



MATRYCS

## **D3.1 | MATRYCS- GOVERNANCE (1<sup>st</sup> technology release)**

**WP3 – Data Services and Semantic  
Enrichment Layer**

*August 2021*



**[www.matrycs.eu](http://www.matrycs.eu)**

## Modular Big Data Applications for Holistic Energy Services in Buildings



### Disclaimer

---

The sole responsibility for the content of this publication lies with the authors. It does not necessarily reflect the opinion of the European Union. Neither the EASME nor the European Commission is responsible for any use that may be made of the information contained therein.

---

### Copyright Message

---

This report, if not confidential, is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0); a copy is available here: <https://creativecommons.org/licenses/by/4.0/>. You are free to share (copy and redistribute the material in any medium or format) and adapt (remix, transform, and build upon the material for any purpose, even commercially) under the following terms: (i) attribution (you must give appropriate credit, provide a link to the license, and indicate if changes were made; you may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use); (ii) no additional restrictions (you may not apply legal terms or technological measures that legally restrict others from doing anything the license permits).

---



The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158



Grant Agreement Number	101000158	Acronym	MATRYCS
Full Title	Modular Big Data Applications for Holistic Energy Services in Buildings		
Topic	LC-SC3-B4E-6-2020   Big data for buildings		
Funding scheme	H2020- IA: Innovation Action		
Start Date	October 2020	Duration	36
Project URL	www.matrycs.eu		
Project Coordinator	ENG		
Deliverable	D3.1 MATRYCS-GOVERNANCE (1st technology release)		
Work Package	WP3 – Data Services and Semantic Enrichment Layer		
Delivery Month (DoA)	August 2021	Version	1.0
Actual Delivery Date	31/08/2021		
Nature	Other	Dissemination Level	Public
Lead Beneficiary	ENG		
Authors	Dario Pellegrino [ENG], Marija Borisov [ENG], Leandro Lombardo [ENG], Francesco Saverio Nucci [ENG], Sofía Mulero [CARTIF], Víctor Iván Serna [CARTIF], Timotej Gale [COMSENSUS], Tomaž Bračič [COMSENSUS], Andrej Čampa [COMSENSUS], Zoi Mylona [HOLISTIC], Panagiotis Kapsalis [NTUA], Daniele Antonucci [EURAC], Zhiyu Pan [RWTH]		
Quality Reviewer(s):	Javier Román [CARTIF], Gema Hernández Moral [CARTIF], Zoi Mylona [HOLISTIC]		
Keywords	Data Governance, Data Services, Semantic Enrichment		





## Preface

MATRYCS focuses on addressing emerging challenges in big data management for buildings with an **open holistic solution** for Business to Business platforms, able to give a competitive solution to stakeholders operating in building sector and to open new market opportunities. **MATRYCS Modular Toolbox**, will realise a holistic, state-of-the-art AI-empowered framework for decision-support models, data analytics and visualisations for Digital Building Twins and real-life applications aiming to have significant impact on the building sector and its lifecycle, as it will have the ability to be utilised in a wide range of use cases under different perspectives:

- Monitoring and improvement of the energy performance of buildings - **MATRYCS-PERFORMANCE**
- Design facilitation and development of building infrastructure - **MATRYCS-DESIGN**
- Policy making support and policy impact assessment - **MATRYCS-POLICY**
- De-risking of investments in energy efficiency - **MATRYCS-FUND**





## Who We Are

	Participant Name	Short Name	Country Code	Logo
1	ENGINEERING – INGEGNERIA INFORMATICA SPA	ENG	IT	
2	NATIONAL TECHNICAL UNIVERSITY OF ATHENS	NTUA	GR	
3	FUNDACION CARTIF	CARTIF	ES	
4	RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN	RWTH	DE	
5	ACCADEMIA EUROPEA DI BOLZANO	EURAC	IT	
6	HOLISTIC IKE	HOLISTIC	GR	
7	COMSENSUS, KOMUNIKACIJE IN SENZORIKA, DOO	COMSENSUS	SL	
8	BLAGOVNO TRGOVINSKI CENTER DD	BTC	SL	
9	PRZEDSIĘBIORSTWO ROBOT ELEWACYJNYCH FASADA SP. Z O.O.	FASADA	PL	
10	MIASTO GDYNIA	GDYNIA	PL	
11	COOPERNICO - COOPERATIVA DE DESENVOLVIMENTO SUSTENTAVEL CRL	COOPERNICO	PT	
12	ASM TERNI SPA	ASM	IT	
13	VEOLIA SERVICIOS LECAM SOCIEDAD ANONIMA UNIPERSONAL	VEOLIA	ES	
14	ICLEI EUROPEAN SECRETARIAT GMBH (ICLEI EUROPASEKRETARIAT GMBH)	ICLEI	DE	
15	ENTE PUBLICO REGIONAL DE LA ENERGIA DE CASTILLA Y LEON	EREN	ES	
16	VIDES INVESTICIJU FONDS SIA	LEIF	LV	
17	COMITE EUROPEEN DE COORDINATION DE L'HABITAT SOCIAL AISBL	HOUSING EUROPE	BE	
18	SEVEN, THE ENERGY EFFICIENCY CENTER Z.U.	SEVEN	CZ	





## Contents

<b>1</b>	<b>Introduction .....</b>	<b>15</b>
1.1	Purpose of the document .....	15
1.2	Structure of the document .....	15
<b>2</b>	<b>MATRYCS-GOVERNANCE architecture .....</b>	<b>16</b>
2.1	Overall MATRYCS-GOVERNANCE architecture .....	16
2.2	MATRYCS-GOVERNANCE Building Blocks .....	17
2.2.1	Interoperability Service Module .....	17
2.2.2	Data pre-processing and semantic enrichment .....	18
2.2.3	Streaming module.....	20
2.2.4	Data Storage.....	20
2.2.5	High Performance Distributed Query Engine.....	21
2.2.6	Reasoning Engine.....	22
2.2.7	Trusted Data Sharing (DLT/Blockchain).....	23
2.2.8	End-to-End Security framework .....	24
<b>3</b>	<b>MATRYCS Data Governance solution .....</b>	<b>26</b>
3.1	Overview Data Governance solution .....	26
3.2	Interoperability Service Module .....	26
3.2.1	Interoperability implementation description .....	26
3.2.2	Interoperability Data Connectors.....	27
3.2.3	Technological components .....	28
3.2.4	Interaction with other Data Governance components .....	34
3.3	Data pre-processing and semantic enrichment.....	35
3.3.1	Data pre-processing and semantic enrichment implementation description .....	35
3.3.2	Technological components .....	35
3.3.3	Interaction with other Data Governance components .....	37
3.4	Streaming module.....	37
3.4.1	Streaming module implementation description .....	37
3.4.2	Technological components .....	38
3.4.3	Interaction with other Data Governance components .....	42
3.5	Data Storage.....	43
3.5.1	Data Storage implementation description .....	43
3.5.2	Technological components .....	44
3.5.3	Interaction with other Data Governance components .....	46
3.6	Reasoning Engine.....	46
3.6.1	Reasoning Engine implementation description .....	46
3.6.2	Technological components .....	47
3.6.3	Interaction with other Data Governance components .....	49
3.7	High Performance Distributed Query Engine.....	50
3.7.1	High Performance Distributed Query Engine implementation description .....	50
3.7.2	Technological components .....	50





3.7.3	Interaction with other Data Governance components .....	53
3.8	Trusted Data Sharing (DLT/Blockchain) .....	53
3.8.1	Trusted Data Sharing implementation description.....	53
3.8.2	Technological components .....	55
3.8.3	Interaction with other Data Governance components .....	58
3.9	End-to-End Security framework .....	58
<b>4</b>	<b>MATRYCS Data Model.....</b>	<b>60</b>
4.1	Vocabularies and Ontologies.....	61
4.1.1	Brick schema.....	61
4.1.2	SAREF .....	67
<b>5</b>	<b>MATRYCS-GOVERNANCE Integration at M11 .....</b>	<b>69</b>
5.1	Case study LSP1 and LSP5 .....	69
5.1.1	Data Acquisition.....	71
5.1.2	Data processing and modelling.....	73
5.1.3	Data Storage and Reasoning Engine .....	79
5.1.4	Data Access Layer.....	84
<b>6</b>	<b>MATRYCS-GOVERNANCE: Final considerations and next steps.....</b>	<b>85</b>



## Figures

Figure 1: Big Data Management phases.....	16
Figure 2: MATRYCS-GOVERNANCE Conceptual Architecture .....	17
Figure 3: Interoperability Service Module .....	18
Figure 4: Data Pre-processing Service .....	19
Figure 5: Streaming Module.....	20
Figure 6: Data Storage.....	21
Figure 7: Distributed Query Engine .....	22
Figure 8: Reasoning Engine .....	23
Figure 9: Trusted data sharing.....	24
Figure 10: End-to-End Security Framework.....	25
Figure 11: Overview of the MATRYCS-GOVERNANCE Solution .....	26
Figure 12: Apache NiFi processor groups .....	28
Figure 13: Apache NiFi Architecture.....	29
Figure 14: Apache NiFi Cluster .....	29
Figure 15: Nginx Load-balancing Reverse Proxy.....	30
Figure 16: Interoperability Service Module.....	31
Figure 17: Apache Kafka Architecture.....	38
Figure 18: Kafka with multiple partitions for the topic and two producers .....	39
Figure 19: Interaction of the Streaming Module with other components.....	42
Figure 20- NiFi PublishKafka processor configuration.....	42
Figure 21: Confluent Kafka package in Jupyter environment.....	43
Figure 22: Reasoning Engine Solution and sub-modules .....	47
Figure 23: Reasoning Engine interaction with Data Streaming Module.....	50
Figure 24: Blockchain data structure.....	53
Figure 25: Blockchain network.....	54
Figure 26: Trusted data sharing flow.....	54
Figure 27: IoT Gateway .....	57
Figure 28: Blockchain platform architecture.....	58







Figure 29: Example of Brick Schema.....	66
Figure 30: Overview of the SAREF ontology.....	67
Figure 31: General overview of the top levels of the SAREF4BLDG.....	68
Figure 32: LSP1 and LSP5 data acquisition .....	71
Figure 33: LSP1 data connector .....	72
Figure 34: LSP5 data connector .....	73
Figure 35: LSP01 BTC data model .....	77
Figure 36: LSP05 Coopernico data model.....	77
Figure 37: Stored LSP1 data on GraphDB .....	82
Figure 38: LSP5 data stored in the GraphDB.....	83

## Tables

Table 1: NiFi node Docker Compose file .....	32
Table 2: NGINX Docker Compose file.....	33
Table 3: NGINX configuration file .....	34
Table 4: Comparison data model with ontology .....	36
Table 5: Docker-compose file for Apache Kafka.....	40
Table 6: Data storage microservice Dockerfile.....	45
Table 7: Docker-compose file for ScyllaDB.....	45
Table 8: Example of Reasoning Engine REST API .....	48
Table 9: Example of Reasoning Engine LEIF Recommendation Service.....	48
Table 10: Docker-compose file for Reasoning Engine .....	48
Table 11: Dockerfile for Presto .....	51
Table 12: Docker-compose file for Presto.....	52
Table 13: Metadata schema as resulting of review process of existing applications .....	62
Table 14: Main core concepts of building ontology .....	63
Table 15: Relationship and definition for brick and brick plus schema .....	66
Table 16: DATASET_ORDER_DICT in python script.....	78
Table 17: Data Storage Golang Kafka consumer <sup>78</sup> .....	79





Table 18: Staging area file for LSP1 .....	80
Table 19: Staging area file for LSP5 .....	80
Table 20: LSP1 Reasoning Engine payload .....	80
Table 21: LSP1 import script to Reasoning Engine .....	81
Table 22: LSP5 Reasoning Engine payload .....	82
Table 23: LSP5 import script to Reasoning Engine .....	83
Table 24: Example Presto query submission .....	84
Table 25: Reasoning Engine Rest API.....	84





## Abbreviation and Acronyms

Acronym	Description
<b>ADE</b>	Application Domain Extension
<b>AHU</b>	Air Handling Unit
<b>AI</b>	Artificial Intelligence
<b>API</b>	Application Programming Interface
<b>BACs</b>	Building Automation and Control systems
<b>B2B</b>	Business-to-Business
<b>BDVC</b>	Big Data Value Chain
<b>BEDES</b>	Building Energy Data Exchange Specifications
<b>BEMS</b>	Building Energy Management System
<b>BIM</b>	Building Information Modelling
<b>BMS</b>	Building Management System
<b>BOT</b>	Building Topology/Ontology
<b>CLI</b>	Command Line Interface
<b>CO<sub>2</sub></b>	Carbon dioxide
<b>D</b>	Deliverable
<b>DB</b>	Database
<b>DEB</b>	Debian Package
<b>DL</b>	Deep Learning
<b>DLT</b>	Distributed Ledger Technology
<b>DH</b>	District Heating
<b>DHN</b>	District Heating Network
<b>DSO</b>	Distribution System Operator
<b>DSM</b>	Data Storage Management
<b>ECM</b>	Energy Conservative Measure
<b>EEM</b>	Energy Efficiency Measure
<b>EE</b>	Energy Efficiency
<b>EM-KPI</b>	?????
<b>EMS</b>	Energy Management System
<b>EP</b>	Energy Performance
<b>EPBD</b>	Energy Performance of Buildings Directive
<b>EPC</b>	Energy Performance Certificate
<b>ESCO</b>	Energy Services Company
<b>ETL</b>	Extract, Transform, Load
<b>EU</b>	European Union
<b>EV</b>	Electric Vehicle
<b>EVM</b>	Ethereum Virtual Machine
<b>gbXML</b>	Green Building eXtensible Markup Language
<b>GDPR</b>	General Data Protection Regulation
<b>GUI</b>	Graphical User Interface
<b>HTO</b>	Haystack Tagging Ontology
<b>HTTPS</b>	Hypertext Transfer Protocol Secure
<b>ICT</b>	Information and Communication Technologies
<b>IEQ</b>	Indoor Environmental Quality
<b>IFC</b>	Industry Foundation Classes data model
<b>IoT</b>	Internet of Things





<b>IP</b>	Internet Protocol
<b>JSON</b>	JavaScript Object Notation
<b>KM4City</b>	Knowledge Model for City
<b>KPI</b>	Key Performance Indicator
<b>LSP</b>	Large Scale Pilot
<b>M&amp;V</b>	Measurement and Verification
<b>MQTT</b>	Message Queue Telemetry Transport
<b>ML</b>	Machine Learning
<b>NDJSON</b>	Newline Delimited Java Script Object Notation
<b>NGSI</b>	Next Generation Service Interfaces
<b>NGSI-LD</b>	Next Generation Service Interfaces Linked Data
<b>NIS</b>	Network Information Security
<b>NoSQL</b>	Not Only Structured Query Language
<b>obXML</b>	Occupant Behavior XML Schema
<b>oneDM</b>	One Data Model
<b>OP</b>	Occupancy Profile
<b>OPM</b>	Ontology of Property Management
<b>OWL</b>	Web Ontology Language
<b>O&amp;M</b>	Operations & Maintenance
<b>PMB</b>	Project Management Board
<b>PV</b>	Photovoltaic
<b>RDF</b>	Resource Description Framework
<b>RDFS</b>	Resource Description Framework Schema
<b>RECs</b>	Renewable Energy Communities
<b>RES</b>	Renewable Energy Sources
<b>REST</b>	Representational State Transfer
<b>RPM</b>	Red Hat Package Manager
<b>SAREF</b>	Smart Applications REference
<b>SECAP</b>	Sustainable Energy and Climate Action Plan
<b>SFTP</b>	SSH File Transfer Protocol
<b>SME</b>	Small and Medium-sized Enterprises
<b>SHACL</b>	Shapes Constraint Language
<b>SOSA</b>	Sensor, Observation, Sample and Actuator
<b>SQL</b>	Structured Query Language
<b>SSN</b>	Semantic Sensor Network
<b>SSH</b>	Secure Shell
<b>SSL</b>	Secure Sockets Layer
<b>TBM</b>	Technical Building Management
<b>TLS</b>	Transport Layer Security
<b>TTL</b>	Turtle file
<b>URI</b>	Uniform Resource Identifier
<b>URL</b>	Uniform Resource Locator
<b>VBIS</b>	Virtual Building Information Systems
<b>WMS</b>	Warehouse Management System
<b>WP</b>	Work Package
<b>YAML</b>	Ain't Markup Language



## Executive Summary

The D3.1 – *MATRYCS-GOVERNANCE (1<sup>st</sup> technology release)* provides a description of the implementation of the 1<sup>st</sup> technology release for Data Services and Semantic Enrichment layer (MATRYCS-GOVERNANCE) according with the MATRYCS high-level reference architecture defined in the deliverable *D2.3 MATRYCS Reference Architecture for Buildings Data v1.0*.

This 1<sup>st</sup> technology release is mainly focused to the preliminary valuation of the envisaged technologies solutions for the MATRYCS-GOVERNANCE layer with limited data and in a constraint scenario and it reports the activities done until M11 in the WP3 *Data services & Semantic Enrichment Layer (MATRYCS-GOVERNANCE)* and in particular related to tasks T3.1 *DLT & Smart Contracts for B2B Cross-stakeholder Trusted Off-chain Data Sharing and Re-use*, T3.2 *Data Interoperability*, T3.3 *Data Curation, Anonymisation, Access Policy and Semantic Enrichment*, T3.4 *Data Storage and High Performance, Distributed Query Engine*, T3.5 *Reasoning Engine* and T3.6 *End-to-end security framework*.

The work reported in D3.1 - MATRYCS-GOVERNANCE (1<sup>st</sup> technology release) provides an overview of the conceptual architecture of the MATRYCS-GOVERNANCE layer with a general description of its building blocks. For each building block, technological solutions have been identified and evaluated including information on implementation methods and interactions between the different MATRYCS-GOVERNANCE modules. A preliminary study of the MATRYCS Common Data Model was started, considering also reference vocabularies and ontologies. Finally, the MATRYCS-GOVERNANCE data flow processes were demonstrated for two large-scale pilots (LSPs), namely, for LSP1 (BTC) and LSP5 (Coopernico).



# 1 Introduction

## 1.1 Purpose of the document

The purpose of D3.1 “MATRYCS-GOVERNANCE (1<sup>st</sup> technology release)” is to report the implementation of the 1<sup>st</sup> technology release of the Data services & Semantic Enrichment Layer (MATRYCS-GOVERNANCE). In this regard, this deliverable reports the activities carried out and the outcomes obtained in WP3 *Data services & Semantic Enrichment Layer (MATRYCS-GOVERNANCE)* and, in particular, related to tasks T3.1 *DLT & Smart Contracts for B2B Cross-stakeholder Trusted Off-chain Data Sharing and Re-use*, T3.2 *Data Interoperability*, T3.3 *Data Curation, Anonymisation, Access Policy and Semantic Enrichment*, T3.4 *Data Storage and High Performance, Distributed Query Engine*, T3.5 *Reasoning Engine* and T3.6 *End-to-end security framework*.

The work done so far is focused to the preliminary valuation of the envisaged technologies solutions for the MATRYCS-GOVERNANCE layer highlighting the conceptual architectures of the different MATRYCS-GOVERNANCE components, the technological solutions, the interaction between modules, implementation aspects and deployment approaches.

## 1.2 Structure of the document

The D3.1 “MATRYCS-GOVERNANCE (1<sup>st</sup> technology release)” is organized as follows:

- In section 1, the purpose of the document and related structure is presented.
- In section 2, an overview of the conceptual architecture of MATRYCS-GOVERNANCE layer with a general description of its building blocks is presented.
- In section 3, the MATRYCS-GOVERNANCE layer solution is provided. An explanation of the adopted solution for each module is given focusing on the implementation aspects, technological solutions, deployment approaches and interactions between modules of the MATRYCS-GOVERNANCE layer.
- In section 4, preliminary work done to define the MATRYCS Common Data Model.
- In section 5, the MATRYCS-GOVERNANCE data pipeline is demonstrated for two case studies, namely: LSP1 (BTC) and LSP5 (Coopernico).
- Finally, section 6 outlines the upcoming activities to be undertaken to proceed with the implementation of the 2<sup>nd</sup> technology release of the MATRYCS-GOVERNANCE layer.

## 2 MATRYCS-GOVERNANCE architecture

### 2.1 Overall MATRYCS-GOVERNANCE architecture

The main objective of the data services and semantic enrichment (MATRYCS-GOVERNANCE) layer is to provide the necessary middleware to act as a mediator between MATRYCS Data Providers and the MATRYCS data users (Analytics tools and services).

The MATRYCS-GOVERNANCE, using a holistic approach, will consider different domain and non-domain data such as building data, energy data, sensors data, energy uses data (heating, lighting, cooling, air conditioning, ventilation), weather data, etc..

According to a Big Data Value Chain approach, MATRYCS-GOVERNANCE has been designed identifying the key high-level activities to guarantees the integration, pre-processing, semantic annotation, storing and querying of the heterogeneous data handled in the MATRYCS project.

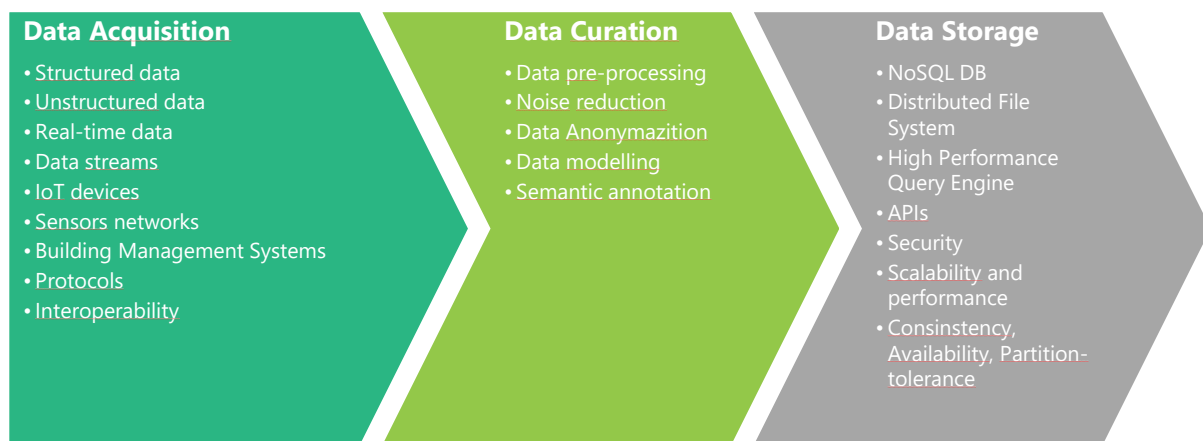


Figure 1: Big Data Management phases

**Data acquisition** step is guaranteed through an Interoperability Service Module which will be in charge of facilitating data integration of heterogeneous data sources and/or platforms belonging to different MATRYCS actors. A Streaming module will also manage the data streaming for the MATRYCS PROCESSING layer for the in-memory processing of the high latency near real time data.

**Data Curation** step is performed through the Pre-processing & Semantic Enrichment Module which is in charge data pre-processing activities such as cleansing, curation, anonymisation and semantic annotation. This module is also in charge for data modelling activities by using pre-existing vocabularies and ontologies to define the MATRYCS Common Data Model.

**Data storage** phase is covered by two different modules: the High-Performance Query Engine and the Reasoning engine. The High-Performance Query Engine is built on top of a NoSQL DB to perform complex queries in very efficient and high scalable way. The Reasoning Engine based on Graph Database technology is used to persist semantic datasets and any RDF information produced by the Pre-processing & Semantic Enrichment Module. Both modules expose intelligent querying systems and APIs to be used by the upper MATRYCS-PROCESSING and MATRYCS-ANALYTICS layers.

Finally, a security layer for the whole MATRYCS platform and then also for the MATRYCS-

GOVERNANCE layer, is guaranteed through the **End-To-End Security Framework** reported in the Deliverable D3.2<sup>1</sup>.

In the *Figure 2* the overall MATRYCS-GOVERNANCE conceptual architecture.

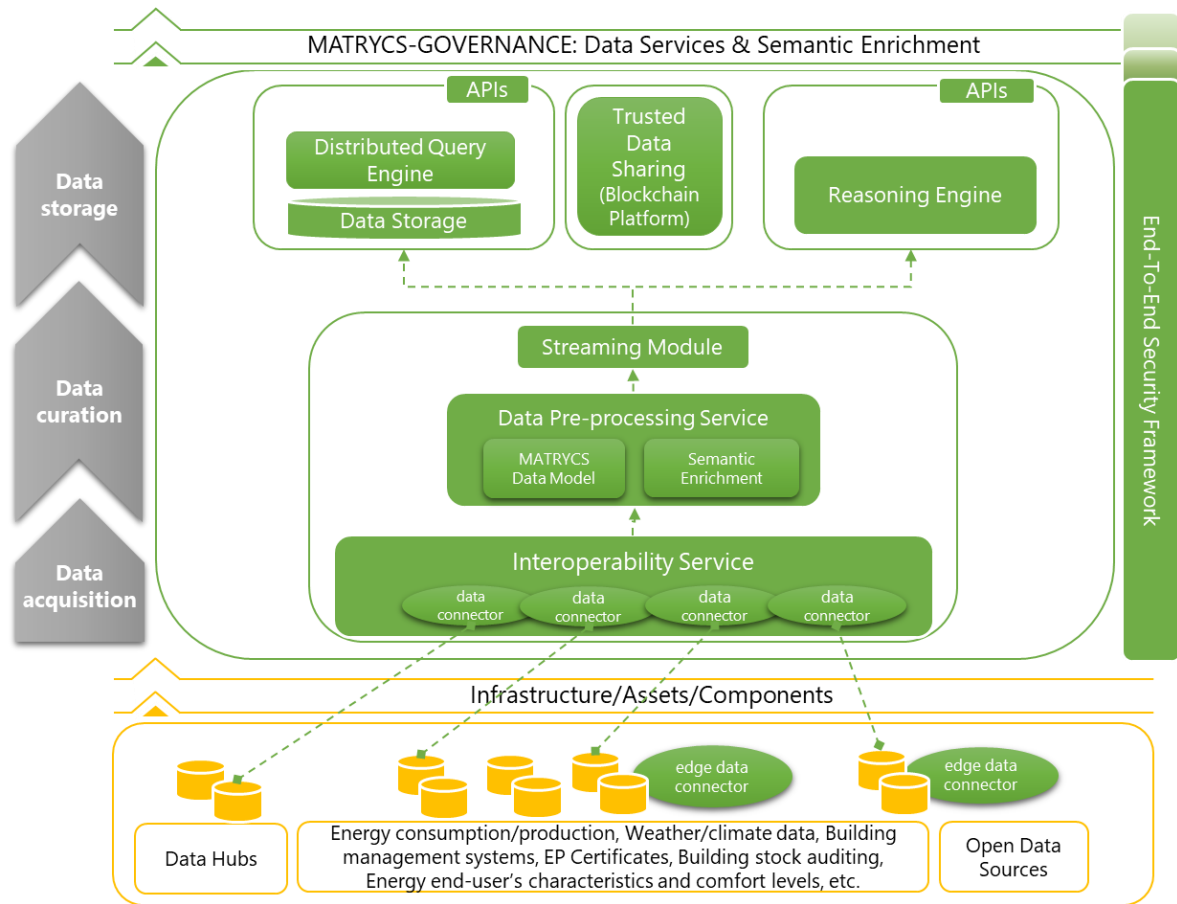


Figure 2: MATRYCS-GOVERNANCE Conceptual Architecture

## 2.2 MATRYCS-GOVERNANCE Building Blocks

### 2.2.1 Interoperability Service Module

The Interoperability Service Module (*Figure 3*) will be in charge of integrating heterogeneous data from different sources and/or platforms belonging to different MATRYCS Data Providers in the energy and non-energy ecosystem such as Smart Meters, Sensors, IoT devices, Building Management Systems (BMSs), Energy Performance Contracts/Certificates, legacy systems. This component will be based on micro-services (data connectors) to handle the different data sources provided with different data format and different communication protocols (Rest APIs, SFTP, IoT protocols, Sensor Network) providing also interfaces to other third-party energy and non-energy datasets/platforms willing to federate/integrate with MATRYCS, in order to enable incremental deployment of the MATRYCS Data Hub. The Interoperability Service Module will also be able to handle open file format, such as .IFC (the

<sup>1</sup> Deliverable 3.2: End-to-End Security Framework



Industry Foundation Classes data model<sup>2</sup>) to facilitate the interoperability for BIM (Building Information Model<sup>3</sup>) data sharing.

This module will also be able to cover data exchange mechanisms at EDGE level, with the aim to offer both local computational power and storage capacity to services and functions that may need ultra-low latency, as well as local processing without leveraging on core cloud remote services.

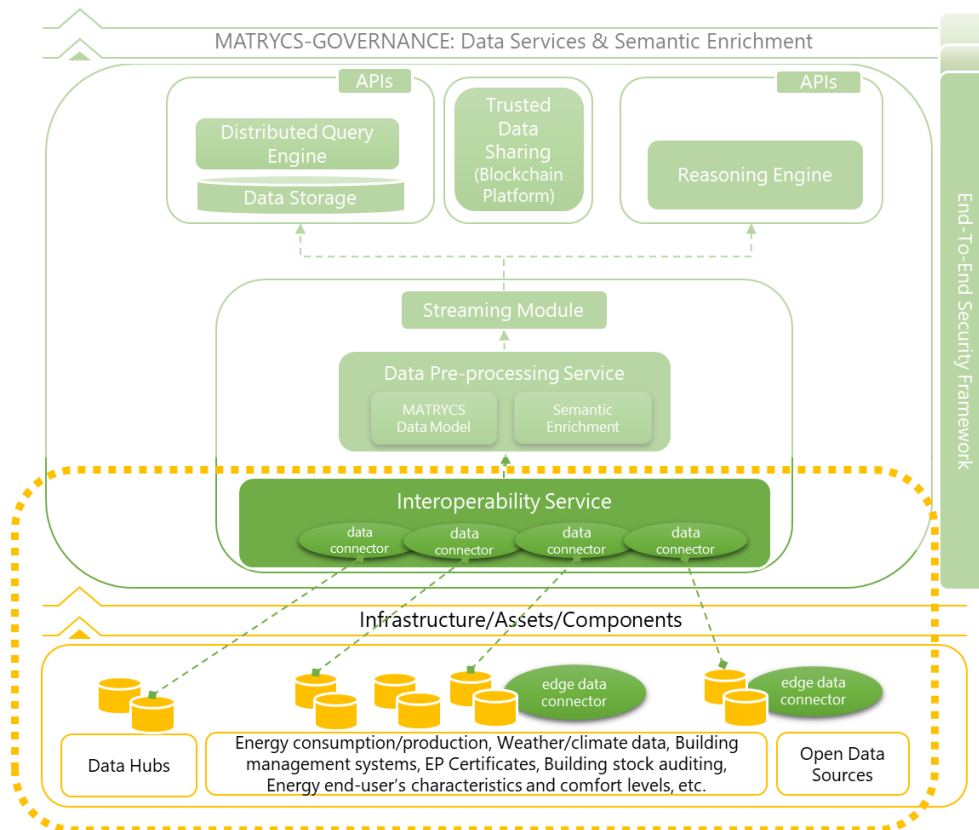


Figure 3: Interoperability Service Module

## 2.2.2 Data pre-processing and semantic enrichment

The main goal of Data pre-processing service (Figure 4) is to provide optimal data, which will be processed by the MATRYCS model development and training services. After receiving the datasets from the Interoperability module, it should go through a whole data pre-processing pipeline, which contains the anonymization, curation, harmonization transformation, and data semantic enrichment.

**Anonymization** has been defined as a “process by which personal data is irreversibly altered in such a way that a data subject can no longer be identified directly or indirectly, either by the data controller alone or in collaboration with any other party”<sup>4</sup>. The functionality of anonymization consists of two parts: personal data detection and personal data protection.

The **data curation** covers all the processes for maintenance, management, and control data, which include restricting, predefined values substitution, reformatting of field, outliers’ detection, data

<sup>2</sup> [https://www.ifcwiki.org/index.php?title=IFC\\_Wiki](https://www.ifcwiki.org/index.php?title=IFC_Wiki)

<sup>3</sup> [https://en.wikipedia.org/wiki/Building\\_information\\_modeling](https://en.wikipedia.org/wiki/Building_information_modeling)

<sup>4</sup> ISO 25237:2017 Health informatics -- Pseudonymization. ISO. 2017. p. 7.

inconsistencies handling, and noise reduction.

The **harmonization transformation** is defined with the objective to integrate data from various sources. In order to overcome the heterogeneity of data, the data is restructured according to the common data model. The common data model facilitates to data interoperability based on a common set of terms concepts and relations across different data sources. In order to build the common data model, different open data models like FIWARE Smart Data Model<sup>5</sup>, SAREF<sup>6</sup>, BRICK SCHEMA<sup>7</sup> are reused.

The **data semantic enrichment** main goal is to develop a semantic module for cross-platform, cross-domain data-information-knowledge interoperability. The well-established vocabularies related to the building domain should be covered. The exogenous context-based data like weather, geographical, or energy networks related data is required to enrich the semantic meaning of pilot data.

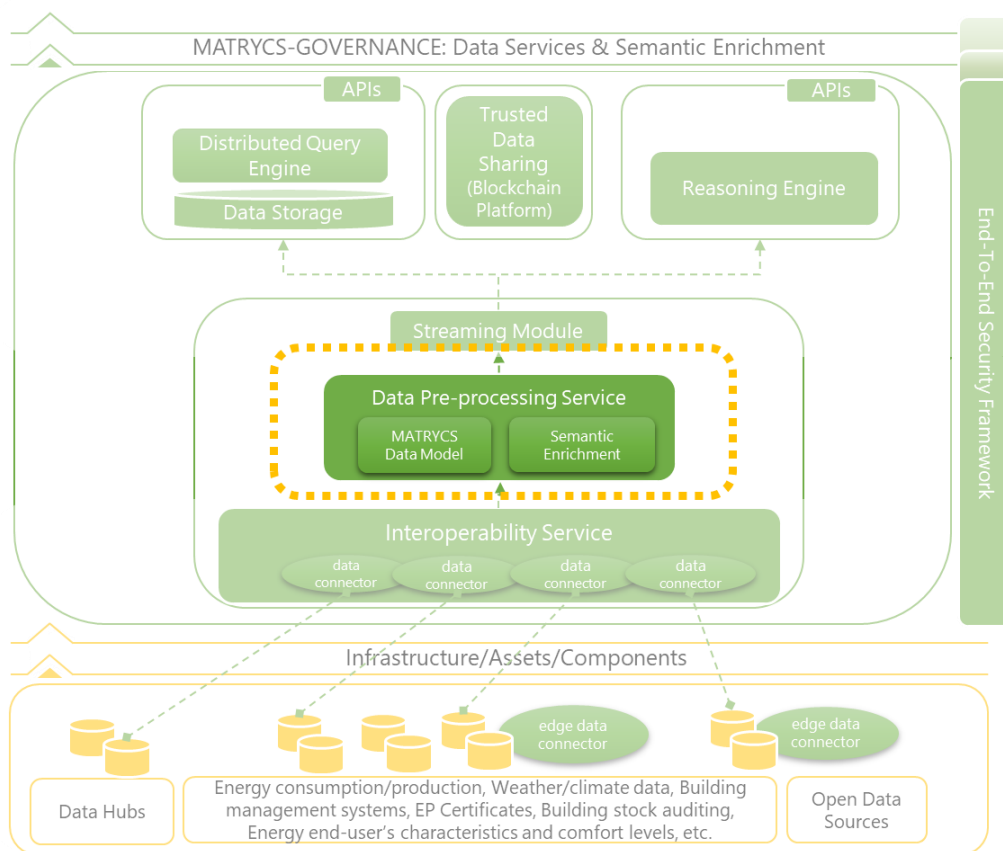


Figure 4: Data Pre-processing Service

<sup>5</sup> <https://www.fiware.org/developers/data-models/>

<sup>6</sup> <https://saref.etsi.org/>

<sup>7</sup> <https://brickschema.org>

### 2.2.3 Streaming module

The Data Streaming Module (*Figure 5*) manages dynamically the frequency rate of the data streaming allowing re-routing the data to the different MATRYCS-GOVERNANCE services for the subsequent in-memory processing of the high latency near real time data.

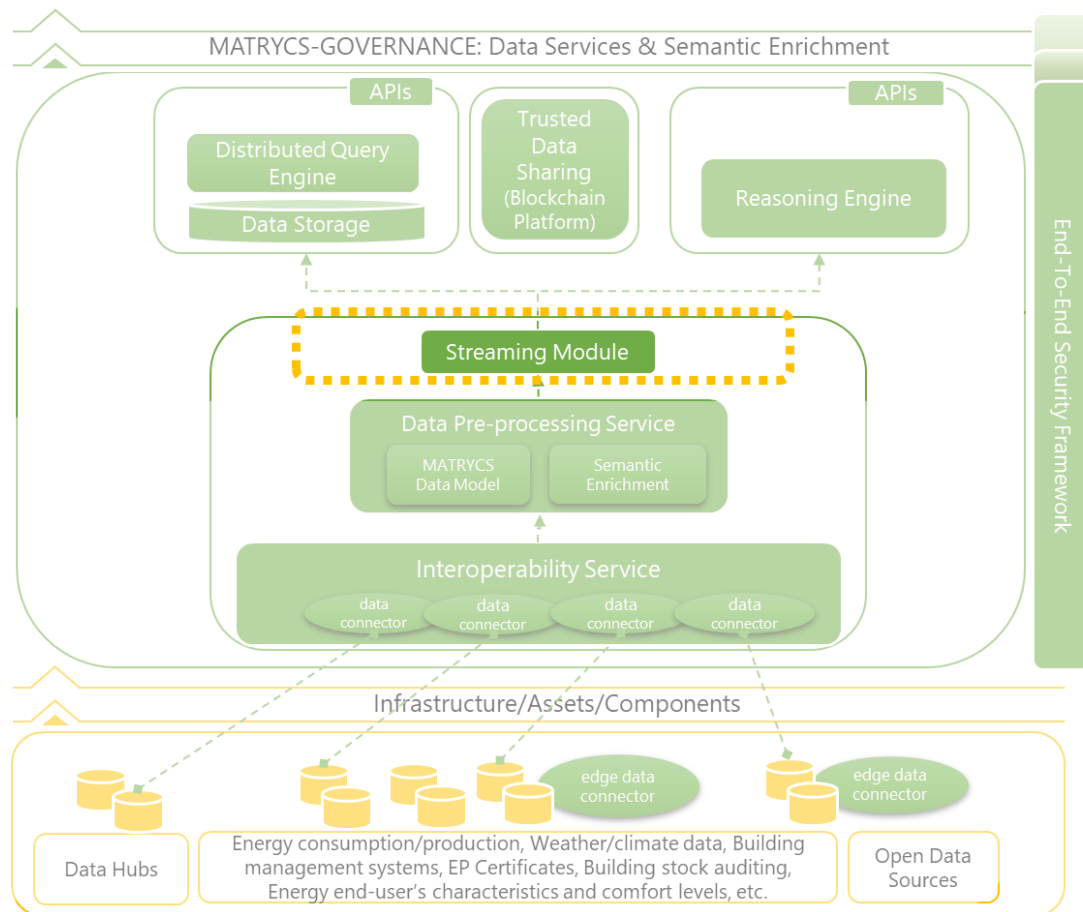


Figure 5: Streaming Module

### 2.2.4 Data Storage

The Data Storage module (*Figure 6*) enables local storage of datasets at the edge and in the cloud. It focuses on providing means for indexing and efficient data querying with respect to high availability and scalability.

Data storage, as one of the steps in the data pipeline, is one of the crucial building blocks that provides data persistence and its availability for further modification and usage. To this end, the Data Storage will be divided into three parts:

- **Temporary Storage (also Staging Area):** The purpose of the Temporary Storage is to keep recent data readily available for a short time period before being deleted or written to persistent storage. Any data processing must be carried out inside the predefined time window related to persistence settings.

- **Structured/Semi-structured Data Storage:** This storage will provide access to transformed data for further analytical purposes.
- **Networked/Cloud File storage:** This storage will provide data persistence and a cost-effective storage for longer retentions.

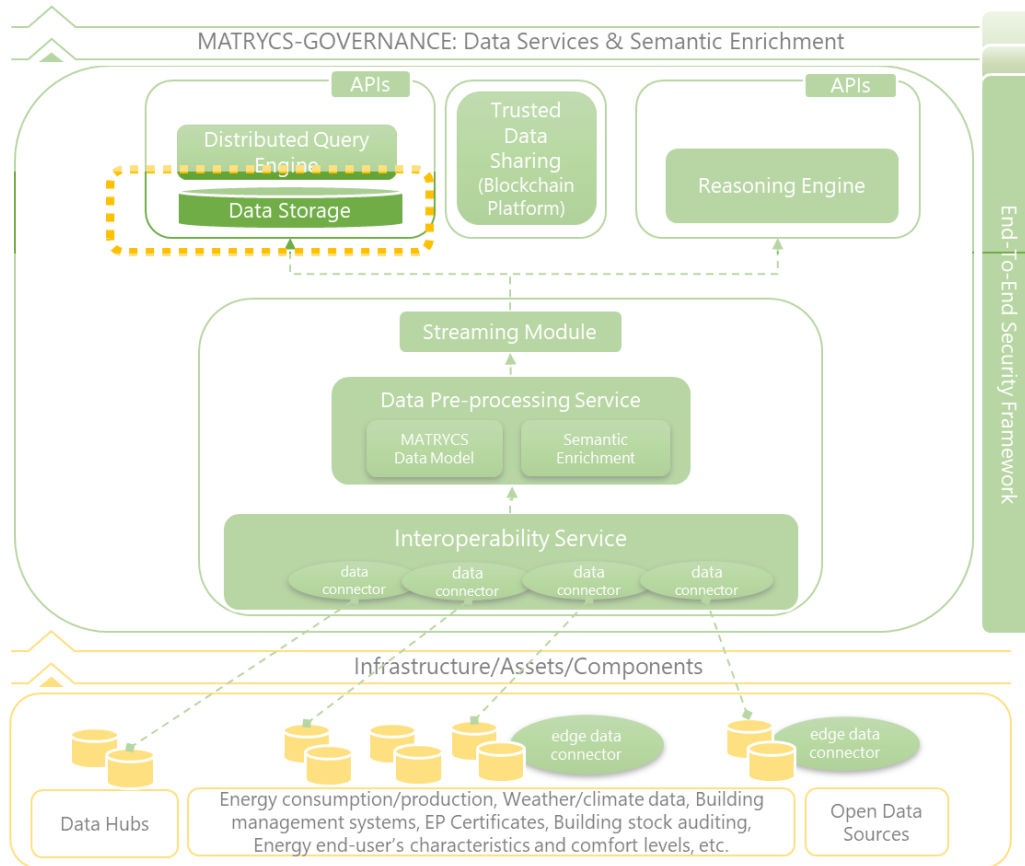


Figure 6: Data Storage

### 2.2.5 High Performance Distributed Query Engine

The High-Performance Distributed Query Engine (*Figure 7*) is built on top of Data Storage, providing a distributed query execution engine that enables high-performance data retrieval and processing to the upper layers of MATRYCS-GOVERNANCE (i.e., services, AI). In addition and in order to utilize NoSQL-oriented approaches, the engine will enforce data privacy and enable minimal network usage by reducing the data footprint where possible. Whereas the Distributed Query Engine will enable running complex interactive analytic queries on big data, real-time constraints will be considered.

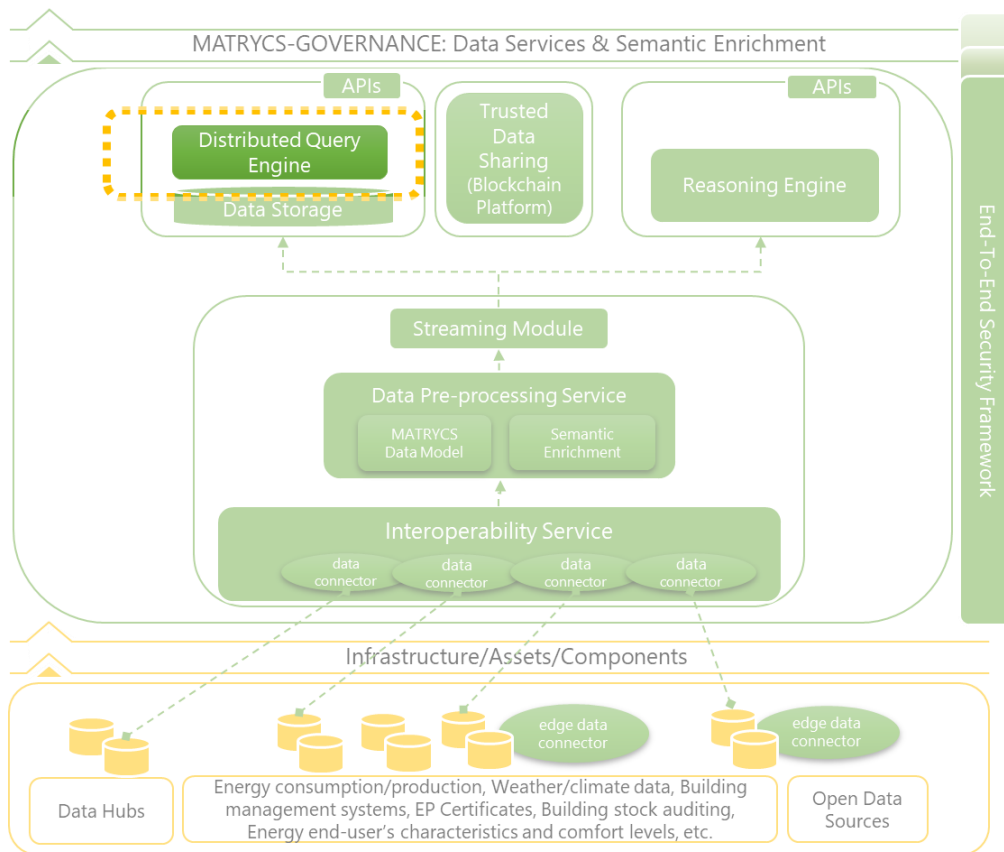


Figure 7: Distributed Query Engine

### 2.2.6 Reasoning Engine

The Reasoning Engine (*Figure 8*) is a mechanism that will provide intelligent querying, insights and search capabilities by leveraging the available knowledge for the Digital Twin and the building analytics services. Furthermore, it will provide functionalities for adding more data to the existing knowledge base. The Reasoning Engine module is capable of consuming data that are produced to Kafka topics from the Streaming module. These data could be in both JSON<sup>8</sup> and RDF<sup>9</sup> formats and when received they are persisted to Reasoning Engine's graph database. The graph database that is used is a powerful inference engine that enables graph functionalities over entities and connections for extracting new insights and patterns from datasets.

<sup>8</sup> <https://www.json.org/json-en.html>

<sup>9</sup> <https://www.w3.org/RDF/>

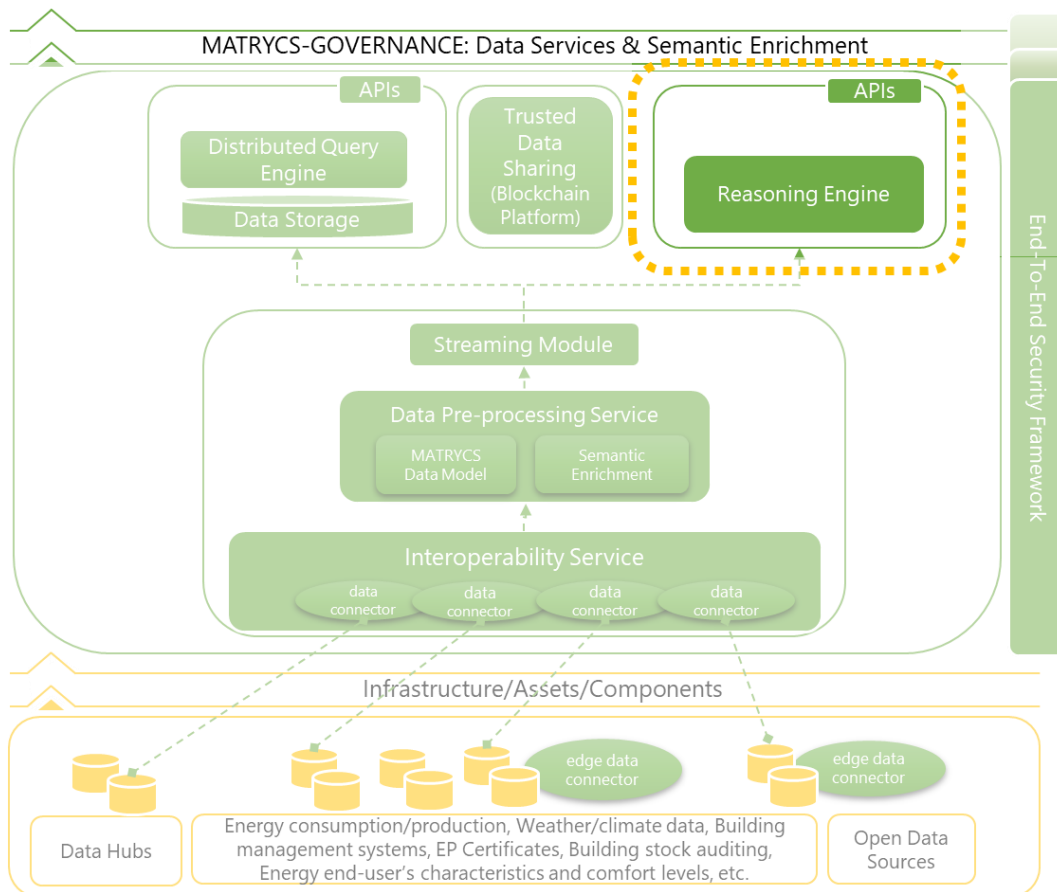


Figure 8: Reasoning Engine

### 2.2.7 Trusted Data Sharing (DLT/Blockchain)

The Trusted Data Sharing module is dedicated to use the DLT/Blockchain technology to ensure an integrity and trustiness of the data shared in MATRYCS and include in the MATRYCS Data Storage, as indicated in the Figure 9. The primary purpose of the blockchain technologies, used in this module is to remove the need for intermediaries and replace them with a distributed network of digital users who work in partnership to verify transactions and safeguard the integrity of the ledger. Use of blockchain in the data sharing model of MATRYCS have these main key advantages:

- **Traceability and data storage:** decentralised and distributed system that becomes a secure way to track changes in information and data over time.
- **Trust of data:** the creation of trust among untrusted participants and among the other providing the possibility to maintain data trustiness during time.
- **Peer-to-peer transactions:** the absence of intermediaries promotes a more transparency data sharing.

In addition, we should note that the Blockchain is a distributed ledger, based on a shared and distributed database, containing a log of transactions in chronological order. Transactions are grouped into blocks and chained through cryptographic hashes into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work.

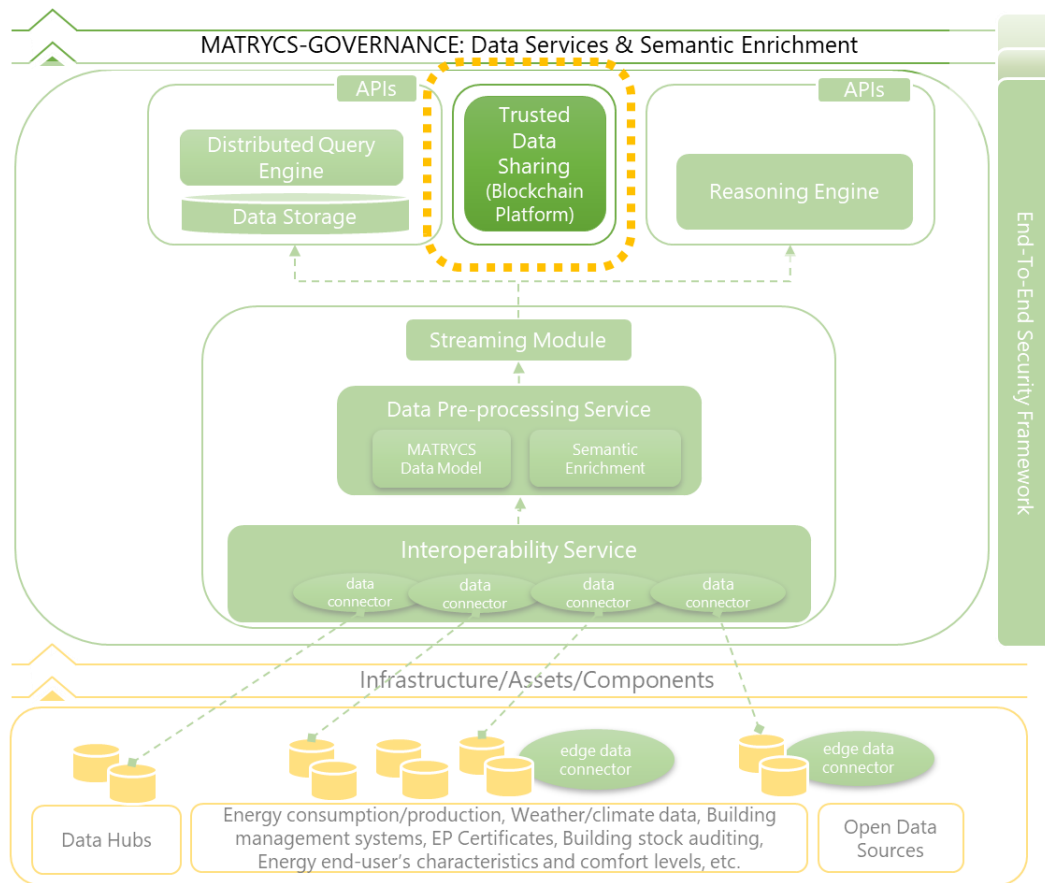


Figure 9: Trusted data sharing

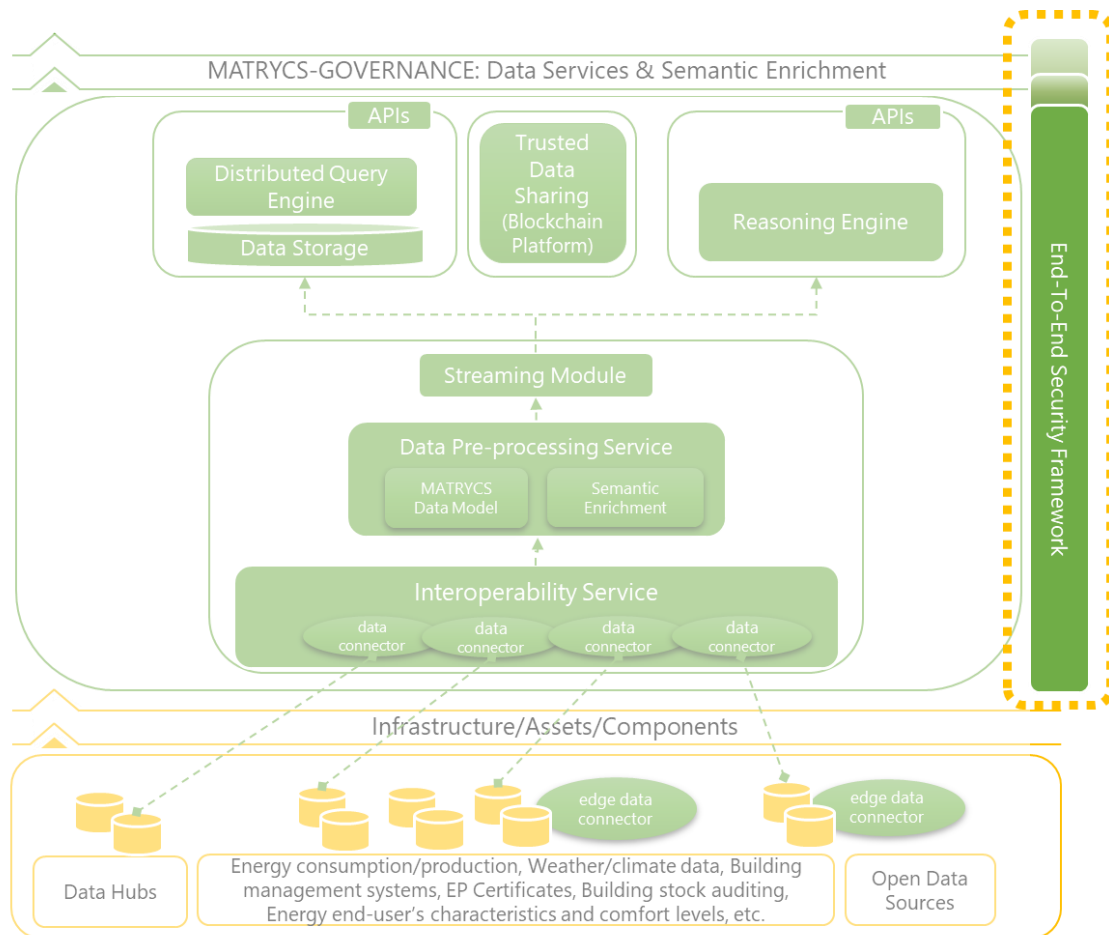
### 2.2.8 End-to-End Security framework

The End-to-End Security Framework (Figure 10) is the security layer for the whole MATRYCS platform (MATRYCS-GOVERNANCE, MATRYCS-PROCESSING, MATRYCS-ANALYTICS) focusing on the holistic design and implementation of the following aspects:

- Privacy.
- Anonymization.
- Authentication, authorization, auditing.
- Encryption.
- Software vulnerabilities/flaws detection and mitigation.

The End-to-End Security Framework will provide high-level security and fine-grained access control over all MATRYCS resources (i.e., data, services, end-user applications, etc.) in addition to encrypted communication between the resources. In this way, the platform and information will be kept safe, thus enhancing the trustfulness of the system. Appropriate mechanisms for maintaining and reinforcing legal/security policies will be employed. In relation to Data Semantic Enrichment, High Performance Distributed Query Engine and Data Storage, the End-to-End Security Framework will ensure that transferred data conforms to processing and security constraints, focusing on data encryption (at rest or in transit) and anonymization.

In the context of the MATRYCS project, the End-to-End Security Framework encompasses and relates to several entities; these entities are associated with infrastructure/assets (e.g., servers), services focusing on artificial intelligence, machine learning and big data, MATRYCS end-users, and data that are generated and shared.



**Figure 10: End-to-End Security Framework**



## 3 MATRYCS Data Governance solution

### 3.1 Overview Data Governance solution

Figure 11 shows an overview of the 1<sup>st</sup> technology release of the MATRYCS-GOVERNANCE layer implementation with the envisaged technological solutions.

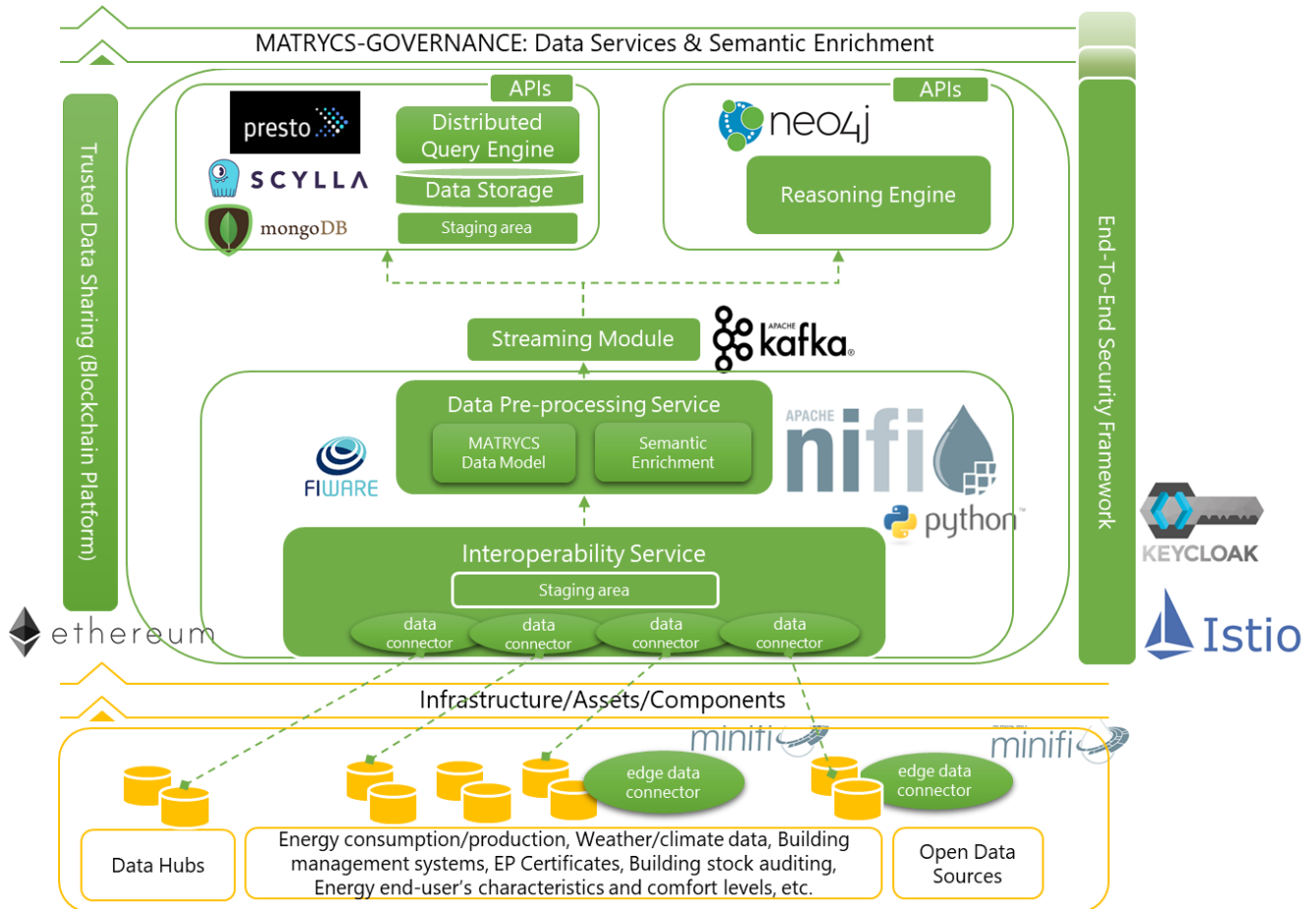


Figure 11: Overview of the MATRYCS-GOVERNANCE Solution

### 3.2 Interoperability Service Module

#### 3.2.1 Interoperability implementation description

The main technology used for the implementation of the Interoperability Service Module is Apache NiFi<sup>10</sup>, an open-source tool under the Apache License 2.0, which supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic with the aim to automate modern dataflow in the era of the Internet of Things and Big Data paradigms.

In the context of the MATRYCS project, a cluster of three NiFi instances has been set up to ensure data

<sup>10</sup> <https://nifi.apache.org/>

exchange of large volumes and varieties of both real-time and historical data made available by the MATRYCS data providers (i.e. Smart Meters, Sensors, IoT devices, Building Management Systems, Energy Performance Contracts/Certificates, legacy systems). The ability of Apache NiFi to work with different data sources, APIs, databases, services and SFTP servers makes it the right tool to ensure data exchange and interoperability of the heterogeneous data to be managed in the MATRYCS project. Apache NiFi has been chosen as “...NiFi was built to automate the flow of data between systems...”; therefore, within the context of Apache NiFi, the term dataflow is used to mean the automated and managed flow of information between systems. In the MATRYCS project context, different NiFi processors, the main building blocks in Apache NiFi processing, have been configured to create specific data connectors with the different datasets provided with different communication interfaces. Data connector integrations can easily provide a solution to transform data into meaningful information and provide a building block for generating valuable insights. This segmented approach to data management will no longer cause disruption in the analysis, but a holistic overview that will enable to disrupt data silos, poor communication, and quality of insights.

Furthermore, Apache Minifi component<sup>11</sup>, a sub-project of Apache NiFi, is used to fully realise the most modern distributed computing paradigm extending the cloud to the edge, by offering both local computational power and storage capacity to services and functions that may need ultra-low latency, as well as local processing without leveraging on core cloud remote services. In the MATRYCS project context, the Minifi component will be configured at the building edge-level to cover data exchange mechanisms at the EDGE level.

The Interoperability Service module also includes an SFTP server for MATRYCS data providers who cannot provide direct access to APIs, IoT devices, Databases, or their legacy systems to share their datasets.

Nginx<sup>12</sup> is used as a reverse proxy. It is one of the servers with the best performance characteristics. Its modular architecture, fault tolerance, support for HTTP/2<sup>13</sup>, load balancing, etc. make it an obvious choice to be implemented in the MATRYCS project.

Finally, raw data integrated into the MATRYCS Governance by the Interoperability Service Module is stored in a shared file system to be processed by the “Data pre-processing and semantic enrichment” described in the *section 3.3*.

### 3.2.2 Interoperability Data Connectors

Data connectors enable the Interoperability Service Module to combine and integrate various sources of data within the MATRYCS-GOVERNANCE layer with a holistic approach able to disrupt data silos and the poor communication.

In the MATRYCS project different NiFi processors, the main building blocks in Apache NiFi data flow, have been configured to create specific data connectors with the different datasets provided with different communication interfaces.

A set of NiFi Process Groups (see *Figure 12*) have been created for each MATRYCS Data provider (e.g

---

<sup>11</sup> <https://nifi.apache.org/minifi/>

<sup>12</sup> <https://www.nginx.com/>

<sup>13</sup> <https://en.wikipedia.org/wiki/HTTP/2>

Pilots, Data HUBs, etc.) which include all relative data connectors (NiFi processors). The Process Group represents a particular composition of input ports, processors, output ports and connections among them. Each processor group solves a specific task regarding data flow, contributing to the overall goal of data gathering and data integration in an interoperable manner from all assets, infrastructures, data Hubs and datasets handled in the MATRYCS project.

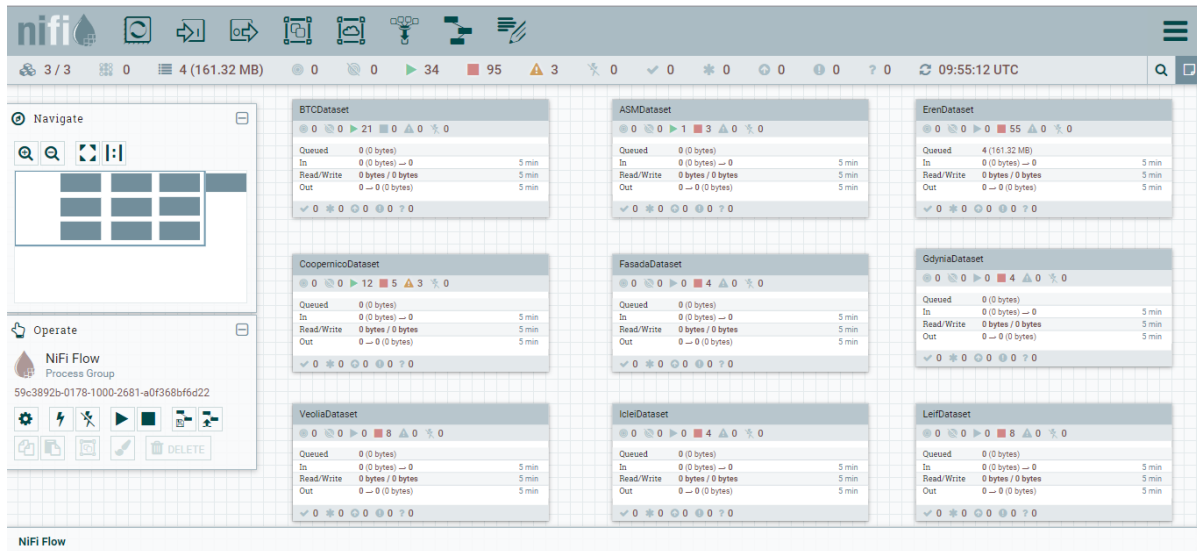


Figure 12: Apache NiFi processor groups

### 3.2.3 Technological components

#### 3.2.3.1 Overall description

The Interoperability Service module has been deployed by using and combining the following tools and frameworks:

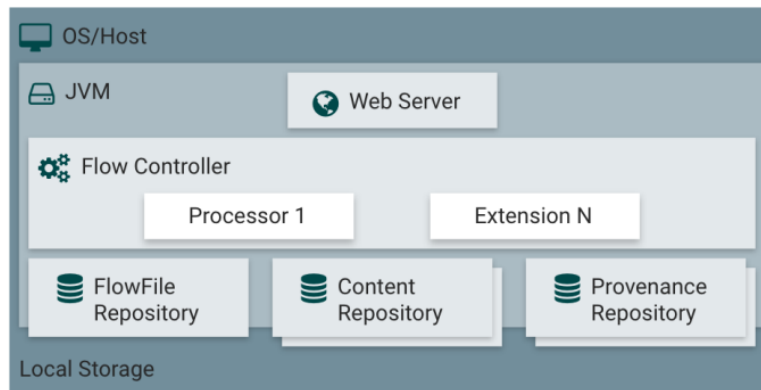
- Apache NiFi Cluster
- Apache Zookeeper
- Apache MiniNiFi
- Nginx

#### Apache NiFi and Zookeeper

Apache NiFi is an open-source tool under the Apache License 2.0 which provides the following high-level capabilities:

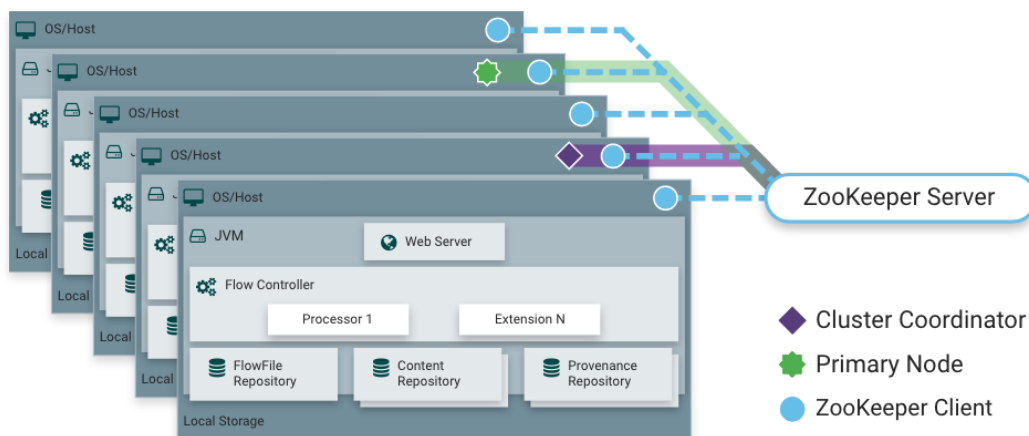
- User-friendly Web User Interface
- Highly configurable (loss tolerant, low latency, high throughput, dynamic prioritization, guaranteed delivery)
- Data Provenance (dataflow tracking from beginning to end)
- Designed for extension
- Secure (SSL, SSH, HTTPS, encrypted content, etc.)

In the *Figure 13* the architecture of Apache NiFi is presented. The core concepts and further details on the architecture of Apache NiFi can be found at <https://nifi.apache.org/docs.html>.



**Figure 13: Apache NiFi Architecture**

Apache NiFi also includes the ability to operate within clusters by using the Zero-Leader Clustering paradigm to improve cluster performances. Each node in a NiFi Cluster has its own dataset, but they all run the same preconfigured tasks on their data. So, each node will produce different results because the data are different, although the task is the same. It is important to note that for the correct execution of the NiFi cluster, Apache Zookeeper<sup>14</sup> is used. Apache Zookeeper appoints one NiFi node to be the Cluster Coordinator, and that node will keep track of the status of other nodes in a cluster, as well as connection/disconnection of nodes (see *Figure 14*). Moreover, Zookeeper makes one node to be the Primary Node. User Interface is used for easy and intuitive interaction with each node.



**Figure 14: Apache NiFi Cluster**

<sup>14</sup> <https://zookeeper.apache.org/>

### Apache NiFi

Apache NiFi is developed with the aim to enable data collection as close to the source of data creation as possible. It can run at the edge level, on hardware with different main intend, such as various IoT agents. It is lightweight and easy to deploy. NiFi supports a number of protocols for data gathering, enables transformations of data, as well as routing to more powerful tools that can do more complicated conversions, harmonisation, data curation, etc. Because of these aspects, NiFi is often integrated with NiFi, which is the case in the MATRYCS project too.

### Nginx

Nginx is a free and open-source web server and a reverse proxy server, capable of serving more than 10,000 simultaneous connections. It has TLS/SSL support, load balancing, support to serve static files (see *Figure 15*). It can serve name-based and IP-based servers and have access control depending on the client address.

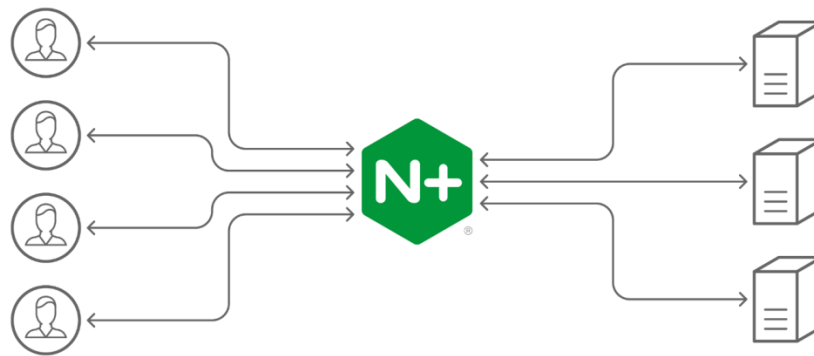


Figure 15: Nginx Load-balancing Reverse Proxy

#### 3.2.3.2 Deployment approach

As shown in the *Figure 16*, different tools and frameworks described in this section have been deployed and configured to work closely together to create the Interoperability Service module environment.

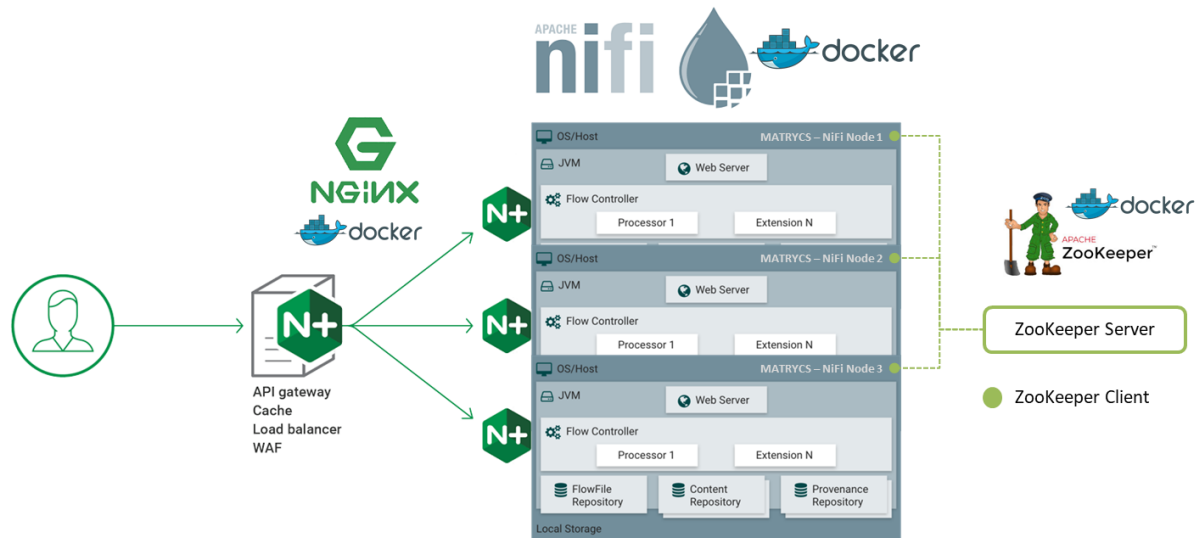


Figure 16: Interoperability Service Module

### Apache NiFi cluster

During this first phase of the project, a NiFi cluster composed of three different nodes has been configured to guarantee the scalability of the platform and to be easily scaled up based on the future project's needs.

The characteristic of this specific configuration also means that it is not necessary to indicate a priori which is the master node and its slaves serving services such as the user interface and the Cluster Coordinator, but that each node can serve them independently.

Apache NiFi Cluster has been deployed using Docker technologies<sup>15</sup> and each NiFi node has its own Docker container. The official Apache NiFi Docker Hub repository is <https://hub.docker.com/r/apache/nifi>.

Docker Compose, as a tool for configuring and running multi-container Docker applications, was an obvious choice for Apache NiFi cluster deployment. In particular Docker Compose<sup>16</sup> with a docker-compose.yml file has been used to apply all the configuration details to the docker containers of each NiFi node, and to configure the connections among them, as well as volumes and environment variables used in the Docker container. Volumes have been used to preserve the configurations, states, and data of each container even if the failure of the container happens. In *Table 1* the docker compose file used for the Apache NiFi node configuration is displayed.

All docker containers used to deploy the Apache NiFi cluster have been secured with TLS/SSL certificates, both on the server and client side, enabling mutual authentication.

<sup>15</sup> <https://www.docker.com/>

<sup>16</sup> <https://docs.docker.com/compose/>

**Table 1: NiFi node Docker Compose file**

```

version: "3"

services:
  nifi:
    image: nifi_python3_pip3
    hostname: matrycsf.ml
    container_name: matrycs

    ports:
      - 1026:1026
      - 8080:8080    - 8084:8084

    environment:
      - NIFI_WEB_HTTP_PORT=8080
      - NIFI_REMOTE_INPUT_SOCKET_PORT=1026
      - NIFI_CLUSTER_IS_NODE=true
      - NIFI_CLUSTER_NODE_PROTOCOL_PORT=8084
      - NIFI_ZK_CONNECT_STRING=192.168.111.106:2181
      - NIFI_ELECTION_MAX_WAIT=1 min
      - NIFI_REMOTE_INPUT_HOST=matrycsf.ml
      - NIFI_CLUSTER_ADDRESS=matrycsf.ml
      - NIFI_JVM_HEAP_MAX=2g

    volumes:
      - /dati/dockers/apache-nifi:/opt/nifi/nifi-current/ls-target
      - /dati/dockers/apache-nifi/tmp:/tmp

    extra_hosts:
      - "matrycsf.tk:192.168.111.106"
      - "matrycsf.ga:192.168.111.107"

```

### Apache Zookeeper

Apache ZooKeeper is the service for enabling distributed synchronisation, management and group services of the MATRYCS Apache NiFi cluster. Confluent ZooKeeper, the NiFi cluster orchestrator, is deployed using Docker technologies. A specific ZooKeeper docker container has been configured to enable the correct behaviour of the NiFi cluster. This dockerized solution of Apache NiFi and ZooKeeper has proved to be stable and easily transferable to other systems.

### Apache MiNIFI

As described in section 3.2.3.1 Apache MiNiFi can be deployed at edge level and it can run as agent



software to collect data directly to its source. It means that the environments that will host the MiNiFi agent could be different and not known a-priori. For these reasons, despite the possibility of installing the various dependencies and downloading and installing the binaries files from NiFi's official repository, Docker technologies have been identified as the best solution to deploy MiNiFi in the MATRYCS project. A guide how to manage the dockerisation of Apache MiNiFi can be found at the official GitHub repository<sup>17</sup> of the Apache Software foundation.

## Nginx

In the MATRYCS context, the NGINX Reverse Proxy has been deployed by using Docker technologies (docker-compose method) in order to give a public entry-point to the Apache NiFi cluster. TLS/SSL certificates have also been created and implemented for Nginx, guaranteeing encrypted connections. In this way, secure connections are established between each of the applications presented and each client. The official Nginx service on Docker Hub, available at [https://hub.docker.com/\\_/nginx](https://hub.docker.com/_/nginx), has been used to deploy the service.

In *Table 2* the docker-compose.yml file used to deploy the NGINX Reverse Proxy for the MATRYCS Interoperability Module. It creates the NGNIX docker container and maps to an external volume the configuration file and log files.

**Table 2: NGINX Docker Compose file**

```
version: "3"

services:
  nginx:
    image: nginx:latest
    container_name: nginx

    ports:
      - "80:80"
      - "8085:8085"
      - "8086:8086"
      - "8087:8087"

    volumes:
      - /dati/dockers/nginx/conf:/etc/nginx/conf.d/
      - /dati/dockers/nginx/log/access.log:/var/log/nginx/access.log
      - /dati/dockers/nginx/log/error.log:/var/log/nginx/error.log
```

In *Table 3* the NGINX configuration file is presented.

<sup>17</sup> Apache Mini NiFi GitHub repository: <https://github.com/apache/nifi-minifi>



Table 3: NGINX configuration file

```

upstream matrycsf.tk {
    server 217.172.12.158:8087;
}
server {
    listen 8087;
    server_name matrycsf.tk;
    server_tokens off;
    location / {
        proxy_pass http://192.168.111.107:8080;
        proxy_pass_header Server;
        proxy_hide_header X-Powered-By;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_set_header X-Nginx-Proxy true;
        client_max_body_size 10m;
        client_body_buffer_size 128k;
        proxy_connect_timeout 90;
        proxy_send_timeout 90;
        proxy_read_timeout 90;
        proxy_buffer_size 4k;
        proxy_buffers 4 32k;
        proxy_busy_buffers_size 64k;
        proxy_temp_file_write_size 64k;
    }
}

```

### 3.2.4 Interaction with other Data Governance components

In the MATRYCS Governance, the Interoperability Service module interacts with the:

- MATRYCS assets, infrastructures, components
- Data Pre-processing and Semantic Enrichment component

Interoperability Service module interacts with the assets, infrastructures and components of the MATRYCS data providers deploying dedicated data connectors able to integrate their datasets with different standard.



The datasets integrated in the MATRYCS platform will be made available in a file staging area of the Interoperability Service module to be processed by the Data Pre-processing & Semantic Enrichment layer.

## 3.3 Data pre-processing and semantic enrichment

### 3.3.1 Data pre-processing and semantic enrichment implementation description

The Data pre-processing & semantic enrichment module is divided into three modules:

- Data pre-processing service
- MATRYCS data module
- Semantic enrichment

The data pre-processing service will cover the functionality of anonymization, and data curation. The MATRYCS data module transforms LSP data to the common data model, which relies on the standard building data model and well-established vocabularies. The semantic enrichment will provide a semantic annotation to the data and establish an ontology to cover genetic semantic information in the building domain.

### 3.3.2 Technological components

#### 3.3.2.1 Overall description

The Data pre-processing service is integrated with Apache NiFi, which is designed to automate the flow of data between software systems. Apache NiFi follows the concept of extract, transform, load (ETL), which extracts data from heterogeneous source and transform them into a proper format, which in MATRYCS is JSON format. JSON<sup>18</sup> is a lightweight data-interchange format. It is human-readable and easy for machines to parse and generate. These properties make JSON ideal for the output format of the data pre-processing service. Additionally, Apache NiFi provides a web-based user interface, which makes the monitoring intuitive and improves usability. One of the key features, which is used in data pre-processing service, is the ExecuteScript NiFi processor<sup>19</sup>. The ExecuteScript supports Clojure<sup>20</sup>, ECMAScript<sup>21</sup>, Groovy<sup>22</sup>, Lua<sup>23</sup>, Python<sup>24</sup>, and Ruby<sup>25</sup>. This provides the possibility of individual data processing applying suitable language. Additionally, different possible anonymization technologies: directory replacement, masking out, data encryption, and custom anonymization, can be integrated in

<sup>18</sup> <https://www.json.org/json-en.html>

<sup>19</sup> <https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-scripting-nar/1.5.0/org.apache.nifi.processors.script.ExecuteScript/index.html>

<sup>20</sup> <https://clojure.org/>

<sup>21</sup> <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>

<sup>22</sup> <https://groovy-lang.org/>

<sup>23</sup> <https://www.lua.org/>

<sup>24</sup> <https://www.python.org/>

<sup>25</sup> <https://www.ruby-lang.org/en/about/>

the Data pre-processing service. Therefore, the MATRYCS data module and Semantic enrichment module are also integrated with Apache NiFi. This makes the data pipeline between each module more stable and easier to interconnect.

### 3.3.2.2 Deployment approach

In this 1<sup>st</sup> technology release, we are focusing on the MATRYCS Data Model and Semantic Enrichment since the output is the input for Data storage and Reasoning engine.

Before starting with the development approach, the distinction between data model and ontology, which will be defined by the MATRYCS Data Model and Semantic Enrichment module, should be described.

The data model and the ontology are compared in four aspects<sup>26</sup> (see Table 4). Both describe the structure of the data but, while the data model is more task specific and implementation oriented, the ontology is more generic. The more generic the ontology is, the more interoperable and reusable it is.

A compromise between generality and specificity should be reached. In the MATRYCS project we propose to reuse already existing standard ontologies and data models. The Data model is processed by normalization, which reduces data redundancy and improve data integrity. Therefore, the volume of data in Data storage is reduced. FIWARE Smart Data Model is applied to normalize the data, which provides different data model in smart domain. We also propose modifying then to meet specific requirements while keeping the ontology as generic as possible by avoiding modifying specific data. The data model is modified by our specific requirement and the ontology will keep generic by avoiding modifying the specific data.

**Table 4: Comparison data model with ontology**

Aspects	Data Model	Ontology
Operation levels	Less abstract	More abstract
Expressive power	High	Low
User, purpose and goal relatedness	High	Low
Extendibility	Low	High

The Common Data Model design approach is defined as follows:

- Determine the domain and scope of the data model
- Consider reusing existing schemas
- Define the common entity and entity hierarchy, which should cover all the different LSP data.
- Define the common attribute in the defined class
- Map each pilot data to the Common Data Model

<sup>26</sup> SPYNS, Peter; MEERSMAN, Robert; JARRAR, Mustafa. Data modelling versus ontology engineering. ACM SIGMod Record, 2002, 31. Jg., Nr. 4, S. 12-17

As mentioned before, the Semantic Enrichment module will establish an ontology. An ontology defines a common vocabulary for researchers who need to share information in a domain. It provides a common understanding of the structure of information among people or software agents. Developers or other data users can easily use the data without specific domain knowledge.

The approach<sup>27</sup> to develop an ontology can be summarized as follows:

- Determine the domain and scope of the ontology
- Consider reusing existing ontologies
- Define the common class and class hierarchy
- Define class properties.
- Define the facets of the properties, which can be considered as “the properties of the properties”
- Create instances

The raw data integrated through the Interoperability Service module is handled by the Data Pre-processing module which will process and transform it according to the defined MATRYCS common data model. These data modelling processes are developed with Python and integrated into Apache NiFi. The implementation details can be found in the *section 5.1.2*. The development of the data pre-processing is yet to begin and will be described in the 2<sup>nd</sup> *Technology Release* of this deliverable.

### 3.3.3 Interaction with other Data Governance components

As shown in the overview data governance implementation (see *Figure 2*), the raw data is provided by the Interoperability Service module. The raw data can be in different data formats. The data pre-processing module transforms the data format to JSON and processes other necessary data cleaning functions. The top-level interaction contains two parts. The first part is the modified data with the data storage. This connection is through the Streaming module (see *section 3.4*). The second part puts the RDF data into the Reasoning Engine. This is also through the Streaming module. The huge advantage in this structure is that the Reasoning Engine can not only get the semantic data, but also the modified common data. Therefore, the Reasoning Engine may have a better chance of building advanced semantic services.

## 3.4 Streaming module

### 3.4.1 Streaming module implementation description

As streaming module in the MATRYCS project, the Confluent data streaming platform<sup>28</sup> that relies on Apache Kafka<sup>29</sup> is used. This platform enables high-volume real-time streaming, publishing and subscribing to the messages, storing and processing of data. Confluent is also chosen as it gives the most complete distribution of Kafka. This platform enhances the experience of both operators and

<sup>27</sup> NOY, Natalya F., et al. *Ontology development 101: A guide to creating your first ontology*. 2001

<sup>28</sup> <https://www.confluent.io/>

<sup>29</sup> <https://kafka.apache.org/>

developers.

### 3.4.2 Technological components

The Confluent platform is implemented on-premises. The whole set of tools that revolve around Apache Kafka and help in working with and managing the streaming platform is installed. ZooKeeper component is essential in doing the management of the Kafka brokers. Apache Kafka is an event streaming platform. As Kafka facilitates real-time data analysis, the connected MATRYCS modules can have up-to-date information and make predictions regarding different subjects. In the *Figure 17* a graphical representation of the general Kafka Architecture.

As Kafka is designed for distributed environment, it is well suited for possible expansion to the network of nodes, and therefore easily adaptable and scalable for the needs of the MATRYCS project.

#### Kafka Architecture

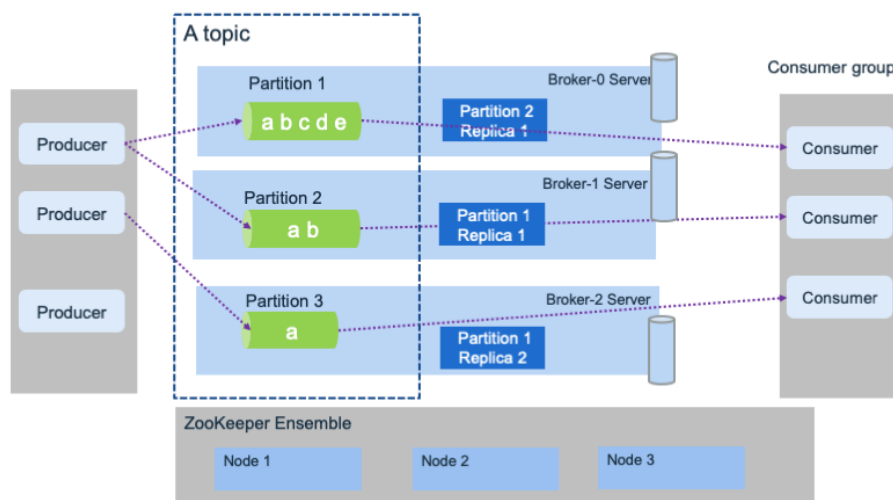


Figure 17: Apache Kafka Architecture

Apache Kafka is able to manage a huge number of data sources, processes the stream of their information, and organises them into topics. Messages that are received are often called events or records. There are two main concepts: a function called Producer and another interface called Consumer. The Producer is an interface that enables sending data to topics. Topics represent the ordered, segmented data, a kind of database known as Kafka Topic Log. Topics are partitioned, which is important for the environment with multiple Kafka brokers. Messages that have the same event key are written to the same partition.

In the *Figure 18*, the topic with four partitions is shown. Also, two producers are publishing messages to the same topic. They are connecting over the network and they are writing the events to the topic, and particularly, writing into partitions of the selected topic. Events that have the same key are written to the same partition, as it is already stated.

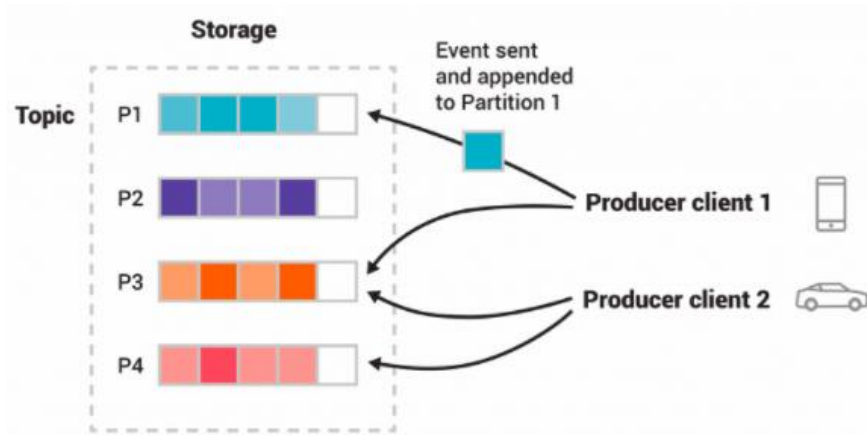


Figure 18: Kafka with multiple partitions for the topic and two producers

Consumers are the clients that can subscribe to (read and process) messages. Consumer enables the events to be read and information to be sent to other modules. Kafka retains all published messages for a certain period that can be configured. It does not make difference if messages are consumed or not, they all will be discarded at the same time, after they are not needed anymore. Storing streams durably and reliably is one of the main features of Kafka.

Besides the Producer API, the Consumer API, and the Kafka Streams API, Kafka APIs<sup>30</sup> include the Admin API and the Kafka Connect API. The Admin API manages all Kafka objects, such as topics and brokers. The Kafka Connect API have hundreds of connectors, such as connectors to relational databases, and other import/export connectors.

The next step includes enabling only SSL connections to Apache Kafka, with mutual authentication which will contribute to much better security and privacy.

### 3.4.2.1 Overall description

Apache Kafka is a central part of the Confluent platform. Other parts include Control Center for topic management, monitoring and analysing, ksqldb<sup>31</sup> for event stream processing, Kafka Connect Datagen source connector for generating mock data, as well as ZooKeeper<sup>32</sup>, Schema Registry<sup>33</sup>, HTTP REST Proxy<sup>34</sup> for Kafka.

Control Center has a Web UI that natively runs on port 9021 and can be used for Kafka topics creation. Topics are selected from the Cluster submenu and a new topic with a specific name and a number of partitions can be created. From Control Center we also can see Overview, Brokers, ksqldb, Consumers, etc.

Apache Kafka is a distributed streaming platform for handling Big Data, evolved from a simple messaging queue. Today it represents a scalable, fault-tolerant system for powerful streaming. Besides robust Web UI and REST API, Confluent Kafka enables users to use CLI in order to create topics,

<sup>30</sup> <https://docs.confluent.io/platform/current/connect/references/restapi.html>

<sup>31</sup> <https://ksqldb.io/>

<sup>32</sup> <https://zookeeper.apache.org/>

<sup>33</sup> <https://docs.confluent.io/platform/current/schema-registry/develop/api.html>

<sup>34</sup> <https://docs.confluent.io/platform/current/kafka-rest/api.html>

publish and consume messages.

Apache Kafka facilitates event streaming. It gathers data in a real-time manner from data sources like databases, IoT devices, sensors, software applications. In this way, Apache Kafka enables a continuous data flow and data integration. Apache Kafka event streaming can be applied to a big number of use cases, such as payments and financial transactions in real-time, monitoring of cars, shipments etc., gathering and analysing of sensor data, reaction to customers' orders, enabling interaction among different divisions of a selected company. The most prominent use cases, that are used in the MATRYCS project also, are to be the bases of data platforms, event-driven architecture, as well as microservices.

### 3.4.2.2 Deployment approach

Confluent Platform can be deployed using several different methods:

- Cloud provider
- Using an archive, DEB or RPM package
- Docker technologies

Confluent Operator enables automation of deployment of the platform as a cloud-native, stateful container on Kubernetes and OpenShift.

In this first technology release, Confluent platform has been deployed on-premise using Docker technologies with the docker-compose method for the deployment.

The official Docker image is being hosted on Docker HUB at the following link <https://hub.docker.com/u/confluentinc>. It is also possible to extend and rebuild images from the official GitHub repository <https://github.com/confluentinc>.

#### **Docker-compose method**

In *Table 5*, the docker-compose file used to deploy all the necessary parts of the Confluent platform, together with Apache Kafka.

**Table 5: Docker-compose file for Apache Kafka**

```
version: '3'
services:
  ...
  broker:
    image: confluentinc/cp-server:6.0.1
    hostname: broker
    container_name: broker
    depends_on:
      - zookeeper
    ports:
```

- "9092:9092"

- "9101:9101"

*environment:*

*KAFKA\_BROKER\_ID: 1*

*KAFKA\_ZOOKEEPER\_CONNECT: 'zookeeper:2181'*

*KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT*

*KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://broker:29092,PLAINTEXT\_HOST://217.172.12.158:9092*

*KAFKA\_METRIC\_REPORTERS: io.confluent.metrics.reporter.ConfluentMetricsReporter*

*KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1*

*KAFKA\_GROUP\_INITIAL\_REBALANCE\_DELAY\_MS: 0*

*KAFKA\_CONFLUENT\_LICENSE\_TOPIC\_REPLICATION\_FACTOR: 1*

*KAFKA\_CONFLUENT\_BALANCER\_TOPIC\_REPLICATION\_FACTOR: 1*

*KAFKA\_TRANSACTION\_STATE\_LOG\_MIN\_ISR: 1*

*KAFKA\_TRANSACTION\_STATE\_LOG\_REPLICATION\_FACTOR: 1*

*KAFKA\_JMX\_PORT: 9101*

*KAFKA\_JMX\_HOSTNAME: localhost*

*KAFKA\_CONFLUENT\_SCHEMA\_REGISTRY\_URL: http://schema-registry:8081*

*CONFLUENT\_METRICS\_REPORTER\_BOOTSTRAP\_SERVERS: broker:29092*

*CONFLUENT\_METRICS\_REPORTER\_TOPIC\_REPLICAS: 1*

*CONFLUENT\_METRICS\_ENABLE: 'true'*

*CONFLUENT\_SUPPORT\_CUSTOMER\_ID: 'anonymous'*

...

Besides the Docker Compose method for deploying Apache Kafka, Docker Compose is used for all other components in the Confluent platform. The Docker-compose file consists of:

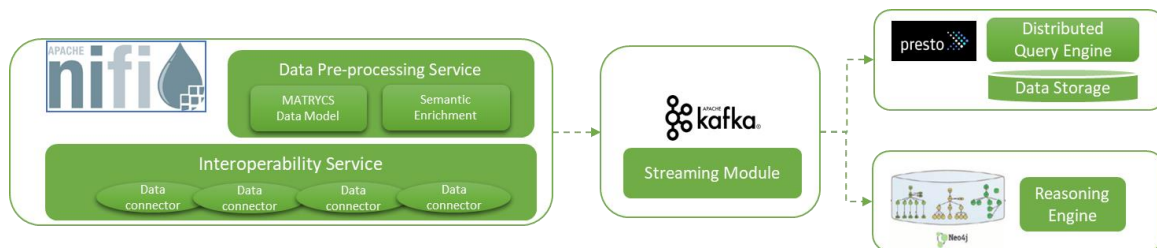
- Name of the service. The containers can be named too.
- Image for each service is from the official Docker Hub of Confluent
- Ports are mapped to the inner ports on which the services are running.
- Multiple environment variables are set. There are other ways of configuring environment variables, such as .env files, that also can be explored.
- Volumes are used where is needed to persist particular data when new Docker Containers are created/rebuilt or even if the failure of Docker container happens, mostly for ZooKeeper and Kafka.
- Bridge networking is utilized as the platform is deployed on a single host.





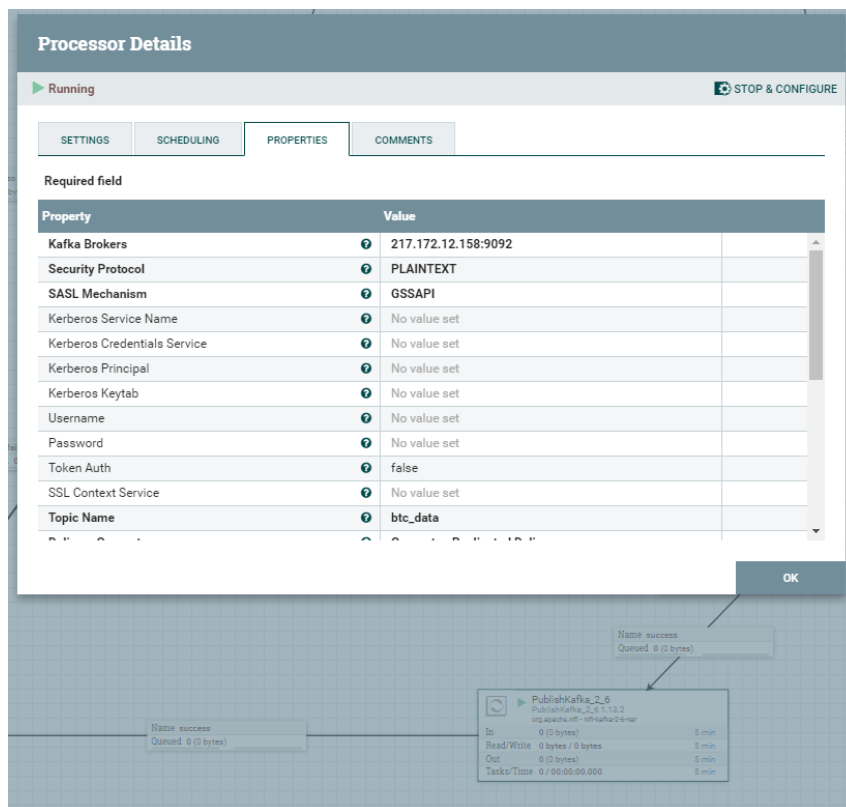
### 3.4.3 Interaction with other Data Governance components

After data acquisition phase in the Interoperability module and data curation phase in the Data Pre-Processing & Semantic Enrichment module, the MATRYCS data is delivered to the Streaming module with the aim of sending it as a data stream to both the Reasoning Engine and the Data Storage components. In particular, after finalising data curation, cleansing, anonymisation, semantic annotation and modelling, Data Pre-Processing & Semantic Enrichment module sends the messages to the appropriate topics in Kafka created for each dataset (see *Figure 19*).



**Figure 19: Interaction of the Streaming Module with other components**

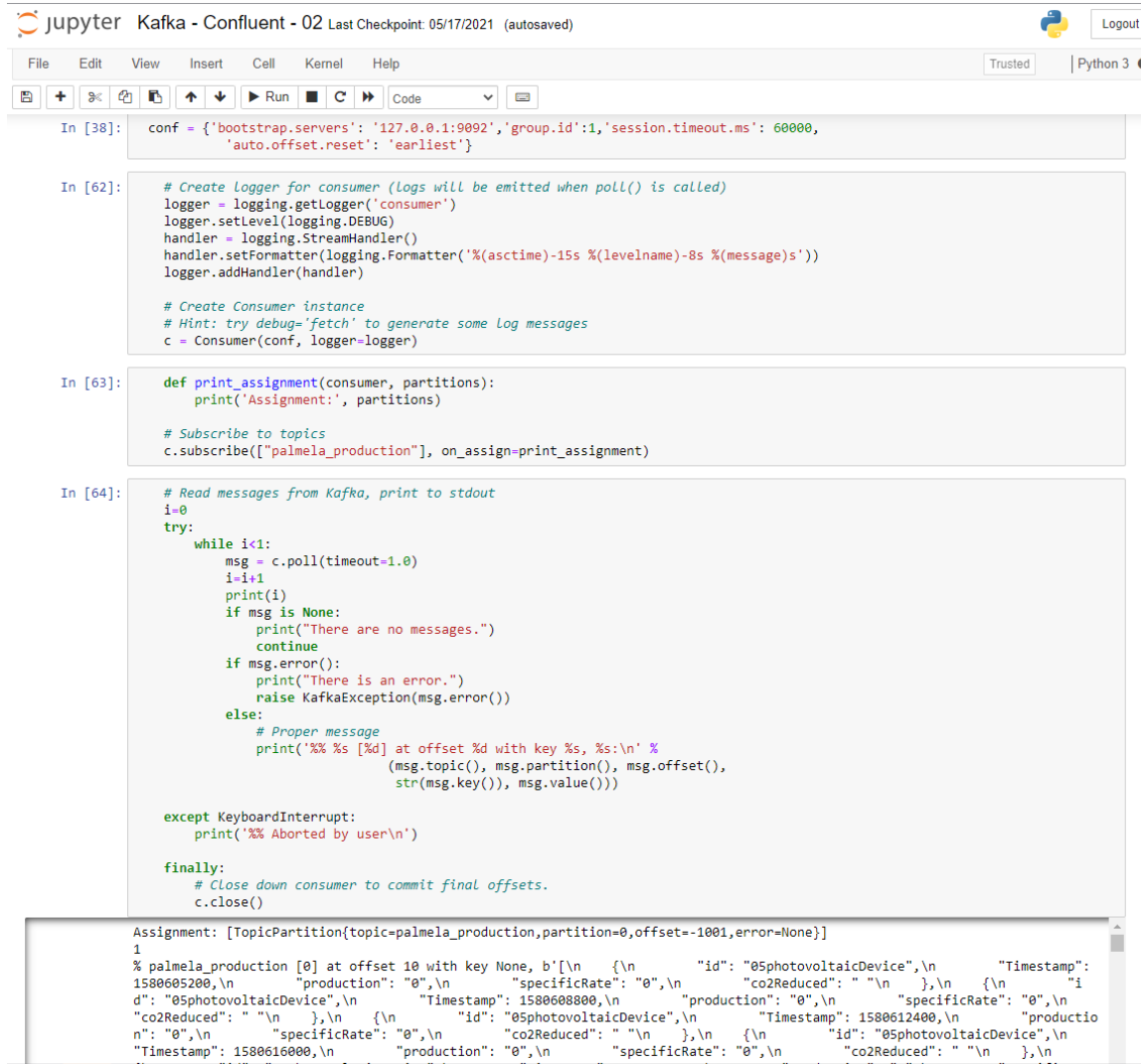
Dedicated NiFi processors have been used for the Kafka connection (see *Figure 20*) and they have been configured to work with a particular topic and at a scheduled time. This configuration enables consumers to easily find the data of interest and get the messages from the topic.



**Figure 20- NiFi PublishKafka processor configuration**

Kafka data streaming is also used by the Evaluation module and Serving framework implemented in the MATRYCS-PROCESSING layer. They can consume messages from the specific topics and make further tuning of ML models and evaluation, as well as predictions based on the consumed messages. In these modules, the appropriate Python libraries are installed and configured that enable high-level

connection and consumption of messages from a specific topic (see *Figure 21*). Several libraries are taken into consideration and their performances are under evaluation. These would lead to more efficient integration, processing and real-time streaming of gathered data sets.



The screenshot shows a Jupyter Notebook titled "Kafka - Confluent - 02" with a last checkpoint of "05/17/2021 (autosaved)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help) and a toolbar with icons for file operations, running, and code execution. The notebook contains four input cells:

- In [38]:** A configuration dictionary for the Kafka consumer:
 

```
conf = {'bootstrap.servers': '127.0.0.1:9092', 'group.id': 1, 'session.timeout.ms': 60000, 'auto.offset.reset': 'earliest'}
```
- In [62]:** Code to create a logger for the consumer and a consumer instance:
 

```
# Create logger for consumer (Logs will be emitted when poll() is called)
logger = logging.getLogger('consumer')
logger.setLevel(logging.DEBUG)
handler = logging.StreamHandler()
handler.setFormatter(logging.Formatter('%(asctime)-15s %(levelname)-8s %(message)s'))
logger.addHandler(handler)

# Create Consumer instance
# Hint: try debug='fetch' to generate some Log messages
c = Consumer(conf, logger=logger)
```
- In [63]:** A function to print the assignment and subscribe to the 'palmela\_production' topic:
 

```
def print_assignment(consumer, partitions):
    print('Assignment:', partitions)

# Subscribe to topics
c.subscribe(["palmela_production"], on_assign=print_assignment)
```
- In [64]:** A loop to read messages from Kafka and print them to stdout:
 

```
# Read messages from Kafka, print to stdout
i=0
try:
    while i<1:
        msg = c.poll(timeout=1.0)
        i=i+1
        print(i)
        if msg is None:
            print("There are no messages.")
            continue
        if msg.error():
            print("There is an error.")
            raise KafkaException(msg.error())
        else:
            # Proper message
            print('%s [%d] at offset %d with key %s, %s:\n' %
                  (msg.topic(), msg.partition(), msg.offset(),
                   str(msg.key()), msg.value()))
except KeyboardInterrupt:
    print('%s Aborted by user\n' % msg.topic())
finally:
    # Close down consumer to commit final offsets.
    c.close()
```

The output of the notebook shows the assignment of the 'palmela\_production' topic to partition 0 at offset -1001. It then displays a sample message from the topic, including fields like 'id', 'Timestamp', 'production', 'specificRate', and 'co2Reduced'.

Figure 21: Confluent Kafka package in Jupyter environment

## 3.5 Data Storage

### 3.5.1 Data Storage implementation description

Implementation of data storage area will be based on Apache Kafka<sup>35</sup> message broker system (Streaming module), which will be the main source of data. For this, a special microservice will be put in place. Its purpose will be fetching data from a Kafka cluster and distributing the data accordingly to the data storage. Microservice will be implemented in Golang<sup>36</sup> language, which is an open source, simple, reliable and efficient language, thus making it a well-suited language for such microservices

<sup>35</sup> <https://kafka.apache.org/>

<sup>36</sup> <https://golang.org/>

and application programming interfaces (APIs).

The microservice will have a Kafka consumer as the main connection endpoint towards Kafka. Messages will be pulled from Kafka when they arise for a specific Kafka topic. Kafka consumer will understand other Kafka related features needed for normal operation, such as topic partitions, Kafka topic offset as marker of already consumed messages, etc.

As messages are taken from a Kafka Topic, they will be stored in a temporary storage area for aggregation and preparation for the Reasoning engine to perform further transformation of data.

This microservice will be built and set up using the most minimal image in Docker (also called scratch Docker image) for optimal resource usage. As such, it may be hosted on a Linux server under systemd<sup>37</sup> suite or pushed to some orchestration system such as Kubernetes<sup>38</sup>.

For storage itself, Linux filesystem will be used as a shared storage for the staging area, whereas for the persistent storage, cloud storage will be used. The storage solution will depend on the selection of the public cloud provider.

An important part of storage is currently ScyllaDB<sup>39</sup> as a real-time big data, column based, distributed database built for modern applications. Initially, ScyllaDB has been set up as Docker-based container; later on, this setup would be done in a clustered manner, providing higher scalability and performance.

However, at M9, after some integration and performance test with the MATRYCS-PROCESSING layer and in particular with the Data Feed Module, a new technological solution based on the document-oriented database MongoDB<sup>40</sup> instead of the columnar database ScyllaDB, is being evaluated in parallel to improve the performance of this module. Preliminary integrations and tests with MongoDB have been performed in the Data Feed Module of the MATRYCS-PROCESSING layer and reported in D4.1 – MATRYCS-PROCESSING (1<sup>st</sup> technology release)<sup>41</sup>. This change of the technological solution for the data storage module will be reported in the 2<sup>nd</sup> technology release.

## 3.5.2 Technological components

### 3.5.2.1 Overall description

As mentioned above, from a technical perspective, the following technologies, systems, and languages will be used:

- Linux server with Ext4 or XFS filesystem for hosting storage and services.
- Golang language for the development of the microservice and APIs required.
- Docker as a containerization tool for easier deployment and maintenance.

---

<sup>37</sup> <https://systemd.io/>

<sup>38</sup> <https://kubernetes.io/>

<sup>39</sup> <https://www.scylladb.com/>

<sup>40</sup> <https://www.mongodb.com/>

<sup>41</sup> D4.1 – MATRYCS-PROCESSING (1<sup>st</sup> technology release)

### 3.5.2.2 Deployment approach

Golang based microservice will be deployed as a Docker container. Such a container may be deployed either on a bare-metal server, cloud virtual server or other technologies like Kubernetes.

The Dockerfile for the data storage microservice can be seen in *Table 6* whereas the ScyllaDB setup approach is illustrated in *Table 7*.

**Table 6: Data storage microservice Dockerfile**

```
FROM golang:1.16.4-alpine as builder
RUN mkdir /build
COPY . /build/
WORKDIR /build
RUN go mod download -x
RUN CGO_ENABLED=0 GOOS=linux go build
# run
FROM scratch
WORKDIR /app
COPY --from=builder /build/endpoint /usr/bin/endpoint
ENTRYPOINT ["endpoint"]
```

**Table 7: Docker-compose file for ScyllaDB**

```
version: '3'
services:
  scylla:
    image: scylladb/scylla:4.1.0
    container_name: scylla-dev
    restart: always
    port:
      - "9042:9042"
    healthcheck:
      test: ["CMD-SHELL", "nodetool status"]
      interval: 20s
      timeout: 10s
      retries: 20
    logging:
```

```
driver: "json-file"
```

```
options: max-size: 50m
```

### 3.5.3 Interaction with other Data Governance components

Interaction with other data governance components will be possible via a direct call or data push from the data storage microservice. The required interactions are under consideration but will be developed using a distributed query engine based on Presto<sup>42</sup>. Presto will allow users to interact with data on shared storage, temporary storage or in DB engine. Additionally, there will be predefined direct access to the staging area for the Reasoning Engine to access data that needs further transformation.

## 3.6 Reasoning Engine

### 3.6.1 Reasoning Engine implementation description

Reasoning engine is the component which function is to extract insights and identify hidden patterns from data. It takes as input a stream of data the output of the Streaming module, produced through a Kafka topic. In order to consume these streams of data, the Reasoning Engine must be a Kafka consumer, subscribed on specific topics for incoming messages. To leverage its inference functionalities, a powerful database suitable for reasoning is being used, Neo4j graph database is being used. After receiving the incoming messages, the Kafka consumer enables several scripts, written in Cypher<sup>43</sup> query language (the query language of Neo4j<sup>44</sup>) to insert the data into Neo4j structures, for later querying and inference. The reasoning engine provides REST APIs, for submitting queries directly to the Graph database, enabling requested parties to gain access into the graph entities and connections. The programming language that has been used developing the first version of Reasoning Engine is Python 3.7 the Kafka Consumer is developed also in Python 3.7, using Confluent Kafka Python library. The Neo4j 4.3 edition is used and the official Neo4j Python driver<sup>45</sup> for executing Cypher queries in Python. The REST functionalities of the Reasoning Engine are served using the Flask Rest framework. In the figure below the overview of the Reasoning Engine solution is demonstrated:

<sup>42</sup> <https://prestodb.io/>

<sup>43</sup> Cypher, <https://neo4j.com/developer/cypher/>

<sup>44</sup> Neo4j, <https://neo4j.com/>

<sup>45</sup> Neo4j Python, <https://neo4j.com/docs/api/python-driver/current/>

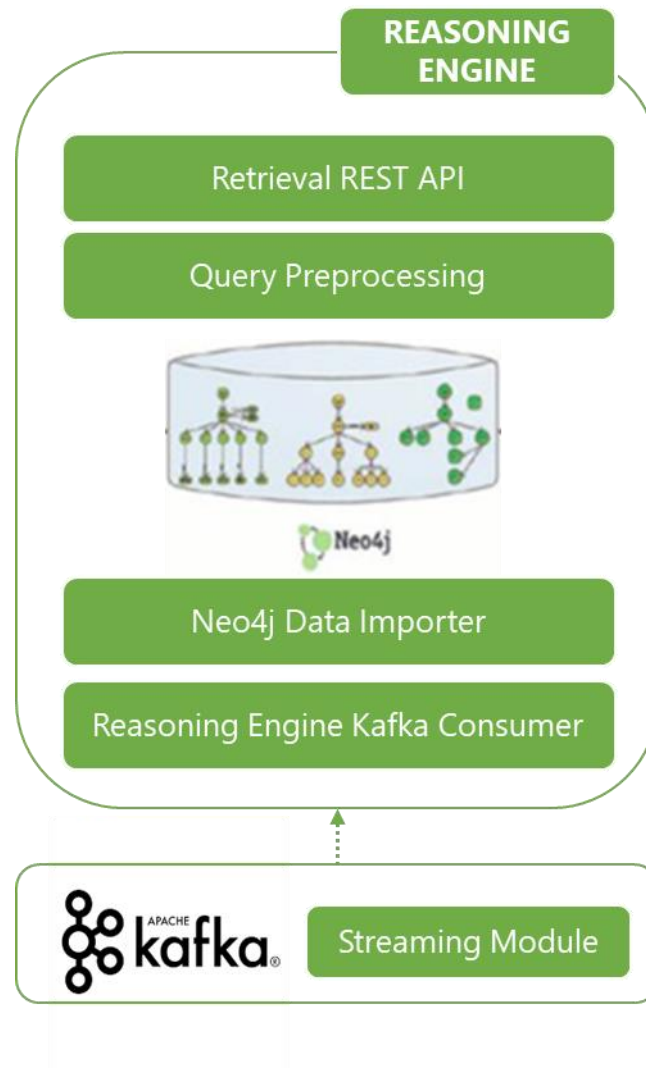


Figure 22: Reasoning Engine Solution and sub-modules

## 3.6.2 Technological components

### 3.6.2.1 Overall description

As shown in the *Figure 22*, the Reasoning engine consists of 3 layers. The data import layer, the Neo4j graph database and the Retrieval REST API. In the first layer, the incoming data are being consumed from the Reasoning Engine's Kafka Consumer. The Kafka consumer then, uses the Neo4j Data importers to upload, the data into Neo4j graph database (using the Cypher programming language and the relevant Neo4j Python driver) and creates the respective entities and connections. The second layer, Neo4j graph database, contains all the data in entities and connections. The format that Neo4j stores the data, is suitable for inference functionalities and reasoning over the data by leveraging graph functionalities. The third layer is the stage in which the application, by exposing a REST service, demonstrates the stored data to the end user. The end user submits Cypher queries to the retrieval engine, the query pre-processing module distributes that query to Neo4j graph database, and then the results are sent back to the end user in JSON format.

**Table 8: Example of Reasoning Engine REST API**

```
curl --location --request POST 'http://reasoning_engine:5000/query' \
--header 'Content-Type: text/plain' \
--data-raw '
MATCH      (b:Building)-[:has_municipality]-(m:Municipality      {name:      "\"ADRADA      (LA)\""})
WITH
MATCH (b)-[:has_primary_cons]->(prim_cons:PrimaryCons)-[:has_primary_label]->(r:Label {rating: "\"C\""})
RETURN b'
```

In the context of reasoning engine a recommendation service for LSP10, has been tested, taking as input building metrics, such as the building's total consumption and CO<sub>2</sub> emissions for heating hot water and electricity. The recommender system enables graph similarity measurements (e.g., graph cosine similarity) to return action plans and activities that similar buildings have followed for CO<sub>2</sub> and energy reduction. In *Table 9* the REST API used for receiving recommendations is presented.

**Table 9: Example of Reasoning Engine LEIF Recommendation Service**

```
curl --location --request POST 'http://reasoning_engine:5000/leif/service' \
--header 'Content-Type: application/json' \
--data-raw '{
  "heating_total_consumption": ${heating_total_consumption},
  "heating_co2_emission": ${heating_co2_emission},
  "hot_water_total_consumption": ${hot_water_total_consumption},
  "hot_water_co2_emission": ${hot_water_co2_emission},
  "electricity_total_consumption": ${electricity_total_consumption},
  "electricity_co2_emission": ${electricity_co2_emission}
}
```

### 3.6.2.2 [Deployment approach](#)

The Reasoning Engine and the Neo4j graph database are installed using Docker and docker-compose scripts. In *Table 10* the docker-compose.yml file that is used for the module's installation is presented.

**Table 10: Docker-compose file for Reasoning Engine**

```
version: '3'
volumes:
  neo4j_data:
services:
  neo4j:
    image: neo4j:latest
```

```

container_name: neo4j

hostname: neo4j

network_mode: "bridge"

volumes:

  - neo4j_data:/data

restart: always

ports:

  - "7474:7474"

  - "7687:7687"

environment:

  NEO4J_dbms_security_procedures_unrestricted: apoc.*

  NEO4J_apoc_import_file_enabled: "true"

  NEO4J_dbms_shell_enabled: "true"

  NEO4J_HEAP_MEMORY: 8G

  NEO4J_CACHE_MEMORY: 8G

  NEO4J_AUTH: neo4j/neo4j1

  NEO4J_dbms_memory_pagecache_size: 8G

  NEO4JLABS_PLUGINS: '["apoc", "n10s", "graph-data-science"]'

  NEO4J_dbms_memory_heap_initial_size: 8G

retrieval_engine:

  hostname: retrieval_engine

  container_name: retrieval_engine

  restart: always

  build:

  context: ".."

  dockerfile: config/Dockerfile

  ports:

    - 5000:5000

```

### 3.6.3 Interaction with other Data Governance components

The Reasoning Engine interacts by streaming data, with the Streaming module. More specifically Reasoning Engine leverages Faust Consumers. Faust<sup>46</sup> is a Python library that implements Kafka Streams in Python and these consumers receive data coming from various Kafka topics and persist these data to Neo4j graph database. The following schema demonstrates the interaction with Data

<sup>46</sup> <https://faust.readthedocs.io/en/latest/>



Streaming Module.

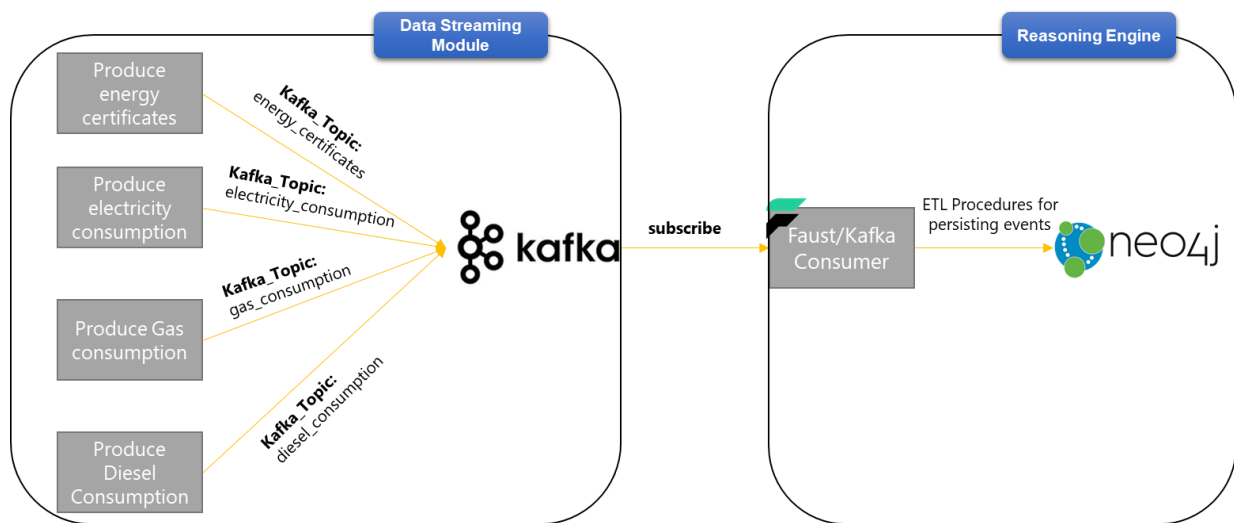


Figure 23: Reasoning Engine interaction with Data Streaming Module

The advantage of using Faust Consumer instead of plain Confluent Kafka consumers is that one consumer can be subscribed on multiple topics and by using Faust streams Windows<sup>47</sup> it is possible to batch insert multiple events. With this strategy the Reasoning engine throughput is minimized.

## 3.7 High Performance Distributed Query Engine

### 3.7.1 High Performance Distributed Query Engine implementation description

The High-Performance Distributed Query Engine is based on Presto<sup>48</sup>, an open-source and highly maintained project. It is one of the main components for accessing data in MATRYCS-GOVERNANCE. Presto allows usage of SQL for accessing and querying data in semi-structured and structured storage. It can be used to access data in files as well as in distributed remote sources. Furthermore, it can be used for accessing and querying data from multiple DB engines. Presto will allow querying over data stored in different locations, that being staging area, cloud storage, etc.

The implementation of High-Performance Distributed Query Engine will be done using the Docker ecosystem. This will serve as a more agile approach with respect to changes in different versions and future upgrades. Later on, this may be used to deploy High Performance Distributed Query Engine on various target system, from bare-metal to high-end cloud virtual servers.

### 3.7.2 Technological components

#### 3.7.2.1 Overall description

Presto is an open-source distributed SQL query engine for running interactive analytic queries against

<sup>47</sup> <https://faust.readthedocs.io/en/latest/userguide/tables.html>

<sup>48</sup> <https://prestodb.io/>

data sources of all sizes, ranging from gigabytes to petabytes. Presto allows querying data from multiple sources, including Hive<sup>49</sup>, Cassandra<sup>50</sup>, ScyllaDB<sup>51</sup>, relational databases or proprietary data stores.

### 3.7.2.2 Deployment approach

The deployment approach for Presto will be initially based on a Docker image and running everything as a standalone Docker container. This can be later deployed to various targets. The deployment Docker files can be seen in *Table 11* and *Table 12*.

**Table 11: Dockerfile for Presto**<sup>52</sup>

```
FROM openjdk:8-jre
EXPOSE 8080

MAINTAINER Greg Leclercq "ggreg@fb.com"
ARG PRESTO_VERSION=0.218
ENV PRESTO_PKG presto-server-$PRESTO_VERSION.tar.gz
ENV PRESTO_PKG_URL https://repo1.maven.org/maven2/com/facebook/presto/presto-
server/$PRESTO_VERSION/$PRESTO_PKG

ENV PRESTO_CLI_JAR_URL https://repo1.maven.org/maven2/com/facebook/presto/presto-
cli/$PRESTO_VERSION/presto-cli-$PRESTO_VERSION-executable.jar

# Install python to run the launcher script
RUN apt-get update
RUN apt-get install -y python less

# Download Presto package
# Use curl rather ADD <remote> to leverage RUN caching
# Let curl show progress bar to prevent Travis from thinking the job is stalled
RUN curl -o /$PRESTO_PKG $PRESTO_PKG_URL
RUN tar -zxf /$PRESTO_PKG

# Create directory for Presto data
```

<sup>49</sup> <https://hive.apache.org/>

<sup>50</sup> <https://cassandra.apache.org/>

<sup>51</sup> <https://www.scylladb.com/>

<sup>52</sup> <https://github.com/prestodb/f8-2019-demo/blob/master/Dockerfile>



```

RUN mkdir -p /var/lib/presto/data

# Add Presto configuration
WORKDIR /presto-server-$PRESTO_VERSION
RUN mkdir etc
ADD etc/jvm.config etc/
ADD etc/config.properties etc/
ADD etc/node.properties etc/
ADD etc/catalog etc/catalog

# Download Presto CLI
RUN mkdir -p bin
RUN curl -o bin/presto-cli $PRESTO_CLI_JAR_URL
RUN chmod +x bin/presto-cli

CMD bin/launcher.py run

```

**Table 12: Docker-compose file for Presto**

```

version: '3'
networks:
  config_matrycs_network:
    external: true
services:
  presto:
    container_name: presto
    hostname: presto
    networks:
      - config_matrycs_network
    restart: always
    build: .
    ports:
      - "8080:8080"

```



### 3.7.3 Interaction with other Data Governance components

The Data Governance components may interact with High Performance Distributed Query Engine in two ways:

- **Presto Client REST API** provides an interface for query submission and obtaining query results.
- **Staging Storage** enables direct interaction and usage from other components.

Additionally, Presto provides a graphical user interface to end users for running queries.

## 3.8 Trusted Data Sharing (DLT/Blockchain)

### 3.8.1 Trusted Data Sharing implementation description

The Trusted Data Sharing will be based on the blockchain implementation; the blockchain data sharing structure is a collection of blocks storing a list of valid transactions and the hash of the previous block (*Figure 24*). The linked structure makes tamper attempts evident since any changes in one of the already registered blocks will break the hash contained in the following blocks. Each transaction must have a valid cryptographic signature and, since the blocks are timestamped and ordered chronologically, this means that it is possible to assure the provenance property by tracking entries at the moment of their registration in the blockchain.

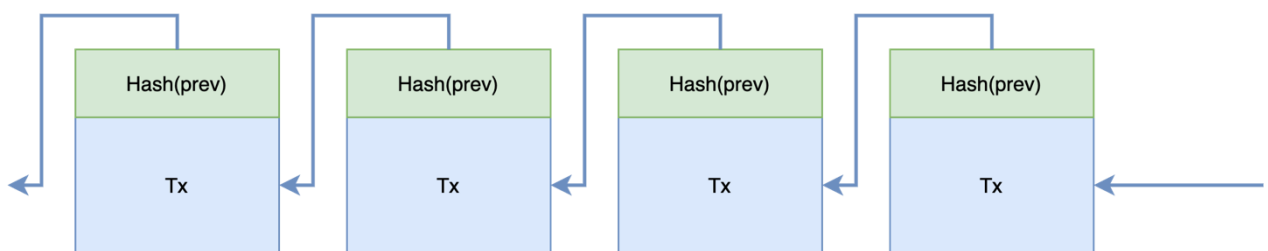


Figure 24: Blockchain data structure

The blockchain network for data sharing energy transactions can be modelled as a private permissioned network in which each user (please note that generally speaking, a user can be a consumer, this is more suitable for the present project, but also a producer of energy) is associated with a node of the blockchain network. Depending on the resources available and the degree of trust between peers within the network, different network configurations can occur. Optionally, it will be possible to rely on a public blockchain according to the specific context needs.

Nodes can be light or full, depending on their computational power and data storage capacity (e.g., nodes 2 and 5 in *Figure 25*). Moreover, some users may decide to delegate the control of transactions to another node they trust and not use their own node.

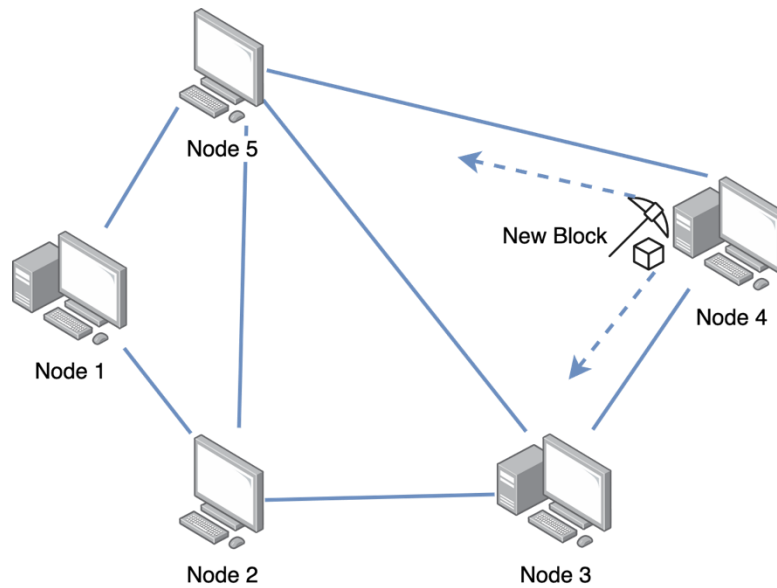


Figure 25: Blockchain network

Our solution will combine the on-chain data storage together with distributed (or centralized) databases for off-chain data storage according to an hybrid approach. In order to minimize the amount of data to be stored on the blockchain, raw data from the data sources will be stored using a distributed database: the blockchain will be then used periodically to store a fixed-length hash of the data received in the last time interval. The hybrid approach combines the scalability of a distributed NoSQL database with the blockchain, making tampering attempts evident, enabling provenance tracking, non-repudiability, and the use of self-enforcing smart contracts.

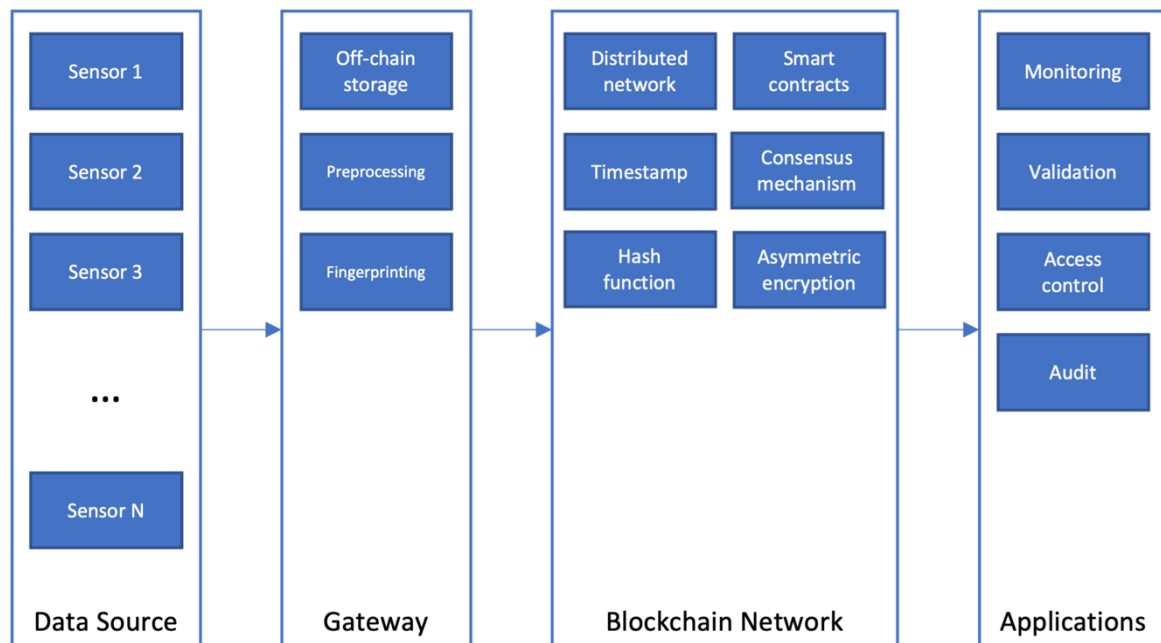


Figure 26: Trusted data sharing flow

Each data source (Figure 26) will be connected to a gateway, an embedded device which takes care of pre-processing, calculating digital fingerprints (hash codes) to assure data immutability via

notarization, and storing the data off-chain. It is also included the possibility that handle data from the Data Storage. A private network based on Ethereum<sup>53</sup> will be used as blockchain network. The Ethereum nodes provide the networking capabilities and the rules needed to determine consensus among them. The private network will be used as a platform for running smart contracts and register data with a timestamp.

Thanks to dedicated APIs, every node in the network can be queried and it is possible to build dedicated applications on top of this stack for monitoring the process, validating the off-chain data against the data stored on-chain, provide limited access for auditing purposes, or provide even finer-grained access to specific users for specific purposes, interacting with smart contracts deployed in the network.

## 3.8.2 Technological components

### 3.8.2.1 Overall description

The proposed solution will be based on Ethereum in term of blockchain consensus mechanism, in addition, connection with IoT devices will be implemented via MQTT<sup>54</sup>, a lightweight protocol suitable for unstable connections and low-power devices. Additionally, connection with NoSQL Data Storage will be supported for off-chain data handling.

#### ETHEREUM

Blockchains can be distinguished according to the consensus mechanism and programming capabilities. Considering the consensus mechanism, blockchains differ in the definition of the participation of nodes in the distributed network and the roles that they can perform. In particular, open blockchains and permissioned (or private) blockchains can be distinguished. Considering the programming capabilities, it is possible to differentiate between blockchains programmable via simple scripting and blockchains providing Turing-complete computational capabilities, enabling the creation of "smart contracts". Ethereum was the first blockchain supporting smart contracts and it is still the most notable example of Turing-complete programmable blockchain.

Ethereum was proposed by Vitalik Buterin in 2013 (Ethereum Whitepaper)<sup>55</sup> and further detailed by Gavin Wood in the 'yellow paper' (Ethereum: A Secure Decentralised Generalised Transaction Ledger (Wood))<sup>56</sup>. As the Ethereum website (Ethereum) reports, "Ethereum is a decentralized platform that runs smart contracts." These contracts run on the "Ethereum Virtual Machine" (EVM), a distributed computing network made up of all the devices running Ethereum nodes. Like other blockchains, Ethereum has a native cryptocurrency called Ether (ETH) and it has a double use: it is used as an incentive for the network "validators", but also to regulate the use of the blockchain computational resources.

Each operation on the Ethereum network has its own cost, determined by the computation, storage

---

<sup>53</sup> <https://ethereum.org/en/>

<sup>54</sup> <https://mqtt.org/>

<sup>55</sup> <https://ethereum.org/en/whitepaper/>

<sup>56</sup> <https://gavwood.com/paper.pdf>

and bandwidth required, and this cost is measured in a unit called Gas. gasLimit (the maximum amount of gas that can be spent) and gasPrice (the price-per-gas) are standard transaction parameters in Ethereum and also invoking a smart contract function both must be specified. Since smart contracts are run on a virtual machine, the presence of the gas price and limit have the purpose of preventing denial-of-service attacks. More specifically, is impossible to run infinite loops (due to the gas limit) and the non-optimised usage of system resources would be expensive due to the gas price. Gas and Ether are different concepts. Ether is a currency, while Gas is an internal transaction pricing mechanism that measures the amount of computational effort that it will take to execute certain operations (each line of code of an application requires a certain amount of gas to be executed in the Ethereum network). Gas is always paid in Ether. Having a separate unit allows maintaining a distinction between the actual valuation of the cryptocurrency (Ether), and the computational cost (Gas).

## MQTT

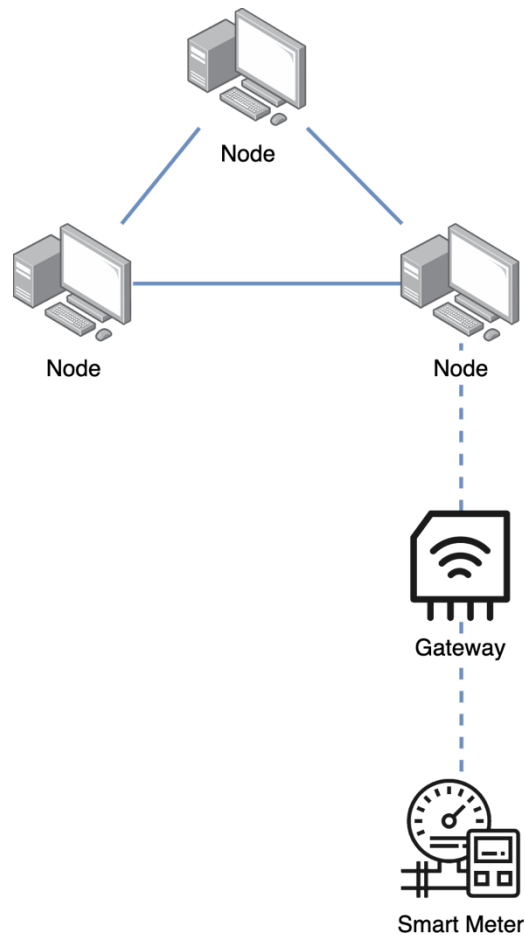
MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. MQTT today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc.

### 3.8.2.2 [Deployment approach](#)

The section 3.8.1 describes the role of the gateways in enabling IoT devices for blockchains. These are embedded devices (or similar, low-power devices) operating at the edge level connecting and managing the different IoT devices (*Figure 27*).

Connection with IoT devices will be implemented via MQTT, a lightweight protocol suitable for unstable connections and low-power devices. The gateway periodically will receive information about the status of the devices connected, pre-processes input data as needed, and store them to a distributed database in the cloud, using an asynchronous communication queue.

The gateway periodically calculates a fingerprint of the data received from the device and stores the result on the blockchain.



**Figure 27: IoT Gateway**

MQTT communication will be authenticated and each IoT device (*Figure 28*) assigned a gateway and a specific topic for its messages, so it will be possible to keep track of the origin of the information received.

Blockchain transactions will be signed with the private cryptographic key of the gateway that sent them, so that each reading will be associated in the smart contract with the corresponding gateway.

Off-chain data consistency verification will be performed by calculating on-the-fly the fingerprint of the data to be verified and comparing it with the fingerprint stored on the blockchain for the same time interval. If the two fingerprints match, the information is unaltered; if not, there may be a possible manipulation or malfunction.



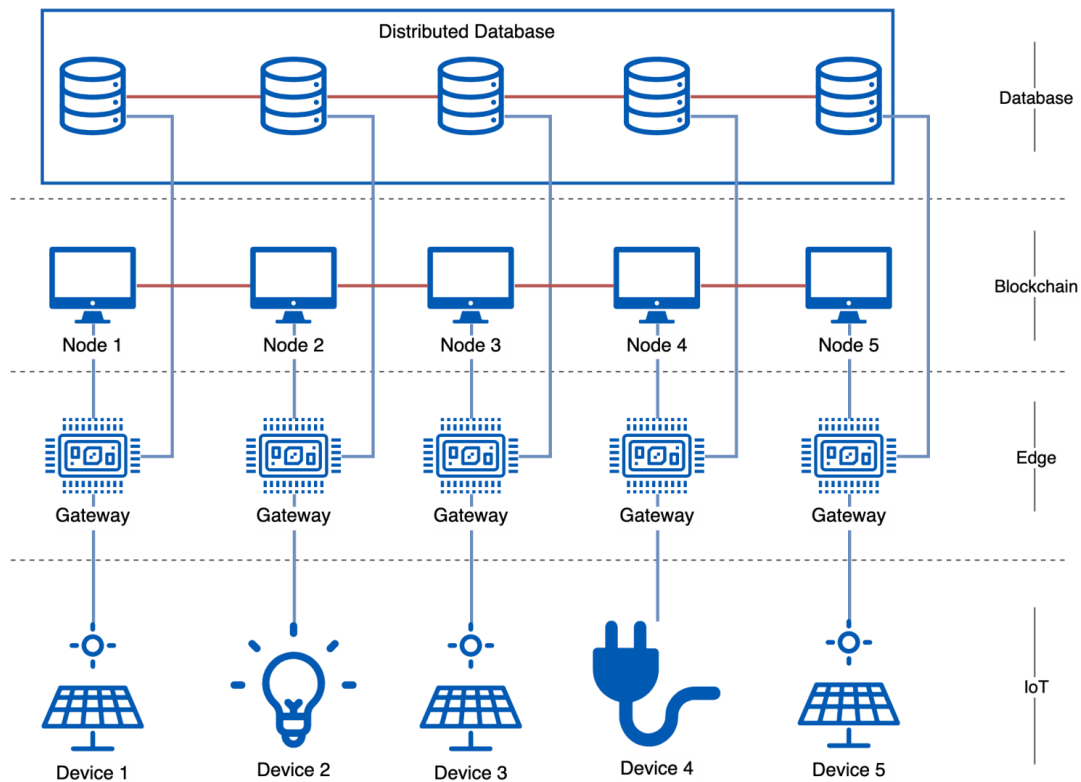


Figure 28: Blockchain platform architecture

### 3.8.3 Interaction with other Data Governance components

Within the MATRYCS-Governance the Trusted data sharing module will interact mainly with:

- the Data Storage for the extraction of harmonized data in order to calculate related fingerprint, (it should be noted that, even if in Figure 28 the data storage is indicated as a distributed database, this approach can be used for distributed, such as for a central database, as indicated in the MATRYCS general architecture)
- the Blockchain network (Ethereum) for the physical fingerprints (hash codes) storage in the chain blocks to assure data immutability.

This will be the main mechanism to assure immutability of data stored into the MATRYCS Data Storage. Optionally, the Trusted data sharing module could interact directly with the MATRYCS components, and external data providers in case of direct raw data handling; in this case the interaction will require specific data connectors to integrate the datasets with different standard and access interfaces.

## 3.9 End-to-End Security framework

At the heart of every information system are data. Data governance – the definition of involved processes in the management of access, availability, usability, (regulatory) compliance, and security of data – is recognized as necessary in this domain. The governance of data must be fundamentally defined in terms of 1) policies for creating, accessing, and using the data, 2) authentication, authorization, and auditing enforcement, and 3) management of data and policies over the complete

lifecycle. Additionally, the governance frameworks must be extended with security standards and tools. To enable secure data exchange, trust must be established with respect to identification, communication protection and usage control. An overview of the state-of-the-art on cyber security and governance models for secure data and information delivery has been provided in *D2.1: State-of-the-art analysis and Big Data Value Chain*.

The End-to-End Security framework encompasses MATRYCS platform privacy/security mechanisms with respect to anonymization, authentication, authorization, auditing, encryption, and software vulnerabilities/flaws detection and mitigation. It provides fine-grained access control and enables the implementation of advanced security policies to MATRYCS infrastructure, assets, services, end-users, and data. The End-to-End Security framework solution is based on four pillars:

1. **Identity and access management:** Entity authentication and fine-grained authorization policies based on access control mechanisms. System event logging provides access auditing and traceability. The solution to be utilized is Keycloak<sup>57</sup>.
2. **Service mesh:** As MATRYCS opts for a microservice architecture, a service mesh microservices composition for application-level security provisioning and secure service integration will be employed. Ensures service-to-service level encryption. The solution utilized will be Istio<sup>58</sup>.
3. **Privacy management:** Anonymization, compliance, and privacy policies enforcement in the system-wide deployment.
4. **Security management:** Software and system vulnerabilities/flaws detection, mitigation, and security configuration/enforcement.

The End-to-End Security framework specification and further details, focusing on the design and applied technologies of the MATRYCS-GOVERNANCE security layer, are available in *D3.2: End-to-End Security Framework*.

---

<sup>57</sup> <https://www.keycloak.org/>

<sup>58</sup> <https://istio.io/>

## 4 MATRYCS Data Model

The overall MATRYCS will be applied in 11 LSPs across 9 countries. It covers the whole lifecycle for building, which is from the product manufacturing to the operation and maintenance, as well as cross-cutting interests, such as policy making and research. This brings to one of the 5 big data challenges "Variety". After analysing the LSP data, the data category of each pilot is listed as the following:

- LSP01 BTC: Building, Device, People, Weather, Energy Consumption
- LSP02 FASADA: Building, Device
- LSP03 VEOLIA: Device
- LSP04 ASM: Building, Device, Transportation, Green Energy
- LSP05 COOPERNICO: Green Energy, Weather, Energy Performance Certificate (EPC)
- LSP06 VEOLIS: Device
- LSP07 ICLEI: Building, Green Energy, Transportation, Energy Consumption, Questionnaires
- LSP08 GDYNIA: Building, Energy Consumption
- LSP09 EREN: Building, EPC, Energy Consumption
- LSP10 LEIF: Building, Device, Energy Consumption
- LSP11 HOUSING EUROPE: Not defined

Each of the categories can be modelled by reusing the well-established data model. The FIWARE Smart Data Model with NGSI-LD standard is one of the suitable solutions. The categories, which can be covered by FIWARE, are Device, Building, Transportation, Green Energy, and Weather.

In the following is the FIWARE specification of each data model in each category:

- Green Energy: PhotovoltaicDevice<sup>59</sup>
- Green Energy: PhotovoltaicMeasurement<sup>60</sup>
- Building: Building<sup>61</sup>
- Building: BuildingOperation<sup>62</sup>
- Device: Device<sup>63</sup>
- Device: SmartMeteringObservation<sup>64</sup>
- Transportation: EVChargingStation<sup>65</sup>

<sup>59</sup> <https://github.com/smart-data-models/dataModel.GreenEnergy/blob/master/PhotovoltaicDevice/doc/spec.md>

<sup>60</sup> <https://github.com/smart-data-models/dataModel.GreenEnergy/blob/master/PhotovoltaicMeasurement/doc/spec.md>

<sup>61</sup> <https://github.com/smart-data-models/dataModel.Building/blob/master/Building/doc/spec.md>

<sup>62</sup> <https://github.com/smart-data-models/dataModel.Building/blob/master/BuildingOperation/doc/spec.md>

<sup>63</sup> <https://github.com/smart-data-models/dataModel.Device/blob/master/Device/doc/spec.md>

<sup>64</sup> <https://github.com/smart-data-models/dataModel.Device/blob/master/SmartMeteringObservation/doc/spec.md>

<sup>65</sup> <https://github.com/smart-data-models/dataModel.Transportation/blob/master/EVChargingStation/doc/spec.md>

○ Weather: WeatherObserved<sup>66</sup>

After reusing the FIWARE Smart Data Model, there are still categories Energy Consumption, EPC, and People, which need to be modelled.

## 4.1 Vocabularies and Ontologies

### 4.1.1 Brick schema

Nowadays, buildings are increasingly becoming incubators of data and information. The integration of intelligence systems, sensor and networking (i.e. Internet of Things - IoT) in buildings is always more and more common. Even though the amount of data generated by buildings is growing exponentially, there is still no clear industry-wide standard for using, sharing and exchanging information in a unified way.

Indeed, some of the day-to-day actions applied in construction, such as energy audits, optimization of controls or detection of faults in building systems are often slowed down by the lack of standardization of metadata, making processes much more time-consuming and burdensome (from a labor point of view) and not reusable in other applications. This problem is generally related to the lack of semantic interoperability.

The latter is defined by Pritoni et al.<sup>67</sup> as “the capability of two or more networks, systems, devices, applications, or components to work together, and to exchange and readily use information securely, effectively, and with little or no inconvenience to the user”. On the technical side, the interoperability between devices is achieved using the same communication protocol, on the contrary the semantic layer is not defined or unambiguously defined, not allowing the development of applications that can be used in different buildings.

All this shows that it is extremely important to define a univocal semantic layer, which, based on a standard model, allows interoperability between different services and platforms.

The semantic model is a metadata schema that describes precisely and unambiguously the different elements that characterize the building and its systems. The peculiarity is that it identifies different entities by means of a glossary or dictionary and links them to each other using relationships. Ontologies establish the domain's concepts and relationships, classes, and attributes.

As written in [67]: “*The World Wide Web Consortium (W3C) established standards that created the Semantic Web, an extension of the World Wide Web aimed to make internet data machine- readable. Ontologies that comply with W3C standards use triples in the form of subject–predicate–object to encode knowledge, following the Resource Description Framework (RDF) data model. When multiple triples are put together, they form a directed multigraph. The W3C also provides a set of fundamental languages that can be leveraged to define ontologies using classes and properties (i.e., Resource Description Framework Schema or RDFS), description logics (i.e., Web Ontology Language or OWL) and constraints (i.e., OWL and Shapes Constraint Language or SHACL). Ontologies and Semantic Web technologies have*

<sup>66</sup> <https://github.com/smart-data-models/dataModel.Weather/blob/master/WeatherObserved/doc/spec.md>

<sup>67</sup> Metadata Schemas and Ontologies for Building Energy Applications: A Critical Review and Use Case Analysis – energies- April 2021 DOI: <https://www.mdpi.com/1996-1073/14/7/2024>

*experienced some adoption for internet services, providing interoperability of digitized data, for example, between search engines, web crawlers, and other web-based software"*

On the building domain the ontologies developed and under development are summarized in the following schema (see Table 13).

**Table 13: Metadata schema as resulting of review process of existing applications**

Phase of the Building Life Cycle	Group	Schemas (Year Created)
Design and energy modelling	-	<ul style="list-style-type: none"> <li>• Industry Foundation Classes (IFC)</li> <li>• Green Building XML (gbXML)</li> <li>• ifcOWL</li> <li>• Tubes</li> <li>• SimModel Ontology</li> <li>• Energy-ADE</li> </ul>
Operations	Sensor network, Internet of things (IoT) and smart homes	<ul style="list-style-type: none"> <li>• Semantic Sensor Network/Sensor, Observation, Sample and Actuator (SSN/SOSA)</li> <li>• Web Thing Model</li> <li>• OneM2M<sup>68</sup> BaseOntology's</li> <li>• One Data Model (oneDM)</li> <li>• Smart Energy Aware Systems</li> <li>• ThinkHome</li> <li>• Building Ontology for Ambient Intelligence</li> <li>• DogOnt</li> <li>• Ontology of Smart Building</li> <li>• Smart Application REference (SAREF)</li> </ul>
Operations	Commercial building, automation and monitoring	<ul style="list-style-type: none"> <li>• Project Haystack</li> <li>• BASont</li> <li>• Haystack Tagging Ontology (HTO)</li> <li>• Brick Schema</li> <li>• Google Digital Building Ontology</li> <li>• Semantic BMS Ontology</li> <li>• CTRLont</li> <li>• Green Button</li> <li>• RealEstateCore (REC)</li> <li>• Building Topology Ontology (BOT)</li> <li>• Building Automation and Control Systems (BACs)</li> <li>• Knowledge Model for City (KM4City)</li> <li>• EM-KPI Ontology</li> </ul>

<sup>68</sup> <https://en.wikipedia.org/wiki/OneM2M>

Phase of the Building Life Cycle	Group	Schemas (Year Created)
Operations	Grid-interactive efficient building (GEB) applications	<ul style="list-style-type: none"> <li>Facility Smart Grid Information Model</li> <li>RESPOND</li> </ul>
Operations	Occupants and behaviour	<ul style="list-style-type: none"> <li>DNAs Framework (obXML)</li> <li>Occupancy Profile (OP) Ontology</li> <li>Onto-SB: Human Profile Ontology for Energy Efficiency in Smart Building</li> <li>OnCom</li> </ul>
Operations	Asset management and audits	<ul style="list-style-type: none"> <li>Building Energy Data Exchange Specifications (BEDES)</li> <li>Virtual Building Information Systems (VBIS)</li> <li>Ontology of Property Management (OPM)</li> </ul>

The core concept of an ontology applied on a building domain is defined in the following table.

**Table 14: Main core concepts of building ontology**

Category	Concept	Proprieties	Relationship to/from
Zones and Spaces	Space	Function Floor Area	Composed of spaces Adjacent spaces
	Zone	Floor area	Overlaps one or more spaces Overlaps other zones
	Building floor	Orientation	Composed of spaces
Envelope	Envelope element	Type of envelope element(wall, roof, floor, window) Envelop characteristics (e.g., thermal resistance, storage, solar seat gain coefficient)	Part of space
Building System and Equipment	System	Type of system	Composed of components
	Equipment	Type of equipment Rated power draw Rated efficiency Remaining lifespan	Serves zone Located in space Metered by meter Connected to equipment
	HVAC equipment	Rated capacity	
	Lighting equipment	Rated(max.) luminous	Serves zone/space

Category	Concept	Proprieties	Relationship to/from
		flux Minimum relative light output Rated (max.) power Correlated color temperature Spaectral power distribution Rated Input voltage Rated (max.) input current	Located in space Metered by (internal/external) Meter Connected to electrical Junction box or other equipment
	Other end use	Type of end-use	
	Component	Type of component	Part of system Located in space Connected to component
Control Devices	Control device		Has points
	Control point	Input/Output Type Physical/Virtual type Type of virtual point (setpoints, command, alarm) Unit of measure Control interval	Linked to sensor/actuator Linked to time series data
	Control strategy	Schedule Event	Has inputs Has outputs Linked to sensor Linked to actuator Linked to time series data
Sensor/Actuator	Sensor	Type of sensor Unit of measure Measurement Interval Reporting Interval	Senses/Measures point Senses/Measures equipment Aggregates measurements
	Actuator	Unit of measure Actuation interval	Actuates point Actuates equipment Integrates/Prioritizes actuations

In this 1<sup>st</sup> *technology release*, preliminary test with brickschema ontology was carried out with the aim to generate the data model for the building case study of LSP2 (FASADA).

Brick is a metadata scheme that takes from the Haystack project the use of tags to preserve the flexibility and ease of use of annotating metadata. Brick unlike Haystack schema places restrictions to prohibit arbitrary tag combinations and relationships. For example, the unit for temperature to be chosen can be only Fahrenheit or Celsius or give an error if sensor and set point occurring together in a tags combination for a data point.

Brick introduces the concept of tagset that group together relevant tags to represent an entity. They are:

- **Points** are physical or virtual entities that generate time-series data. Physical points include actual sensors and setpoints in a building, whereas virtual points encompass synthetic data streams that are the result of some process which may operate on other timeseries data, e.g. average floor temperature sensor.
- **Equipment:** Physical devices designed for specific tasks controlled by points belonging to it. E.g., light, fan, Air Handling Unit (AHU).
- **Location:** Areas in buildings with various granularities. E.g. room, floor.
- **Resource:** Physical resource or materials that are controlled by equipment and measured by points. An AHU controls resources such as water and air, to provide conditioned air to its terminal units.

Together with these entities, Brick defines a minimal set of relationships that capture the connection between them. A Brick building model can be visualize using the Resource Description Framework (RDF) which represents graph-based knowledge as tuples of (subject, predicate, object) termed triples.

Unlike the other languages for the building metadata scheme, Brick is distinguished for:

- **Completeness:** The current version of Brick covers the 98% of the vocabularies found in six buildings in different countries.
- **Vocabulary Extensibility:** The structure of Tags/TagSets allow easy extensions of TagSets for newly discovered domains and devices while allowing inferences of the unknown TagSets with Tags.
- **Usability:** Brick represents an entity as a whole instead of annotating it. It promotes consistent usages by different actors. Furthermore, its hierarchical TagSets structure allows user queries more generally applicable across different systems.
- **Expressiveness:** Brick standardizes canonical and usable relation-ships, which can be easily extended with further specifications.
- **Schema Interoperability:** Using RDF enables straightforward integration of Brick with other ontologies targeting different domains or aspects.



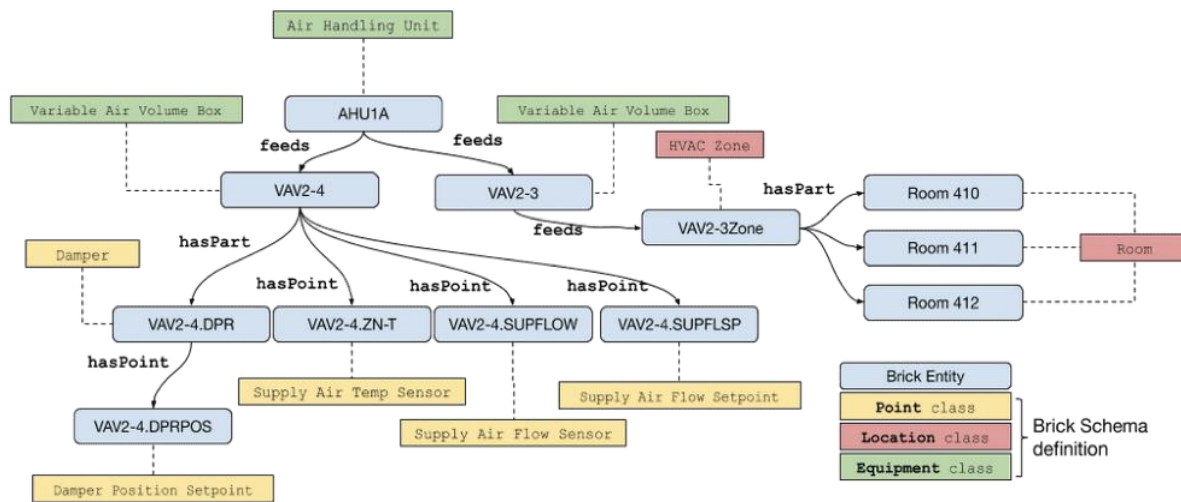


Figure 29: Example of Brick Schema

The relationship support by Brick and updated by Brick+ are shown in the following table:

Table 15: Relationship and definition for brick and brick plus schema

Relationship	Definition
hasLocation	Subject is physically located in the object entity
feeds	Subject conveys some media to the object entity in the context of some sequential process
hasPoint	Subject has a monitoring, sensing or control point given by the object entity
hasPart	Subject is composed – logically or physically – in part by the object entity
Measures	Subject measures a quantity or substance given by the object entity
Regulates	Subject informs or performs the regulation of the substance given by the object entity
hasOutputSubstance	Subject produces or exports the object entity as a product of its internal process
hasInputSubstance	Subject receives the object entity to conduct its internal process

More information related to these projects are available at the following link <https://brickschema.org/#home>

Regarding the application in MATRYCS, one building has been modelled using the brick schema. The building is a kindergarten located in the city of Gdynia. Using a **BIM** file, an **IFC** file was generated with which it was possible to generate an **RDF** (Resource Description Framework) file of type **turtle (.ttl)** containing the most important geometric information of the building.

Only two comfort sensors, monitoring the thermal comfort and Indoor air quality in two rooms of the

building, were installed. These sensors have been modelled separately, as this information was not present in the IFC file.

A python script was generated (using a starting work of G.Fierro<sup>69</sup>), to allow the automatic translation of some elements from IFC files to brickschema. This work is still under development aiming to improve the existing library including much more information coming from the .ifc file.

The brickschema model of the kindergarten together with the python files, used to generate the model are available in the Github MATRYCS repository<sup>70</sup>.

The file will then be integrated into the MATRYCS infrastructure through the use of the Neo4j graphical database (using the neosemantics plugin).

## 4.1.2 SAREF

The Smart Applications REFERENCE (SAREF)<sup>71</sup> is one of the well-established Ontologies, which is intended to cover the various actors in the Internet of Things (IoT).

Figure 30 shows an overview of the SAREF. The main classes are contained in the box. The connection between each class is the relationship<sup>72</sup>.

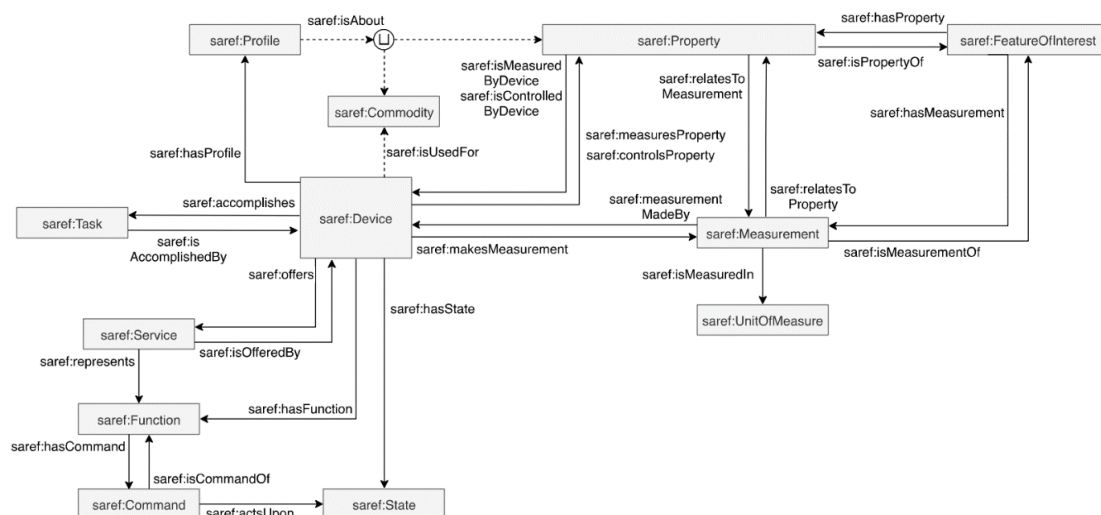


Figure 30: Overview of the SAREF ontology

SAREF4BLDG is an extension of SAREF, which is created based on the Industry Foundation Classes standard for building information<sup>73</sup>. The goal of SAREF4BLDG is intended to improve the interoperability among different phases of the building life cycle. The overview of SAREF4BLDG is depicted in Figure 31. As can be observed, saref:device is reused from SAREF. The class geo:SpatialThing is from the geo ontology, which proposed the conceptualization for location. This extension can be reused in MATRYCS to build an ontology to cover the whole life cycle of building.

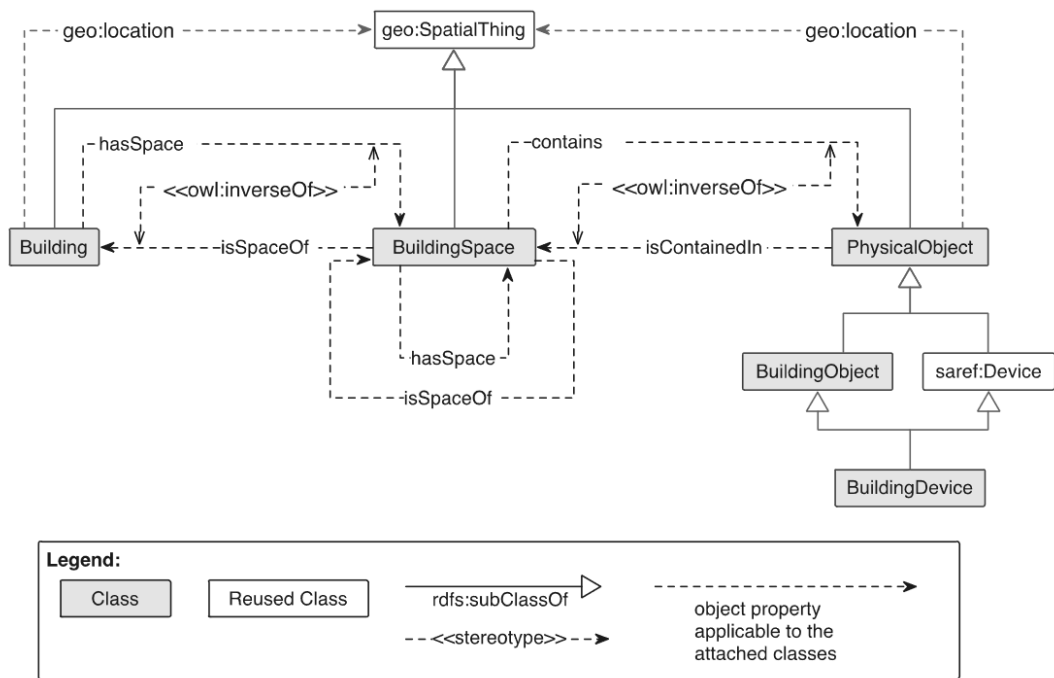
<sup>69</sup> <https://github.com/gtfierro/brick-ifc-convert>

<sup>70</sup> <https://github.com/MATRYCS>

<sup>71</sup> <https://saref.etsi.org>

<sup>72</sup> [https://www.etsi.org/deliver/etsi\\_ts/103200\\_103299/103264/03.01.01\\_60/ts\\_103264v030101p.pdf](https://www.etsi.org/deliver/etsi_ts/103200_103299/103264/03.01.01_60/ts_103264v030101p.pdf)

<sup>73</sup> [https://www.etsi.org/deliver/etsi\\_ts/103400\\_103499/10341003/01.01.02\\_60/ts\\_10341003v010102p.pdf](https://www.etsi.org/deliver/etsi_ts/103400_103499/10341003/01.01.02_60/ts_10341003v010102p.pdf)



**Figure 31: General overview of the top levels of the SAREF4BLDG**

## 5 MATRYCS–GOVERNANCE Integration at M11

### 5.1 Case study LSP1 and LSP5

In this section a general description of the **LSP1** (BTC: BUILDING OPERATION-Facility and resources fingerprinting for efficiency and optimal balancing of energy vectors) and **LSP5** (COOPERNICO: ENERGY COMMUNITIES-services for the better management of self-production systems) case studies can be found. For more information, please check “D2.1 - State-of-the-art analysis and Big Data Value Chain”.

#### **LSP1: BUILDING OPERATION: Facility and Resources Fingerprinting for Efficiency and Optimal Balancing of Energy Vectors**

LSP1 focuses on the building level and is classified under the MATRYCS-PERFORMANCE pilot category. It is led by BTC (Slovenia) and contemplates three facilities.

The BTC d.d. is one of the leading commercial property development companies in the Central and Southeastern Europe. The company is managing a range of business, commercial and recreational, entertainment and cultural activities with wide range of logistics services, covering an area of approx. 475.000 m<sup>2</sup>. BTC also runs a logistic service business unit, which holds the leading market position in FMCG logistics in Slovenia. It also provides first-class property management services for the largest Slovenian clients in the commercial real estate service sector.

LSP1 includes three BTC facilities: the Business tower BTC City, the Atlantis water park and the Logistics center. All facilities have different control systems.

The **Business tower BTC City** is entirely occupied by tenants of business premises. It has two different BMS controlling separately old and new equipment for temperature control. Besides there is an EMS collecting data from electricity meters and form calorimeter in the heating sub-station. The access control system is also installed in the building and it is used to control and monitor access to the premises and record working time of employee. However there is no links between BMS, EMS and access control data.

The **Atlantis Water Park** consists of indoor and outdoor pools, various saunas and water attractions. It has also a separate BMS, EMS and access control system for visitors. Also here there are no links between BMS, EMS and access control data.

The **Logistics center** is the third element of the BTC d.d. that is included in the MATRYCS project, and consists of warehouses and cold storages. Unfortunately, only cold storages are equipped with BMS, for operation of refrigeration units and for control of temperature in the refrigeration chambers. The area is partially covered by EM that monitors electricity consumption in warehouses and cold storages. Also, a part of warehouse heating and water consumption is included in the EMS. Access control system is responsible for monitoring and control of the access in the warehouses. There is also a warehouse management system (WMS), responsible for the planning and management of the warehouse occupancy. There are no links between BMS, EMS, WMS and access control data.

The main objectives of the large – scale pilot are the following:



- Improving the operation and maintenance of the pilots taking into account reliable analysis.
- Upgrade the system of existing and development of new energy performance indicators (KPI)
- Expose potentials for flexibility and energy efficiency to various energy service providers or utilities.

Within this first release, UC01\_01- Action plans for preventive maintenance will be initiated, in order to complete a proof of concept. In this use case, the user will be able to select actions plans and closely monitor the implementation to increase reliability and efficiency of the systems. Two services are involved here: s1.2 (BAC services) and s1.4 (TBMs).

In the case of service s1.4, TBM1 will be deployed (Identification of frozen sensors). This first implementation will be done based on the data provided by BTC pilot about the air conditioning systems within Atlantis Water Park, which is located in Slovenia. The data provided contains information on 43 temperature sensors, coming from 14 air conditioning systems (these temperature variables refer to blow in temperature, exhaust air temperature, and fresh air temperature, mainly). Small differences can be found in variables depending on the specific air conditioning system.

#### **LSP5: ENERGY COMMUNITIES: Services for the better management of self-production systems**

LSP 5 focuses on the building and district level and is classified under the MATRYCS-PERFORMANCE pilot category. It is led by COOPERNICO (Portugal) and will involve a vast number of consumers.

In particular, LSP5 will involve 850 citizens, of which at least half will be Coopérnico's members and a third prosumers. This pilot will aim to collect all available information (such as EPCs, consumption and production profiles, weather data etc.) to provide tools that can help small consumers, citizens and SMEs, improve their energy efficiency standards and better manage their energy assets.

This LSP proposes the use of data coming from smart meters of the members collecting their energy production and matching it to their real electricity consumption. This information will be enriched with other data, for example: the type of building, year of construction and location. This data will be the basis to design a service where advice is given on how to better use the energy produced, consumed and what improvements can be done to the building itself to improve their energy efficiency. Besides, this service could be used by other members who only consume electricity from the grid, offering useful information and giving advises on how they can improve their building or energy use.

Besides, LSP5 will explore solutions to foster collaborations between local energy consumers to support the creation of new Renewable Energy Communities (RECs) and collective self-consumption projects in alignment with the on-going transposition of the REDII directives.

Therefore, the main objectives of the pilot are:

- Help citizens identify ways to save energy by analysing their consumption pattern
- Increase "efficiency" of currently deployed energy assets (e.g. PV systems and EVs) by optimizing local production and consumption
- Increase citizen-owned RES capacity installed
- Map local electricity consumption to identify citizens in energy poverty condition

Within this first release, work will begin on UC05\_02- Identification of PV performance issues, in order to complete a proof of concept. In this use case, the user will be able to identify when his PV system is not performing properly. Two services are involved here: s1.1 (Energy prediction) and s1.4 (TBMs).

Information coming from Coopernico Solar's projects has been explored, and data from Adega Palmela Cooperative has been analysed and used to work on this first release. Two kinds of datasets have been found: one type only provides information about hourly production and CO2 emissions of the PV; the second group provides information closer to the involved equipment, the number of inverters and registered values for representative information (voltage, intensity, power, temperature and energy produced).

### 5.1.1 Data Acquisition

Preliminary work was carried out together with BTC (LSP1) and Coopernico (LSP5) pilots to define both the data to be integrated and the communication interfaces. For privacy and security reasons, both BTC and Coopernico pilots have opted to share their datasets via the SFTP protocol. In this regard, two folders have been created for both LSP1 and LSP5, on the SFTP Server configured in the Interoperability Service Module. The access to their own staging area has been guaranteed through user credentials provided to the pilots. Dedicated data connectors have been created within the Interoperability Service Module in order to integrate the datasets shared in the SFTP Server to be processed by the Data Pre-Processing & Semantic Enrichment Layer (see *Figure 32*).

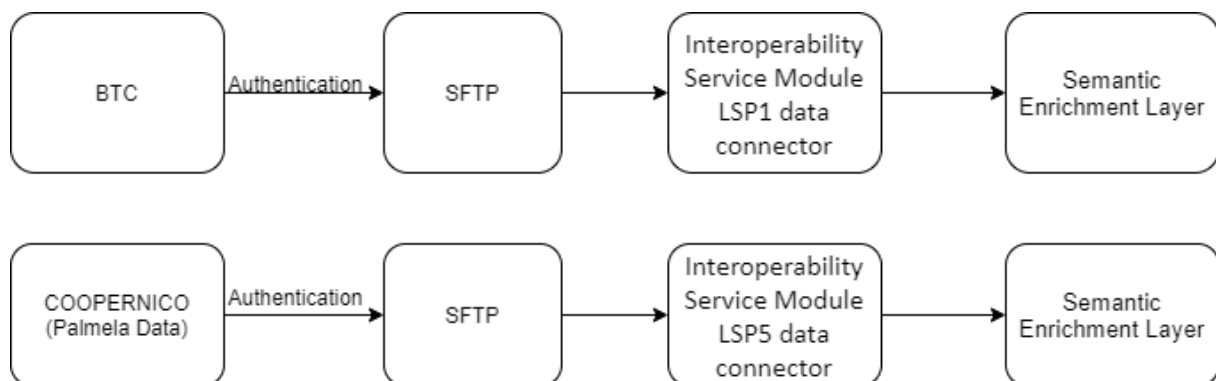


Figure 32: LSP1 and LSP5 data acquisition

#### 5.1.1.1 LSP1

For the LSP1 case study, a data connector microservice (NiFi macro process group) has been configured within the Interoperability Service Module to integrate the following datasets provided by BTC pilot:

- Cold Storage – Electricity
- BTC Tower – Electricity
- BTC Tower – Heating
- Cold Storage – Heating
- Cold Storage - Solar Power Generation
- Water Park – Electricity

As showed in the *Figure 33*, LSP1 data are sent by BTC in the SFTP server and then a set of NiFi processors have been implemented in order to integrate LSP1 data from the SFTP staging area, to process it (data cleansing, data curation, data modelling) with a python script defined in Data Pre-processing & Semantic Enrichment layer (*section 5.1.2*) and to publish it on the Kafka instance (Streaming module).

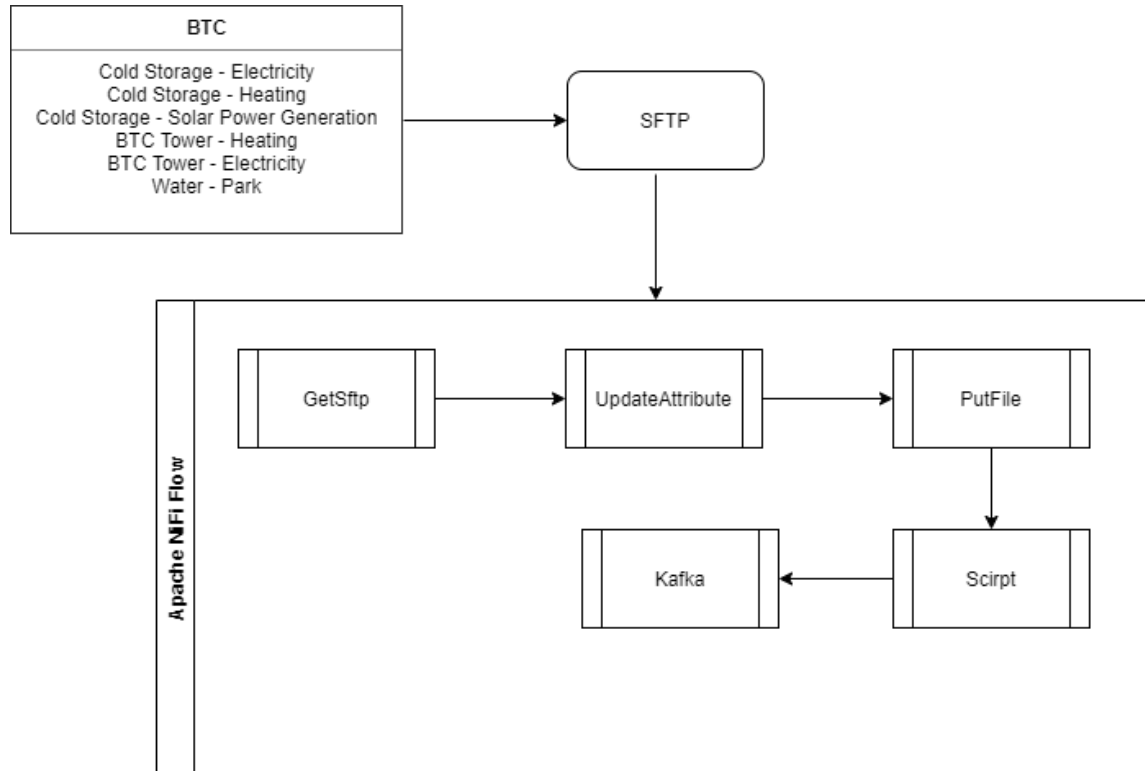


Figure 33: LSP1 data connector

### 5.1.1.2 LSP5

For the LSP5 case study, a data connector microservice (NiFi macro process group) has been configured within the Interoperability Service Module to integrate the following datasets provided by Coopernico pilot:

- Coopérnico solar projects production (Palmela data).

As showed in the *Figure 34*, LSP5 data are sent by Coopernico in the SFTP server and then a set of NIFI processors have been implemented in order to integrate LSP5 data from the SFTP staging area, to process it (data cleansing, data curation, data modelling) with a python script defined in Data Pre-processing & Semantic Enrichment layer (*section 5.1.2*) and to publish it on the Kafka instance (Streaming module).

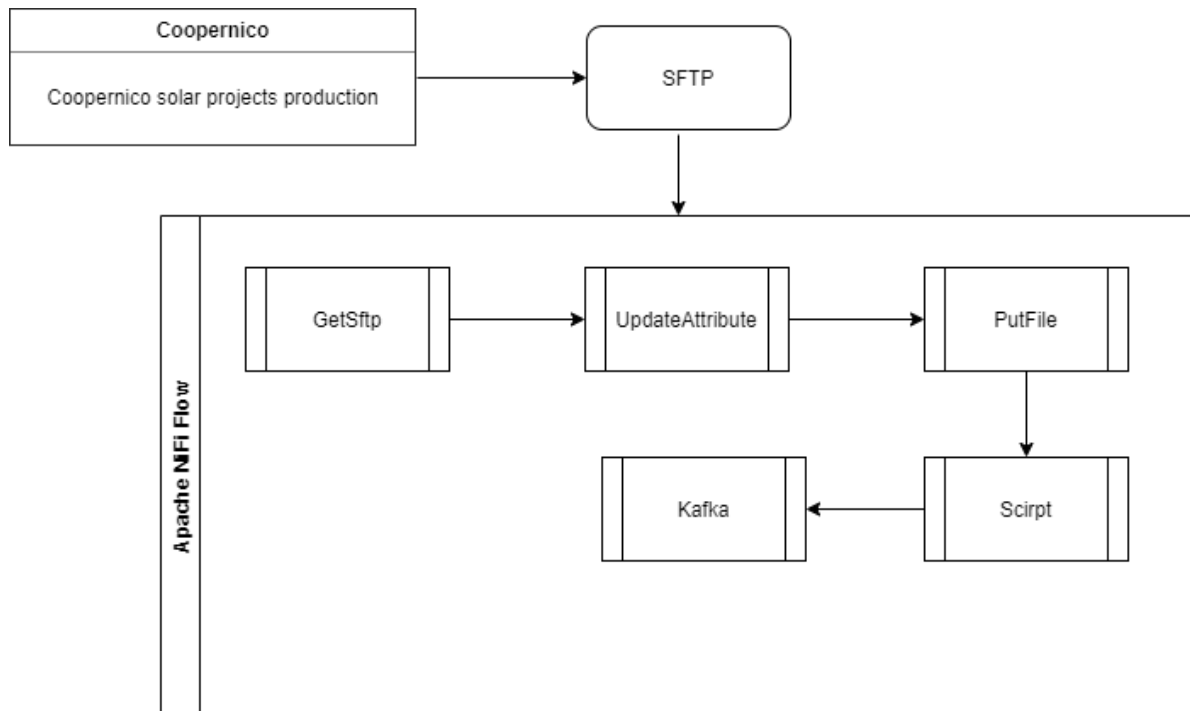


Figure 34: LSP5 data connector

## 5.1.2 Data processing and modelling

The LSP1 (BTC) data contains information about different meters in different buildings. Therefore, the common data model needs to cover the information about the meter itself, the time series data and building related data. After reusing the FIWARE Smart Data Model, the building related data is modified into two entities: Building and Building Operation.

The LSP5 (Coopernico) data contains only information about the photovoltaic device. The Photovoltaic Device entity is based on FIWARE PhotovoltaicDevice<sup>74</sup> data model. The PhotovoltaicDevice is still under development, which does not have an official specification.

The FIWARE Building entity is a human build structure where different activities occur. The properties<sup>75</sup> are described as following:

- address: The mailing address
- alternateName: An alternative name for this item
- areaServed: The geographic area where a service or offered item is provided

<sup>74</sup> [https://github.com/smart-data-models/dataModel.Energy/blob/master/PhotovoltaicDevice\\_incubated](https://github.com/smart-data-models/dataModel.Energy/blob/master/PhotovoltaicDevice_incubated)

<sup>75</sup> <https://github.com/smart-data-models/dataModel.Building/blob/master/Building/doc/spec.md>



- category: Category of the building. Enum:'apartments, bakehouse, barn, bridge, bungalow, bunker, cathedral, cabin, carport, chapel, church, civic, commercial, conservatory, construction, cowshed, detached, digester, dormitory, farm, farm\_auxiliary, garage, garages, garbage\_shed, grandstand, greenhouse, hangar, hospital, hotel, house, houseboat, hut, industrial, kindergarten, kiosk, mosque, office, parking, pavilion, public, residential, retail, riding\_hall, roof, ruins, school, service, shed, shrine, stable, stadium, static\_caravan, sty, synagogue, temple, terrace, train\_station, transformer\_tower, transportation, university, warehouse, water\_tower'
- collapseRisk: Probability of total collapse of the building.
- containedInPlace: The place which contained the building
- dataProvider: A sequence of characters identifying the provider of the harmonised data entity.
- dateCreated: Entity creation timestamp. This will usually be allocated by the storage platform.
- dateModified: Timestamp of the last modification of the entity. This will usually be allocated by the storage platform.
- description: A description of this item
- floorsAboveGround: Floors above the ground level
- floorsBelowGround: Floors below the ground level
- id: Unique identifier of the entity
- location: Geojson reference to the item. It can be Point, LineString, Polygon, MultiPoint, MultiLineString or MultiPolygon
- name: The name of this item.
- occupier: Person or entity using the building
- openingHours: Opening hours of this building.
- owner: A List containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s)
- peopleCapacity: Allowed people present at the building
- peopleOccupancy: People present at the building
- refMap: Reference to the map containing the building
- seeAlso: list of uri pointing to additional resources about the item
- source: A sequence of characters giving the original source of the entity data as a URL. Recommended to be the fully qualified domain name of the source provider, or the URL to the source object.
- type: NGSI Entity type

The FIWARE Building Operation<sup>76</sup> entity describes a generic operation applied to the referenced building. The building operation contains dynamic data reported by or associated with a building or operations to the building.

The data model has the following properties:

- alternateName: An alternative name for this item
- dataProvider: A sequence of characters identifying the provider of the harmonised data entity.
- dateCreated: Entity creation timestamp. This will usually be allocated by the storage platform.
- dateFinished: The actual end date for the operation.
- dateModified: Timestamp of the last modification of the entity. This will usually be allocated by the storage platform.
- dateStarted: The actual start date for the operation.
- description: A description of this item
- endDate: The planned end date for the operation.
- id: Unique identifier of the entity
- name: The name of this item.
- operationSequence: Id of the sequence of the operation when available
- operationType: Type of the operation on the building
- owner: A List containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s)
- refBuilding: Building reference where the operation is performed.
- refOperator: Reference to the Operator doing the operation on the building.
- refRelatedBuildingOperation: Reference to other building operations when in sequence
- refRelatedDeviceOperation: Devices related to the current operation. A list of references to an entity of type Device.
- result: Result of the building operation. Enum:'ok, aborted'
- seeAlso: list of uri pointing to additional resources about the item
- source: A sequence of characters giving the original source of the entity data as a URL. Recommended to be the fully qualified domain name of the source provider, or the URL to the source object.
- startDate: The planned start date for the operation.
- status: Status of the operation. Enum:'cancelled, finished, ongoing, planned, scheduled'
- type: It has to be BuildingOperation

<sup>76</sup> <https://github.com/smart-data-models/dataModel.Building/blob/master/BuildingOperation/doc/spec.md>

Those properties provide a harmonized description of a building. Depends on different use cases, the properties are variant. The mandatory properties are id, type. The FIWARE smart data model has been developed in relation to the FIWARE NGSI standard protocol. Therefore, the part of design principles, which focus on the NGSI protocol, will be included in MATRYCS data model.

The data model of LSP1 is described in Figure 35. The data model of LSP5 is shown in *Figure 36*. The development approach is discussed in *section 3.3.2.2*, which is implemented as below:

- Determine the domain and scope of the data model  
In this case study, we only focus on the LSP1 and LSP5 data. As mentioned before, the data is mainly related to the building and photovoltaic domain.
- Consider reusing existing schemas  
The building and photovoltaic domain specific data model in FIWARE Smart Data Model are reused.
- Define the common entity and entity hierarchy, which should cover all the different LSP data  
Based on the Smart Data Model, LSP1 dataset is defined in four entities: Building, Building Operation, Device, and Time Series Data. The Building Operation entity contains the reference to the Building entity and the related Device entity. The Time Series Data is related to a single Device entity.  
LSP5 dataset is covered by Photovoltaic Device entity, which is a sub-class of Device.
- Define the common property in the defined class  
Right now, the common property is still under development. The mandatory properties are integrated. In Building entity are the id, type, category and address. In Building Operation are the id, type and refBuilding. In other entities are the id relevant.
- Map each pilot data to the Common Data Model  
The last step is to map the LSP1 and LSP5 data to the defined Data Model, which is developed by applying a Python dictionary. The python dictionary maps the LSP dataset to the Common Data Model through key value pairs. The dictionary is executed by the Python script and integrated in Apache NiFi.

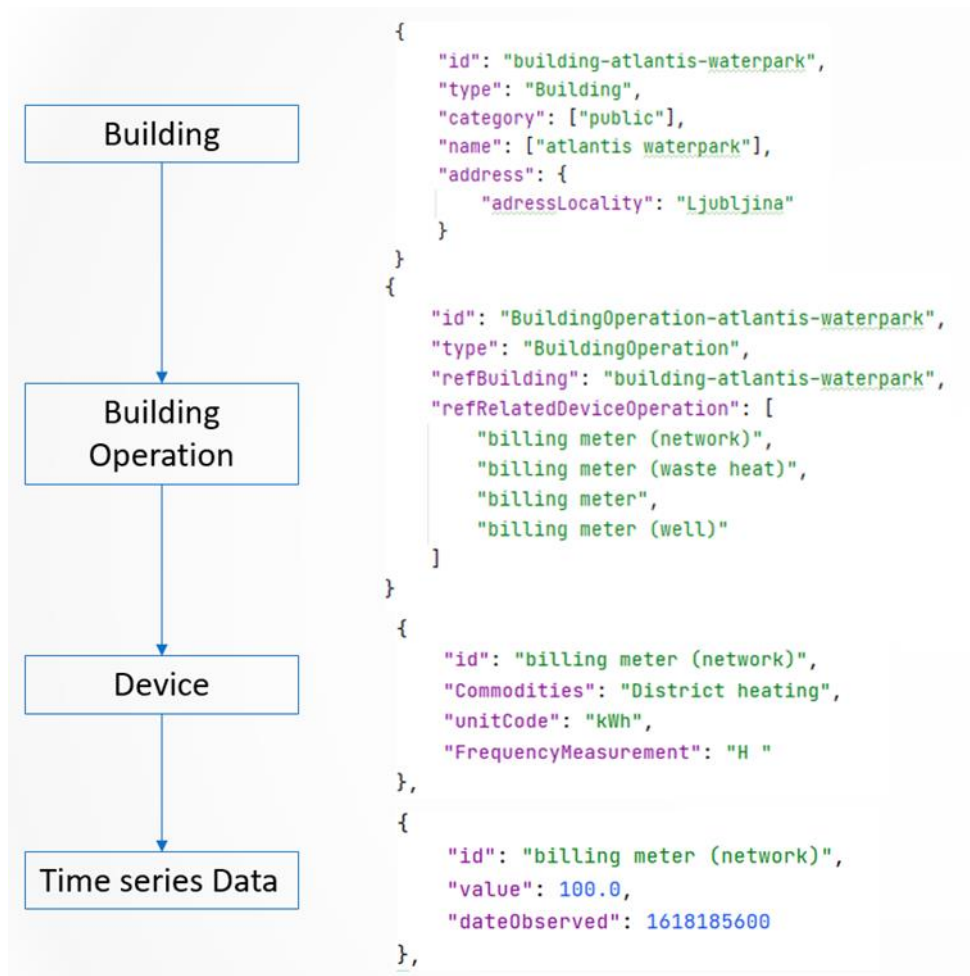


Figure 35: LSP01 BTC data model



Figure 36: LSP05 Coopernico data model

Additionally, there are different timestamps used in LSP1 and LSP5 datasets. Therefore, a unique time

format is necessary for Common Data Model. The Epoch time<sup>77</sup> is a time format for describing a point in time. It is the number of seconds that have elapsed since the Unix epoch, which is 00:00:00 UTC on 1 January 1970. Applying the Epoch time provides the advantage of easily data storage and data manipulate than the conventional data format. It can be easily transformed by using the standard language (e.g. Python, C, Java, etc.). The Epoch time can be found at the bottom of *Figure 35 "dataObserved"* field and *Figure 36 "Timestamp"* field.

As mention in *section 3.3*, the raw data will be processed with python scripts deployed in Apache NiFi instance (Interoperability service module). The DATASET\_ORDER\_DICT in *Table 16* is used to provide the information about the path of the raw data and the mapping to the dataset\_columns\_dict. The dataset\_columns\_dict describe the mapping of the original column name to the Matrycs data model. Currently each LSP has a python script, which take response for all task about pre-processing and data harmonization.

**Table 16: DATASET\_ORDER\_DICT in python script**

```
DATASET_ORDER_DICT
[
  {
    'msg':'BTC_MEASUREMENT',
    'path':'DATA/BTC_data_20210412.csv',
    'dataset_columns_dict':BTC_ENERGY_CONSUMPTION,
    'separator':','
  },
  {
    'msg':'BTC_DEVICE',
    'path':'DATA/BTC_data_20210412.csv',
    'dataset_columns_dict':BTC_DEVICE,
    'separator':','
  },
  {
    'msg':'BTC_PEOPLE_MEASURED',
    'path':'DATA/BTC_data_20210412.csv',
    'dataset_columns_dict':BTC_PEOPLE_MEASURED,
    'separator':','
  }
]
```

As Output for this script, the harmonized data model is published in JSON format to the Kafka instance of the Streaming module (*section 3.4*). Data storage and Reasoning engine will get the data model through Kafka topic.

<sup>77</sup> [https://en.wikipedia.org/wiki/Unix\\_time](https://en.wikipedia.org/wiki/Unix_time)

## 5.1.3 Data Storage and Reasoning Engine

### 5.1.3.1 Data Storage

Data Storage integrates into the MATRYCS-GOVERNANCE layer using a dedicated microservice. The microservice, written in Golang, implements an input Kafka consumer connection towards the Streaming module for retrieving JSON formatted LSP data in a harmonized data model (see *section 5.1.2*) on specific Kafka topics. The implementation of Data Storage Golang Kafka consumer connection using *kafka-go*<sup>78</sup> library can be seen in *Table 17*. The retrieved data are stored on a Linux filesystem (i.e., temporary storage area) using newline delimited JSON (NDJSON) format for usage by the Reasoning Engine and later historical raw data access. Additionally, Data Storage provides a real-time big data distributed database ScyllaDB for storing and retrieving custom data produced by other MATRYCS-GOVERNANCE components. ScyllaDB will be installed in a cloud environment and accessible via ScyllaCQL Drivers<sup>79</sup>. At the time of writing this deliverable, additional experimentation with MongoDB is being performed targeting higher performance optimization.

**Table 17: Data Storage Golang Kafka consumer**<sup>78</sup>

```
r := kafka.NewReader(kafka.ReaderConfig{
    Brokers: []string{"localhost:9092"},
    Topic:   "stream1-MATRYCS",
    Partition: 0,
})

for {
    m, err := r.ReadMessage(context.Background())
    if err != nil {
        break
    }
    fmt.Printf("message at offset %d: %s = %s\n", m.Offset, string(m.Key), string(m.Value))
}

if err := r.Close(); err != nil {
    log.Fatal("failed to close reader:", err)
}
```

<sup>78</sup> <https://github.com/segmentio/kafka-go>

<sup>79</sup> <https://docs.scylladb.com/using-scylla/drivers/cql-drivers/>

#### 5.1.3.1.1 [Case LSP1](#)

Messages produced by the Streaming module and received via Data Storage's Kafka consumer for LSP1 are written on a Linux filesystem using NDJSON format. A sample of a file for LSP1 according to a harmonized data model (see [section 5.1.2](#)) can be seen in [Table 18](#).

**Table 18: Staging area file for LSP1**

```
{ "id": "billing meter (network)", "value": 99.8, "dateObserved": 1618185600 }
{ "id": "billing meter (network)", "value": 100.2, "dateObserved": 1618189200 }
{ "id": "billing meter (network)", "value": 102.0, "dateObserved": 1618192800 }
{ "id": "billing meter (network)", "value": 102.0, "dateObserved": 1618196400 }
{ "id": "billing meter (network)", "value": 102.1, "dateObserved": 1618200000 }
...
```

#### 5.1.3.1.2 [Case LSP5](#)

Messages produced by the Streaming module and received via Data Storage's Kafka consumer for LSP5 are written on a Linux filesystem using NDJSON format. A sample of a file for LSP5 according to a harmonized data model (see [section 5.1.2](#)) can be seen in [Table 19](#).

**Table 19: Staging area file for LSP5**

```
{ "id": "05photovoltaicDevice", "Timestamp": 1580605200, "production": "0", "specificRate": "0", "co2Reduced": " " }
{ "id": "05photovoltaicDevice", "Timestamp": 1580608800, "production": "0", "specificRate": "0", "co2Reduced": " " }
{ "id": "05photovoltaicDevice", "Timestamp": 1580612400, "production": "0", "specificRate": "0", "co2Reduced": " " }
{ "id": "05photovoltaicDevice", "Timestamp": 1580616000, "production": "0", "specificRate": "0", "co2Reduced": " " }
{ "id": "05photovoltaicDevice", "Timestamp": 1580619600, "production": "0", "specificRate": "0", "co2Reduced": " " }
...
```

### 5.1.3.2 [Reasoning Engine](#)

#### 5.1.3.2.1 [Case LSP1](#)

Reasoning Engine's Kafka Consumer receives the following payload for the produced dataset (LSP1 - BTC).

**Table 20: LSP1 Reasoning Engine payload**

```
{
  "id": "building-btc-tower",
  "type": "Building",
  "ENERGY_SOURCE": "electricity",
  "MEASURE": "14th Floor",
  "TIMESTAMP": 12343455,
  "LOCATION": "14th Floor",
  "UNIT_OF_MEASURE": "KWH",
}
```

```
"value": 0.784
}
```

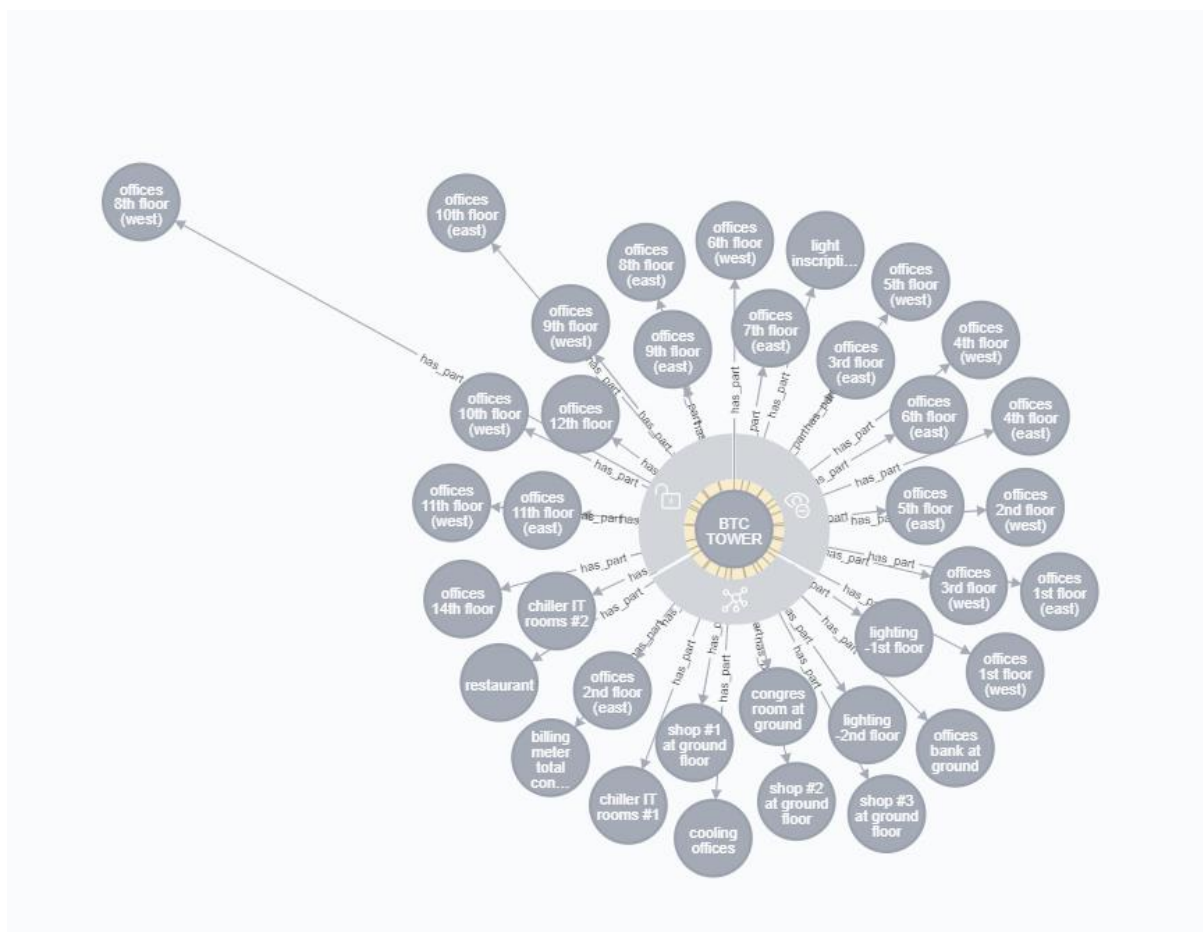
Using the following Cypher code along with Python Neo4j Client the incoming data are imported to the graph database as entities and collections and the entities created are "BTCInstance", "EnergySource", "Part" and "Consumption" and the connection that relates these entities are "has\_part", "has\_energy\_source".

**Table 21: LSP1 import script to Reasoning Engine**

```
UNWIND $batch_list AS line
WITH line
MERGE (btc_tower:BTCInstance {
  id: line['id'],
  type: line['type']
})
MERGE (energy_source:EnergySource {
  source: line['ENERGY_SOURCE']
})
MERGE (part:Part {
  name: line['MEASURE']
})
CREATE (cons:Consumption {
  timestamp: line['TIMESTAMP'],
  location: line['LOCATION'],
  unit_of_measure: line['UNIT_OF_MEASURE'],
  value: toFloat(line['VALUE'])
})
MERGE (btc_tower)-[:has_part]->(part)
CREATE (part)-[:has_consumption]->(cons)
CREATE (cons)-[:has_energy_source]->(energy_source)
```

The following figure demonstrates how the BTC data are stored to the graph database





### Figure 37: Stored LSP1 data on GraphDB

#### 5.1.3.2.2 Case LSP5

Reasoning Engine's Kafka consumer receives the following payload for the produced COOPERNICO dataset (LSP5).

Table 22: LSP5 Reasoning Engine payload

```
{
  'id': '05photovoltaicDevice',
  'Timestamp': 1615334400,
  'production': '0',
  'specificRate': '0',
  'co2Reduced': ' '
}
```

Using the following Cypher code along with Python Neo4j Client the incoming data are imported to the graph database as entities and connections. More specifically the data importers create two entities, the entity "PhotoVoltaicDevice" and the entity "DeviceMeasurement", and one connection that is called "has measurement" that relates these two entities.

**Table 23: LSP5 import script to Reasoning Engine**

```

UNWIND $batch_list AS line
WITH line
MERGE (pd:PhotoVoltaicDevice {device_id: line['id']})
CREATE (device_data:DeviceData {
  Timestamp: line['Timestamp'],
  production: coalesce(toFloat(line['production']), 0.0),
  specificRate: coalesce(toFloat(line['specificRate']), 0.0),
  co2Reduced: coalesce(toFloat(line['co2Reduced']), 0.0)
})
CREATE (pd)-[:has_measurement]->(device_data)

```

The following figure demonstrates the data after being stored and persisted to the Neo4j Graph database. The structure of entities and connections are suitable for executing Cypher queries and for extracting patterns from data.

**Figure 38: LSP5 data stored in the GraphDB**

## 5.1.4 Data Access Layer

### 5.1.4.1 Data Storage & High Performance Distributed Query Engine

Generally, access to the Data Storage and the related staging area, ScyllaDB and other data inside the MATRYCS ecosystem will be provided using the High Performance Distributed Query Engine based on Presto. Additionally, the data in the staging area may be directly accessed by applications residing on and having access to the Data Storage cloud deployment file system. Presto queries can be run using Presto Client REST API or via a Presto CLI. A query is sent to the Presto Client REST API using an HTTP POST request to the `/v1/statement/` endpoint with a SQL query string inside POST body. The query string is based on Presto SQL query language<sup>80</sup>. An example query submission using curl is available in Table 24. The `X-Presto-User` header containing the session user must be supplied with every request.

**Table 24: Example Presto query submission**

```
curl --data "show session" http://localhost:8080/v1/statement/ --header "X-Presto-User: matrycs"
```

### 5.1.4.2 Reasoning Engine

For accessing data from the graph database we have developed using the Python's Flask REST Framework a Rest API for submitting Cypher queries to the graph database. In the table below the curl request used for accessing the data.

**Table 25: Reasoning Engine Rest API**

```
curl --location --request POST 'http://reasoning_engine:5000/query' \
--header 'Content-Type: text/plain' \
--data-raw '${query}'
```

<sup>80</sup> <https://prestodb.io/docs/current/sql.html>

## 6 MATRYCS–GOVERNANCE: Final considerations and next steps

This 1<sup>st</sup> *technology release* focused on the preliminary identification and evaluation of the envisaged technological solutions for the MATRYCS-GOVERNANCE layer with limited data and scenarios. The main functionalities of the MATRYCS-GOVERNANCE components have been defined and implemented with an on-premises approach, as well as the interactions between them and the preliminary test on MATRYCS-GOVERNANCE data flows. The work done in this first release has created a valuable baseline for the next activities that will be included in the 2<sup>nd</sup> *technology release*.

Below, the future activities for MATRYCS-GOVERNANCE layer are listed:

### MATRYCS-GOVERNANCE cloud infrastructure

- Deployment of the MATRYCS-GOVERNANCE layer components in a common MATRYCS cloud infrastructure that will be identified at project level. At the time of writing, the MATRYCS project is evaluating several research cloud infrastructures (EGI FedCloud, GAIA-X, FIWARE Lab) as well as private cloud services (AWS, Azure, Google Cloud Provider).

### Trusted data sharing (DLT/Blockchain)

- Implementation of a hybrid private blockchain platform based on Ethereum technology.
- Definition and implementation of smart contracts to enable trusted data sharing mechanisms within the MATRYCS-GOVERNANCE layer.

### Interoperability Service module

- Implementation of new data connectors for full integration of MATRYCS data provided by the LSPs as well as by open data Hubs and/or external services/repositories.

### Data pre-processing & semantic enrichment

- Definition of the MATRYCS common data model.
- Implementation data pre-processing and modelling scripts.
- Integration activities with the MATRYCS Streaming module.

### Streaming module

- Definition new data streaming flows according with the new data integrated.

### High distributed query Engine and Data Storage

- Evaluating MongoDB as new technological solution for the data storage to improve performance aspects.
- Integration activities with the MATRYCS Streaming module (new MATRYCS datasets).

### Reasoning engine

- Integration activities with the MATRYCS Streaming module (new MATRYCS datasets).



### End-to-End security framework

- Implementation of the End-to-End security framework.
- Integration activities with all MATRYCS-GOVERNANCE components.

The 2<sup>nd</sup> technology release of the MATRYCS-GOVERNANCE layer will be described in detail in *D3.3 - MATRYCS-GOVERNANCE (2<sup>nd</sup> technology release) (M22)*.

