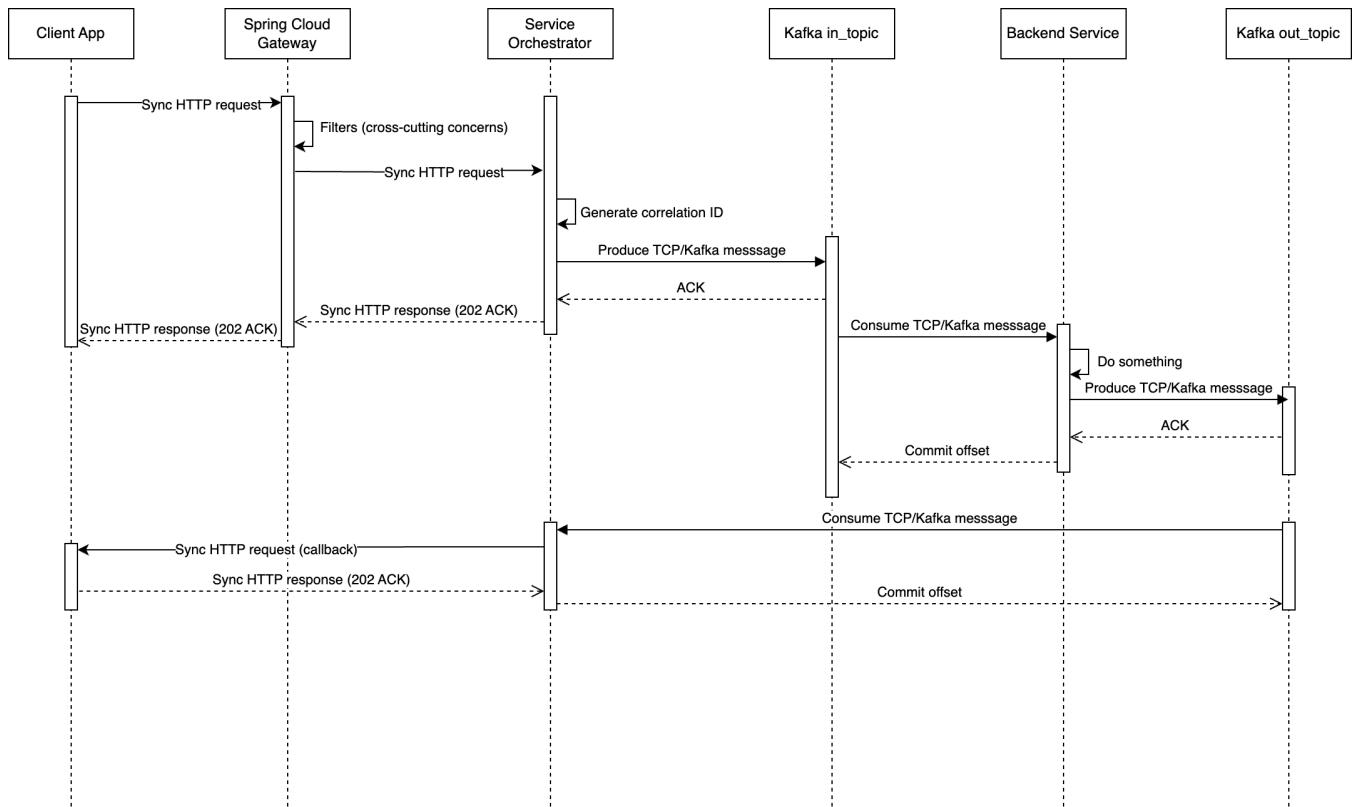
2. Read-only services, which are aimed at providing a fast response to a frontend (which is blocked waiting for a response), that also integrate synchronously over HTTP in a RESTful architecture;
3. Data processing services, which modify the state of the system (typically Create/Update/Delete), that integrate asynchronously over a message broker and are divided in 2 parts:
 - a. The Service Orchestrator receives the synchronous request from the API Gateway, publish a message in the message broker (using the outbox pattern, see below section) and acknowledge receipt of the request back to the API Gateway;
 - b. The Service Processor consumes asynchronously messages from the message broker, process them (potentially calling other services using the same asynchronous pattern) and publishes a response back to the message broker for consumption by the Orchestrator which provides it back to the client application (using polling or callback mechanism).

As it is assumed that data will not be directly consumed from the data provider node, a data & app copy/buffer (unclear at this stage how this should be) is represented on the diagram. This copy/buffer could also be located in an infrastructure provider node.



The following diagram describes at a High Level how the different components presented on the above diagram interact with each other to render a generic functionality.

For the sake of keeping this diagram simple, the outbox pattern is voluntarily not represented in this diagram and is subject to a dedicated diagram further down the page.



Outbox Pattern

In monolith and many older applications, we commonly used transactions that spanned over multiple systems. Since everything was considered to be local and within our control, applying ACID (Atomicity, Consistency, Isolation, Durability) principles was possible.

In microservices and distributed systems in general, this becomes increasingly challenging. Approaches such as implementing a multi-phase commit protocol are difficult and often require heavy weight infrastructure to pull off.

Let's take the case of a Backend Service A (e.g. the SD Publication Service for a provider) which needs to persist data locally within an agent (e.g. in the wallet) and at the same time must propagate that data to another Backend Service B (e.g. the Catalogue of the GA) to keep them in sync.

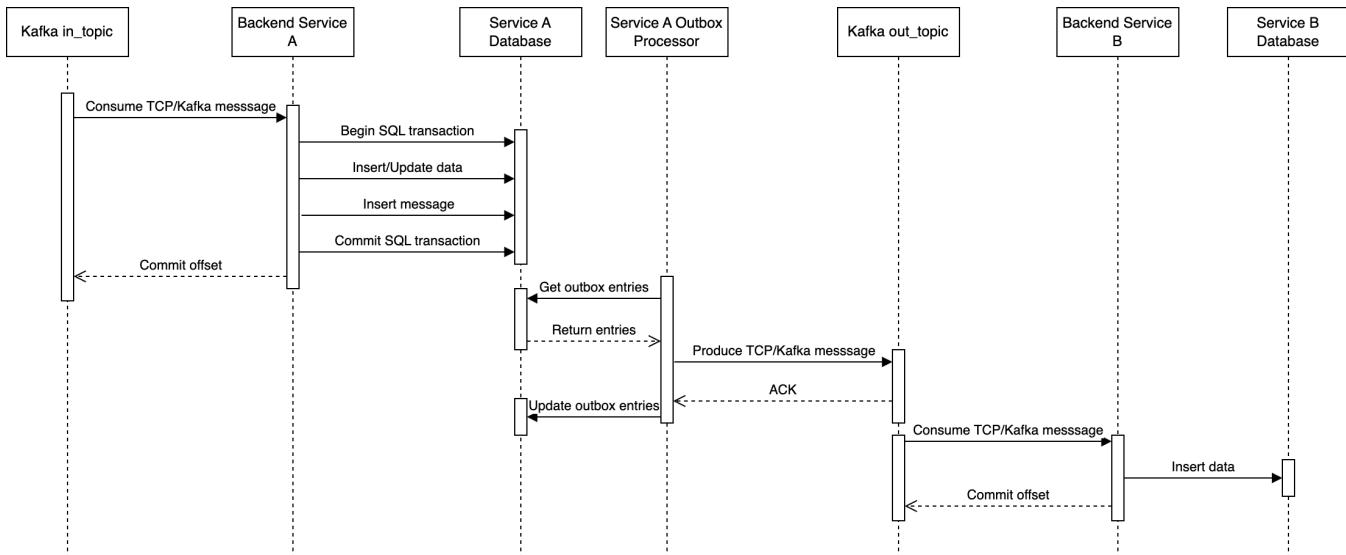
Our goal is to ensure that messages are going to be eventually published to the message broker, and ultimately available for Service B, while still persisting the data to the database of Service A.

To make this possible, we will need to save the data **and** the message that we wish to publish to the database in a single transaction. This is called the **Outbox Pattern**. The outbox pattern allows us to avoid the distributed transaction but still atomically save to the database and publish a message.

The first part of the outbox pattern makes use of a transaction to save both the data and message in an atomic operation.

The second half of this pattern requires another process or worker that will be responsible for retrieving the pending messages from the database. Once successfully retrieved, it can then proceed to publish the corresponding messages and update the state of the entries accordingly.

The sequence diagram for the transactional outbox pattern looks like this:



Annex 5 - Architecture building blocks

This Annex provides detailed descriptions of each of the building blocks that form Simpl-Open High Level Architecture.

Integration layer architecture

Resource discovery	Metadata description	Resources (data, application and infrastructure) need a metadata description to be discoverable by consumers. This metadata description informs consumers about the content of the resource, as well as its location, author, usage policy, etc.
	Resource Catalogue	The resource catalogue contains the list of all the available resources (data, application and infrastructure) for the consumers. It ensures that the providers can publish metadata description of their resources and consumers can discover the existing resources in an easy way.
	Search engine	The resource catalogue mentioned above implements query algorithms to facilitate filtering to make it possible to find the best resources based on query parameters and matching metadata descriptions.
Access control & trust	Identity provider federation	Provides an identity federation for the participants of Simpl-Open. This block includes the identity information validation, creation and management.
	Authentication provider federation	Federates existing or data space specific authentication mechanisms. Participants will access with a specific token whenever consumption or data search is needed, avoiding unapproved access to data. Additionally, it addresses Infrastructure authentication mechanisms.
	User roles	The roles needed to grant permissions and access to determined building blocks within Simpl-Open are assigned at this stage of the Access Control & Trust. Common roles across Simpl-Open enabled data spaces will provide interoperability.
	Authorization	This building block is responsible for handling the permissions of the different users so that it can be defined, what users what actions are allowed to perform on a specific resource. This building block is of crucial importance, as giving a limited access to the necessary users is one of the ways to keep a system secure.
	Security attribute provider federation	For each type of resource, different policies will apply. Often, the providers will define the access policies by assigned security attributes to users. In general, security attribute providers can be third-parties. This building block federates the attribute providers.
	Policy Enforcement	This building block ensures that all usage policies are effectively enforced within the data spaces.
	Onboarding	Provides the necessary features to submit/review/approve onboarding requests and deliver to the applicant the necessary security credentials to join a data space.
Security	Encryption	Supports the creation and management of encryption and decryption, as well as key management in secure vaults.
	Guaranteed authenticity and integrity	Supports the measures in place to ensure end-to-end data integrity, such that actors can validate the authenticity of the delivered information. This building block links to the key management services.
Network	VPN	Virtual private networks may be created in order to protect the data traffic of Simpl-Open from external threats. The VPNs are also to be used to transfer data or applications within and across data spaces over the public internet.
	Firewall	This building block allows the creation of firewall rules in order to restrict the inbound and outbound traffic of the private networks. It provides an additional protection layer on top of identity and access management.

Federation management	Federation orchestration	Provides the means to connect resources in a service network. Manages the operation of networks including processes for monitoring, runtime issues, evaluation, etc. to meet the main principles of interoperability and federation. It also configures the system components of Simpl-Open network to smoothly collaborate in a data space.
IT Application Framework	Configuration service	This building block provides the necessary feature to configure all the other building blocks of Simpl-Open.
	Circuit breaker	Handle faults that might take some time to recover from, when connecting to a remote service or resource. This can improve the stability and resiliency of a distributed application.
	Microfrontend framework	Micro frontends allow to break down a frontend application into smaller, independent pieces that can each be developed, deployed, and scaled separately.
	Policy engine	This building blocks provides technical enabler capability for the Policy Enforcement building block.
	Service mesh	The service mesh building block provides a dedicated infrastructure layer for facilitating service-to-service communications between services or microservices using a proxy. It provides various benefits such as providing observability into communications, providing secure connections, and automating retries for failed requests.

Data layer architecture

Data sharing	Simple data transfer	Simple data transfer building block is used for the exchange of small to medium sized sets of data between participants. The size of the data is typically less than a few MB's to 100 MB's. Simple data transfers can happen synchronously using a single network connection between participants.
	Bulk data transfer	Bulk transfer building block is used for the exchange of large chunks of data between participants. The size of the data is in the range of 100 MB's and above. Bulk transfer building block typically proceeds as asynchronous processes where data is transmitted in digestible chunks and recombined at the receiver side. Bulk data transfer can be sped up by using multiple network connections.
	Data streaming	Data streaming concerns a specific use case where the data exchange is not a singular occurrence, but should happen periodically. Small chunks of data are continuously being transmitted from provider to consumer. Possible sources for data streaming can be sensors or server logs. The use of data streaming typically coincides with real-time and near real-time data processing.
	Data store connector	The data store connector foresees the integration with the internal data federation of a data provider. This connector foresees integrations with multiple popular data management solutions, such as NTFS file systems, MySQL or PostgreSQL relational databases, MongoDB key-value databases, ...
Application sharing	Calculation algorithms	Simpl-Open will provide the means to share basic data science algorithms (e.g., linear regression, logistic regression, Random Forest, K-means, ...). The participant can download these tools in the infrastructure of their choosing.
	Machine learning models	Simpl-Open provides access to an ecosystem to share machine learning models. This is possible if the providers make these models discoverable and apply the correct access control policies such that consumers are authorised to access these machine learning models.
	Software & apps	In general, Simpl-Open could provide access to any type of software application. An example is a 3D rendering engine that can process the data from Destination Earth. This is possible if the providers make these apps discoverable and apply the correct access control policies such that consumers are authorised to access these machine learning models.
Data processing	Data visualisation	Simpl-Open provides access to open-source data visualizations tools like d3.js or leaflet.
	Data analytics tools	Simpl-Open provides access to basic data processing tools like Hadoop Ecosystem, Spark and Jupyter notebook to create scripts for processing data and execute this scripts on shared data.
	Anonymization	To ensure the privacy of data subject, data possibly has to be anonymised before it can be shared to certain data consumers. This anonymisation typically involves aggregating data from multiple data subjects such that individual records cannot be recovered.
Data governance	Data lineage	To monitor and ensure data integrity, tracking errors during data processing, it must be possible to show the complete flow of the data, from start to finish. Data lineage provides a complete overview of the actions that have been taken on the data by all participants.
	Data profiling	Data profiling is essentially the execution of the data governance strategy. It involves collecting descriptive statistics on the data, collecting data types, tagging data with keywords, and other steps. It helps in analysing data, implementing a data governance strategy, and determining data quality.
	Data quality rules	The consumers can assess the data quality of the offered data. The data quality will be described in the data catalogue according to certain data quality rules. These rules can contain attributes like data accuracy, whether data is complete, up to date, or available.
Distribute	Data distribution	As Simpl-Open connects data, applications and infrastructure, it should enable a distributed execution of algorithms. This building block manages the distribution of the data assets in such process.

Cloud computing & Edge computing	on management	
Data orchestration	Data orchestration	The main building block for taking siloed data from multiple data storage locations, combining them, and making them available to data consumer applications.

Infrastructure layer architecture

Cloud computing & Edge computing	VM provisioning	Simpl-Open provides an abstraction layer to provision Virtual Machines on the underlying infrastructure. The consumers may select the virtual machine(s) which are the most tailored for their needs regarding a set of parameters (e.g. operation system, memory size, network bandwidth, etc.).
	Container provisioning	Allows the provisioning & deployment of container (images) in order to launch and stop the execution of container-based algorithms, generic applications or custom code. The containers may be preferred over the VMs for the consumers in case the application portability is the most important aspect for them.
	Serverless provisioning	Serverless computing abstracts away most of the OS configuration, maintenance tasks, and notion of the underlying instances that software runs on. Simpl-Open offers serverless computing services provided by the infrastructure.
	Block storage	Simpl-Open provides an abstraction layer to provision various kinds of storage on the underlying infrastructure. Block storage provides low level storage capabilities to consumers to store any type of binary data in ordered blocks.
	File system provisioning	Simpl-Open provides an abstraction layer to provision various kinds of storage on the underlying infrastructure. File systems are provided to consumers to store files in structured directories.
PaaS services	SQL databases	Standard relational databases. These are the most suitable option for storing primarily structured, transaction-oriented data, in case the structure does not change frequently and when the data integrity is important. The database options provided by Simpl-Open can be open-source or commercially licensed. A relevant tool example include MySQL as the most prominent open source solution in the current market.
	NoSQL databases	NoSQL databases are the best option in case the user needs more flexible schemas and/or horizontal scaling. They also enable faster queries due to the data model. Simpl-Open may offer cloud-native or open-source non-relational databases for the users. Among others, tools like Apache Cassandra or MongoDB provide this type of database management
	Time series databases	Databases designed specifically for the storage and retrieval of data associated with a timestamp (time series data). These databases are typically the most suitable for storing sensor data, as they can compress, manage and summarize the time series data, and handle time-aware queries. As this type of data is useful for monitoring and reporting capabilities, tools like Prometheus and Grafana combined become a powerful stack for this purpose.
	Graph databases	Graph databases are built to allow an easy/performant way to query relationships. Graph databases use nodes to store data entities and edges to store relationships. The best examples are (social) networks or supply chains.
	AI provisioning	Simpl-Open may provide abstractions to allow for a simple deployment and execution of AI models. The consumers may select from the AI services provided through Simpl-Open and use them to gain insights from their own data set(s). Many solutions can be found in the LF & AI landscape founded by Linux, for instance TensorFlow.
	Blockchain	Simpl-Open will provide blockchain services to enable building applications where a decentralised, shared system of records is needed. Its most important advantages are data integrity, reliability, the speed of the storage, immutability and transparency, therefore, it is ideal for, for example, recording logs for audit and regulatory compliance or documenting payments.
	Messaging busses	The messaging busses mediate the message exchange between different systems via a shared set of interfaces (message bus). The sender may publish messages on a queue, which will be transferred to the subscribing receiver(s). Apache Kafka is a relevant solution that specifically implements a messaging system as described.
	Analytics provisioning	This building block enables the consumers of Simpl-Open to access high-level data analytics services without the need to provision the tooling manually (e.g., ETL services, Apache Spark).
HPC	HPC	In a later phase, Simpl-Open will provide high performance computing power to enable parallel data processing and the conduction of complex, resource-intensive calculations.
Distributed execution	Infrastructure management	The infrastructure management runs and clusters the underlying networked computers from different providers to create the impression that a single and reliable machine is processing the computations. It allows computations to be executed close to where the data is located.

Infrastructure orchestration	Infrastructure orchestration	This building block is responsible for automating the provisioning of the infrastructure resources needed for the computations conducted on Simpl-Open. It allows the various infrastructure providers to interconnect and exposes them via a standard interface.
------------------------------	------------------------------	---

Administration layer architecture

C o n tr a cts	License asset management	Administers any topic related to recommended licenses by both Simpl-Open and the Members States, compatibility and harmonization between the different data spaces.
	SLA management	Hosts the mechanism that defines and shares the Service Level Agreements between contractors. Administration capabilities will handle SLA-related issues as well as SLA sharing when required.
	Usage contracts	It administers the types of resources that will be provided to the consumers by the resources providers, and the conditions which consumers must adhere to.
	Billing	The administration of any topic related to the contract billing, such as conditions or specifications are included in this building block. Specifications may include procedures, due times, quantities, periods of payments, etc.
M o ni t o ri ng	Data usage	Supervision of the amount and type of data flowing through Simpl-Open; and what it is being used for.
	Application usage	Supervision of what are the apps being used by any of the consumers; and how they are being used.
	Infrastructure usage	Supervision of the particular infrastructure resources that are being used.
	Usage policies	Integration of usage control capabilities regarding the policies that can be defined in the environment. These policies will describe the terms and conditions under which apps, data or infra can be used on the consumer side.
	Energy metrics and alerts	Monitoring supervision will be reflected in a functionality that gathers metrics and gives alerts regarding the general state of Simpl-Open in terms of data, app and infra usage in order to optimize energy usage and meet sustainability goals.
	QoS metrics and alerts	Monitoring supervision will be reflected in a functionality that gathers metrics and gives alerts regarding the general state of Simpl-Open in terms of data, app and infra usage, providing information about the quality of service that is being given.
	Performance monitoring & enactment	As a previous stage to the performance reporting building block, usage and resource availability performance is monitored in a real-time basis to feed the information sets needed by the reporting block.
R e p o rt ing	Performance	Allows the analysis of the performance of each of the functionalities to serve as a source for the general report building block. Along with the monitoring block, provides past performance and history records of the metrics gathered at the monitoring stage.
	Platform usage	Creates reports and supervises the use of the global Simpl-Open platform in both the infrastructure and data layer by the consumers and providers.
	Energy efficiency & sustainability	To support the objective of the Green Deal, this component will provide insights into the energy efficiency and environmental performance of Simpl-Open's building blocks to ensure that the resources operate in a low power mode.
	Log info extraction	Provides the information needed obtained from sources such as the audit logging helping to generate a comprehensive report of the functionalities.
	Exporting	Exporting building block allows information gathered by the reporting processes to be external and available outside of Simpl-Open environment.
L o g gi ng	Logging	Record and manage events related to the state of the Simpl-Open agent, the usage of a resource and business events.
A u dit	Audit	The administrator can investigate the activities taking part in each of the layers of Simpl-Open to further develop a reporting about issues or non-compliant conditions of the resource at data or infrastructure stages. This building block gathers the information of auditing actions performed in Simpl-Open context, allowing to undertake organizational or legal measures that ensure the compliant functioning of Simpl-Open.

Governance layer architecture

Support	Support page	Consumers will have access to a support webpage with collected and useful documentation regarding Simpl-Open with a FAQ format.
	Ticketing system	Users will have availability of a ticketing system that logs the issues regarding Simpl-Open to be reported to the administration.
	Helpdesk	Third party contractors can connect with the consumers and providers in case the issues remain unsolved. The governance board will coordinate these three sub-blocks.
CSIRT	Incident response	Coordination at the administration layer will give a response to security incidents with proper procedures to restore full Simpl-Open functionality as soon as possible.
	Threat monitoring	Proactive threat monitoring and follow-up of possible malicious activities affecting Simpl-Open to avoid potential security breaches before they happen.