

Project Acronym: STAR
Grant Agreement number: 956573 (H2020-ICT-2020-1 – Research and Innovation Action)
Project Full Title: Safe and Trusted Human Centric Artificial Intelligence in Future Manufacturing Lines
Project Coordinator: INTRASOFT International



Funded by the Horizon 2020
Framework Programme of the
European Union

DELIVERABLE

D3.1 – Decentralized Reliability for Industrial Data and Distributed Analytics- Initial version

Dissemination level	PU -Public
Type of Document	Report
Contractual date of delivery	31/03/2022
Deliverable Leader	INTRA
Status - version, date	Final - V1.0, 20/04/2022
WP / Task responsible	WP3
Keywords:	Blockchain, Data Provenance, metadata, data reliability, trustworthiness

This document is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956573. It is the property of the STAR consortium and shall not be distributed or reproduced without the formal approval of the STAR Management Committee. The content of this report reflects only the authors' view. The European Commission is not responsible for any use that may be made of the information it contains.

Executive Summary

One of the STAR project's offers is to describe and implement a platform for safe data interchange across the STAR infrastructure's components. The transferred data may have a significant impact on the platform's operation; hence, it must be safeguarded from tampering efforts. This deliverable demonstrates the project's decentralized approach for provenance and tracking of industrial data utilized in AI systems in this setting. The solution leverages the power of distributed ledger technologies i.e., blockchains to support secure and trusted traceability of data transactions through the STAR platform.

The document begins with an explanation of why blockchain technology and smart contracts are being utilized to track and trace industrial data used in AI systems. The benefits of blockchain technology for data traceability are discussed in detail, as well as a set of definitions for the various types of blockchain infrastructures and their applicability to the problem at hand. As a result, the deliverable outlines the primary STAR criteria for data traceability and how they drove the blockchain infrastructure decision. The deliverable examines potential blockchain infrastructures versus the requirements as part of this procedure. For performance and access control reasons, the project has chosen to adopt a permissioned blockchain technology. Moreover, the Hyperledger Fabric has been selected as the baseline infrastructure for the development of the STAR distributed ledger solution.

A significant part of the deliverable is devoted to the specification of how data transactions are recorded in the blockchain to leverage its anti-tampering and data security properties. Similarly, the deliverable demonstrates the blockchain's procedures for validating data stored in the blockchain, as well as the APIs and data models that are used to access traceability information from the blockchain. Other components of the STAR platform will be able to get results and metadata about the industrial data utilized in AI systems thanks to the latter models and APIs.

Finally, the deliverable shows how the blockchain infrastructure may be used in practice. This initial deployment allows for the validation of the recommended blockchain solutions as well as the gathering of feedback from key parties. Those comments will be used to fine-tune the blockchain implementation in WP6 of the project, which involves integrating and validating the full STAR platform in several trial scenarios.

Deliverable Leader:	INTRA
Contributors:	INTRA, UPRC
Reviewers:	RUG, THA
Approved by:	INTRA

Document History			
Version	Date	Contributor(s)	Description
0.1	01/02/2022	INTRA	First document release with ToC
0.2	14/02/2022	INTRA	Added Blockchain background and motivation
0.3	22/02/2022	INTRA	Added framework placement to STAR Architecture
0.4	28/02/2022	INTRA	Added DLSDR Architecture and offered services, STAR HLF Network
0.5	8/03/2022	INTRA	Added Framework specifications
0.6	17/03/2022	INTRA	Various corrections and editing throughout the document
0.7	25/03/2022	INTRA	Added Data Reliability framework implementation
0.8	08/04/2022	UPRC	Added algorithm configuration parameter validation
0.9	08/04/2022	INTRA	Added introduction, executive summary, and conclusions
0.10	11/04/2022	INTRA	Updates throughout the document. Preparation for internal review.
0.11	14/04/2022	RUG	Internal quality review 1
0.12	14/04/2022	THA	Internal quality review 2
0.13	19/04/2022	INTRA	Address internal reviewers' comments. Preparation of the document for submission
1.0	20/04/2022	INTRA	Final deliverable approval and submission

Table of Contents

EXECUTIVE SUMMARY	2
TABLE OF CONTENTS.....	4
TABLE OF FIGURES.....	6
LIST OF TABLES.....	7
DEFINITIONS, ACRONYMS AND ABBREVIATIONS	8
1 INTRODUCTION.....	10
1.1 OVERVIEW AND PURPOSE.....	10
1.2 RELATIONSHIP TO OTHER DELIVERABLES.....	10
1.3 DELIVERABLE STRUCTURE	10
2 BLOCKCHAIN: BACKGROUND AND MOTIVATION	12
2.1 BACKGROUND ON BLOCKCHAIN TECHNOLOGY AND SMART CONTRACTS.....	12
2.1.1 <i>Blockchain Technology Overview</i>	12
2.1.2 <i>Blockchain Properties and Characteristics</i>	13
2.1.3 <i>Blockchain Types</i>	15
2.1.4 <i>Smart Contracts</i>	17
2.2 BLOCKCHAINS FOR INDUSTRIAL APPLICATIONS	18
2.2.1 <i>Rationale Behind the Use of Blockchains in Industry</i>	18
2.2.2 <i>Industrial Use Cases for Blockchains</i>	20
2.2.3 <i>Data Provenance and Reliability with Blockchains</i>	21
2.2.4 <i>Industrial Blockchain Projects Linked to STAR</i>	22
2.3 STAR BLOCKCHAIN USE CASES AND SELECTION OF BLOCKCHAIN INFRASTRUCTURE.....	22
2.3.1 <i>STAR Requirements for Blockchain Deployment and Usage</i>	22
2.3.2 <i>Overview of the Use of Blockchain in STAR</i>	23
2.3.3 <i>Non-Functional Requirements</i>	25
2.3.4 <i>Benchmarking and Comparative Evaluation of State-of-the-Art Blockchain Infrastructures</i>	26
2.3.5 <i>Critical Analysis and Selection Outcome</i>	29
3 STAR DATA RELIABILITY FRAMEWORK.....	32
3.1 FRAMEWORK PLACEMENT TO STAR ARCHITECTURE	32
3.2 DLSDR FRAMEWORK ARCHITECTURE	33
3.3 FRAMEWORK SERVICES	35
3.3.1 <i>Analytics Engine Configuration (AEC) Service</i>	35
3.3.2 <i>Analytics Results Publishing (ARP) Service</i>	36
3.4 THE STAR HYPERLEDGER FABRIC NETWORK	37
3.4.1 <i>Structural Components of the Blockchain Network</i>	37
3.4.2 <i>Ordering Service</i>	38
3.4.3 <i>Mapping Stakeholders to Nodes</i>	39
3.4.4 <i>Smart Contracts vs. Chaincode</i>	40
3.4.5 <i>Certificate Authorities</i>	40
3.4.6 <i>Membership Service Providers</i>	41
3.4.7 <i>Scalability Considerations</i>	41
3.5 ALGORITHMS CONFIGURATION PARAMETERS VALIDATION	42
3.5.1 <i>Proposed Approach</i>	44
3.5.2 <i>Quantifying the Difference between Parameter Files</i>	45
4 INDUSTRIAL DATA RELIABILITY FRAMEWORK SPECS.....	50

4.1	DISTRIBUTED LEDGER NODE MANAGEMENT	50
4.1.1	<i>Registration and Discoverability of the Platform Nodes</i>	50
4.1.2	<i>Data Model</i>	51
4.1.3	<i>API Specification</i>	52
4.2	DATA PROVENANCE & TRACEABILITY SERVICES	53
4.2.1	<i>Introduction to the Data Entities recorded on the Ledger</i>	53
4.2.2	<i>Data Sources Traceability</i>	53
4.2.3	<i>Processors Configuration Traceability</i>	57
4.2.4	<i>Algorithm Results Traceability</i>	60
4.2.5	<i>Recording Traceability Data on the Blockchain</i>	64
5	DATA RELIABILITY FRAMEWORK POC IMPLEMENTATION.....	66
5.1	BASELINE INFRASTRUCTURE: HYPERLEDGER FABRIC	66
5.1.1	<i>Services Components Deployment in Docker Containers</i>	66
5.1.2	<i>Forming a Cluster Among Distributed Services with Docker Swarm</i>	67
5.1.3	<i>Docker Swarm Overlay Topology for STAR PoC Fabric Network</i>	69
5.1.4	<i>Cloud Infrastructure Preparation</i>	71
5.1.5	<i>Outline of the Network Installation Steps</i>	72
5.2	INFRASTRUCTURE MANAGEMENT	72
5.2.1	<i>Monitoring the Swarm Network</i>	72
5.2.2	<i>Monitoring the Blockchain Network</i>	76
5.3	BLOCKCHAIN SERVICE BACKEND.....	77
6	CONCLUSIONS.....	78
	REFERENCES	79
	APPENDIX A DATA MODEL SCHEMATA.....	86
A.1	BLOCKCHAIN DATA MANAGEMENT.....	86

Table of Figures

FIGURE 1: MCKINSEY’S ANALYSIS OF BLOCKCHAIN OPPORTUNITIES BY INDUSTRIAL SECTOR	12
FIGURE 2: BLOCKCHAIN GOVERNANCE MODELS	16
FIGURE 3: TOTAL UNIQUE DEVELOPERS ENGAGING IN VARIOUS BLOCKCHAIN INFRASTRUCTURES AND PROTOCOLS [CHAINSTACK21]	30
FIGURE 4: EVOLUTION OF DEVELOPERS’ ACTIVITY FOR VARIOUS BLOCKCHAIN INFRASTRUCTURES [CHAINSTACK21]	31
FIGURE 5 STAR FUNCTIONAL MODULES AND LOGICAL VIEW OF THE ARCHITECTURE [D2.6].....	32
FIGURE 6 STAR SECURITY AND DATA GOVERNANCE FOR AI SYSTEMS IN MANUFACTURING LOGICAL VIEW	32
FIGURE 7: DLSDR FRAMEWORK ARCHITECTURE.....	34
FIGURE 8: ANALYTICS ENGINE CONFIGURATION (AEC) SERVICE OVERVIEW	36
FIGURE 9: ANALYTICS RESULTS PUBLISHING (ARP) SERVICE OVERVIEW	36
FIGURE 10: SAMPLE HYPERLEDGER FABRIC NETWORK.....	37
FIGURE 11: INTERFACING TO THE HLF ORDERING SERVICE [FABRICDOCS]	39
FIGURE 12: ILLUSTRATION OF PERFORMANCE AND SCALABILITY OF DIFFERENT FAMILIES OF POW AND BFT PROTOCOLS [VUKOLIĆ15]	42
FIGURE 13 FLOW FOR GENERAL EDIT DISTANCE COMPUTATION	44
FIGURE 14 TRAINING FLOW FOR AI-BASED CNN-SPECIFIC EDIT DISTANCE COMPUTATION	45
FIGURE 15 JSON DIFF – SAME CONTENTS BUT DISTANCE CALCULATION IS NOT STRAIGHTFORWARD [HUETTER22]..	46
FIGURE 16 S-GCNN ARCHITECTURE FROM [KIROSHEEV21]	48
FIGURE 17 NETWORK ARCHITECTURES GENERATED BY VARIANTS OF BLOCK-QNN [ZHONG18]	49
FIGURE 18: ORGANIZATION NODE IDENTIFIER ENTITY GRAPH.....	51
FIGURE 19 DATA SOURCE PERSISTENCE TO DISTRIBUTED LEDGER NETWORK	56
FIGURE 20 PROCESSOR PERSISTENCE TO DISTRIBUTED LEDGER NETWORK	60
FIGURE 21 OBSERVATION ENTITY OF THE DATA MODEL	62
FIGURE 22: METADATA PERSISTENCE USING THE STAR BLOCKCHAIN	64
FIGURE 23: METADATA VALIDATION USING THE STAR BLOCKCHAIN	65
FIGURE 24: A DOCKER SWARM CLUSTER EXAMPLE [SWARMDOCS].....	68
FIGURE 25: SWARM SECURITY CONCEPT WITH PKI [SWARMDOCS].....	69
FIGURE 26: DOCKER SWARM OVERLAY TOPOLOGY FOR STAR MVP FABRIC NETWORK.....	70
FIGURE 27: SWARMPIT MASTER DASHBOARD FOR RESOURCES MONITORING.....	73
FIGURE 28: SWARMPIT DEPICTING DEPLOYED DOCKER SWARM STACKS	74
FIGURE 29: SWARMPIT DEPICTING THE VIRTUAL MACHINES FORMULATING A NETWORK	74
FIGURE 30: PORTAINER LANDING PAGE FOR CONTAINERS ADMINISTRATION	75
FIGURE 31: PORTAINER SWARM CLUSTER OVERVIEW	75
FIGURE 32: PORTAINER SWARM VISUALIZER WITH NODES ASSIGNED TO STAR SERVICES	76
FIGURE 33: HYPERLEDGER EXPLORER DASHBOARD	77

List of Tables

TABLE 1: STAR REQUIREMENTS VS. BLOCKCHAIN SUITABILITY ASSESSMENT.....	23
TABLE 2: NON-FUNCTIONAL PROPERTIES OF THE STAR BLOCKCHAIN	25
TABLE 3: COMPARATIVE ASSESSMENT OF CHARACTERISTICS OF POPULAR PERMISSIONED ENTERPRISE BLOCKCHAIN INFRASTRUCTURES [KALEIDO19].....	27
TABLE 4 CNN CONFIGURATION INFORMATION CLASSIFIER EXAMPLE	43
TABLE 5: REGISTRATION AND DISCOVERABILITY SERVICES API SPECIFICATION OVERVIEW.....	52
TABLE 6 DLSDR FOR DATA SOURCE METADATA API SPECIFICATION OVERVIEW	55
TABLE 7 DLSDR FOR PROCESSOR CONFIGURATION METADATA API SPECIFICATION OVERVIEW	58
TABLE 8 DLSDR FOR ALGORITHM RESULTS API SPECIFICATION OVERVIEW	62
TABLE 9 STAR BLOCKCHAIN DATA MANAGEMENT XSD SCHEMA	86

Definitions, Acronyms and Abbreviations

Acronym/ Abbreviation	Title
ABAC	Attribute-based Access Control
AEC	Analytics Engine Configuration
API	Application Programming Interface
ARP	Analytics Results Publishing
BFT	Byzantine Fault Tolerance (or Tolerant)
BTC	Bitcoin
CA	Certificate Authority
CE	Circular Economy
CFT	Crash Fault Tolerance (or Tolerant)
CLI	Command-Line Interface
CRUD	Create Read Update Delete
dApp	Distributed Application
DLSDR	Distributed Ledger Services for Data Reliability
DLT	Distributed Ledger Technology
DoA	Description of Action
DPT	Data Provenance and Traceability
DSL	Domain-Specific Language
EAE	Edge Analytics Engine
EDM	Ecosystem Data Manager
ERC	Ethereum Request for Comments
GUI	Graphical User Interface
HLF	HyperLedger Fabric
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MSP	Membership Service Provider
MVP	Minimum Viable Product
OEM	Original Equipment Manufacturer
P2P	Peer-to-Peer
PKCS	Public Key Cryptography Standards
PKI	Public Key Infrastructure
PoS	Proof of Stake
PoW	Proof of Work
RBAC	Role Based Access Control
RMS	Runtime Monitoring System
RSA	Rivest–Shamir–Adleman cryptosystem
SC	Smart Contract
SDK	Software Development Kit
SLA	Service Level Agreement
TLS	Transport Layer Security
TTP	Trusted Third Party
UBAC	User Based Access Control

URI	Universal Resource Identifier
URL	Universal Resource Locator
UUID	Universally Unique Identifier
XACML	eXtensible Access Control Markup Language
XSD	XML Schema Definition

1 Introduction

1.1 Overview and Purpose

One of the objectives of the STAR project is to research and implement decentralized data provenance and reliability mechanisms for industrial data. As described in D2.6 “STAR Reference Architecture and Blueprints” some of the functionalities of the platform, notably the data reliability used in AI systems, must be enabled by a decentralized IT infrastructure, which will be empowered by Distributed Ledger Technologies (DLT). To support such functionalities and associated use cases, the project is designing and implementing a decentralized information sharing and data management infrastructure for data used in AI systems (e.g., algorithm configuration parameters and algorithm results) based on DLT technologies that are also known as blockchain technologies. The present deliverable is devoted to the specification of two main aspects of STAR blockchain technologies:

- The specification of the blockchain infrastructure to be used in STAR, which includes works towards identifying the most suitable blockchain infrastructure for the project needs.
- The specification of the ways in which the blockchain will be used in the project. As outlined in the DoA, the blockchain will be used to ensure for provenance and tracking of industrial data used in AI systems (e.g., algorithm’s hyperparameters and algorithm’s results). In this direction, the deliverable defines models that correspond to data provenance mechanisms for algorithms configuration and the results that are going to be supported.

1.2 Relationship to other deliverables

The present deliverable is closely linked to other WP2 and WP3 deliverables. Specifically, the deliverable is driven by requirements, use cases descriptions and pilot specifications produced in WP2 which are part of deliverables D2.1 “Requirements Analysis and State-of-Art Research”, D2.2 “Reference Scenarios and Use Cases for AI in Manufacturing” and D2.9 “Report on Co-Design Workshops and Focus Groups-Initial version”. At the same time, it considers the initial specifications of the STAR data models and architecture which are part of deliverables D2.4 “Data Models and Data Collection-Initial version” and D2.6 “STAR Reference Architecture and Blueprints” respectively.

Moreover, the deliverable is closely related to D3.5 (“Security and Data Governance Infrastructure-Initial version”), which is destined to specify the usage of the data reliability framework as part of the security and data governance solution.

Finally, the contents of the deliverable will be also used in D3.3 “Cyber-Defence Mechanisms against Poisoning and Evasion Attacks-Initial version” as a service for managing Poisoning and Evasion Attacks algorithms configuration and results data.

1.3 Deliverable Structure

The outline of the deliverable is as follows:

- **Section 2** provides background information on blockchain technology, blockchain types and the use of blockchains in industrial use cases.

- **Section 3** reviews the STAR blockchain requirements and presents candidate blockchain infrastructures. Accordingly, it performs a comparison and benchmarking of alternative blockchain infrastructures. Additionally, it provides an overview of configuration data AI-based validation methodologies.
- **Section 4** delves into details about the data reliability framework specifications for the distributed ledger node management and the framework's offered services.
- **Section 5** provides information about deployment and management of the STAR blockchain.
- **Section 6** is the final and concluding section of the deliverable.

2 Blockchain: Background and Motivation

2.1 Background on Blockchain Technology and Smart Contracts

2.1.1 Blockchain Technology Overview

Ledgers have been at the centre of economic transactions since the dawn of time, recording contracts, payments, buy-sell agreements, and the conveyance of assets and property. Computers have made the process of record-keeping and ledger maintenance much easier and faster during the last few decades. Business requirements, on the other hand, are not constrained by the increased efficiency brought about by machines. Other key needs accentuated in business contexts include a high level of security against devious tampering, as well as resilience against mechanical failures. Furthermore, there is an incremental requirement for transparency, which means that all transactions must be infallibly recorded and auditable, particularly when partners in an extended business ecosystem engage in trustless terms.

To address those concerns nowadays with innovation at the helm, the information stored on computer systems is moving towards much higher forms, which are cryptographically secured, fast, and decentralized. The Distributed Ledger Technologies (DLTs), as they are officially named in bibliography, have been popularized starting in the early 2010s, but capitalize on a significant amount of pre-existing research body compiled from the 1970s and onwards related to distributed systems and databases. Even though DLTs are still considered an immature technology with few production-scale propositions that can be classified as “mainstream”, experts from both the public and private sectors agree that they have the potential to disrupt in the future well-established business models in a variety of industries such as Banking and Financial services, Insurance, Digital Identity management, Law, Digital Rights management, Government, Energy, Supply-Chains and Retail, Healthcare, Manufacturing, and the list is only growing (*Figure 1*).

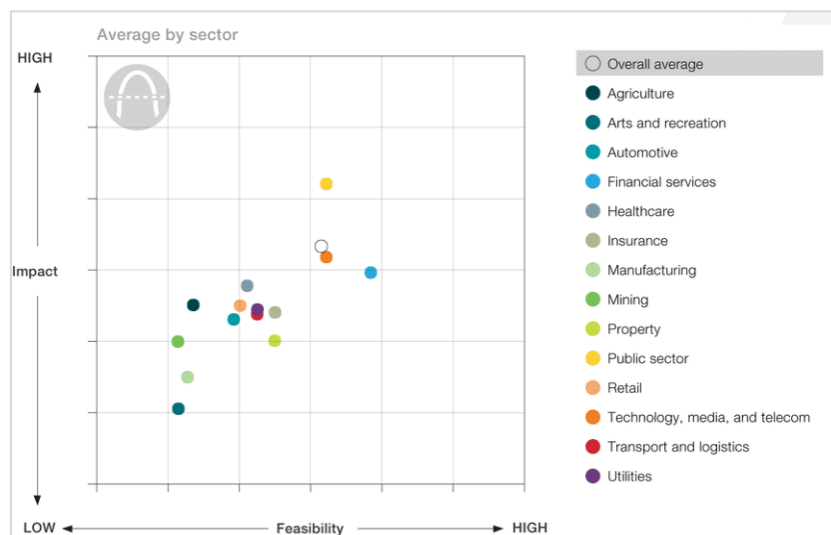


Figure 1: McKinsey's Analysis of Blockchain Opportunities by Industrial Sector¹

¹ Source: <https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/blockchain-beyond-the-hype-what-is-the-strategic-business-value>

While the terms are erroneously used interchangeably quite often, Blockchain is just the most well-known to the general public of the Distributed Ledger Technologies. Blockchain as a new technological proposition was first introduced in early 2009 to provide distributed records of monetary transactions that were not dependent on centralized authorities or financial institutions but rather relied on technology, mathematics and physics to create the necessary trusted environment [Nakamoto08]. This first incarnation, which is the framework serving as the building foundation of the infamous cryptocurrency bitcoin, is the most successful Blockchain to-date.

Various definitions for a Blockchain may be encountered in academic bibliography. For the purposes of the present deliverable, we define blockchain as a Peer-to-Peer (P2P) distributed ledger infrastructure, which is characterized by the following fundamental properties:

- Every node on the system maintains their proper copy of the digital ledger, rather than a unique version of it being stored in a central privately-owned server.
- To add a transaction to the ledger, every peer node in this network of participants needs to check its validity. Only the consensus of a predefined subset of the peers can update the ledger decidedly.
- The ledger is cryptographically secure; the integrity of data is guaranteed by employing elements such as hash values of data and hash pointers to the transaction(s) that follow chronologically.
- Blockchains group transactions into "Blocks" that formulate a "Chain". The latter is append-only i.e., the distributed ledgers grow in size with the introduction of new groups of transactions but cannot be reduced (i.e., transactions cannot be deleted). The size of an individual block is typically variable and depends on the type and conceptual design of the blockchain protocol.

As transactions are conducted and added into the various blocks of the chain, the blockchain can also be regarded as a state machine. Specifically, a blockchain infrastructure can be considered as a state transition mechanism, which takes place when the state of the blockchain changes due to the execution of a transaction and its validation by the rest nodes of the blockchain. Each stakeholder maintains one such sequence of Blocks, therefore at any given time all network participants may respond to the question of what the current global state is.

2.1.2 Blockchain Properties and Characteristics

In following sections of this document, we commonly refer to blockchain properties, assuming that the reader is familiar with them. The following paragraphs go through the typical properties of a blockchain to enhance this familiarity:

- **Blockchain Consensus - Distributed consensus:** Consensus mechanisms (also known as consensus protocols or consensus algorithms) allow distributed systems (networks of computers) to work together and stay secure. In recent years, new consensus mechanisms have been invented to allow distributed ledgers (principally cryptoeconomic systems such as Ethereum), to agree on a single equally acknowledged version of the network's current state. The STAR blockchain services will take advantage of this feature, most notably the lack of need for the involvement of any central authority (i.e., Trusted Third Party).

- **Verified Transactions:** Transactions on a blockchain are only becoming permanent once a set of pre-determined conditions that govern its operations are met. Only valid transactions (i.e., verified against this ruleset) are allowed to be encapsulated into the blocks of the chain.
- **Smart Contracts Execution Platform:** Modern blockchain-based platforms permit on top of their infrastructure the autonomous execution of programs that implement business logic on behalf of their stakeholders. Those programs are commonly referred to as “Smart Contracts”, even though they also come by various alternative protocol-specific names. Smart Contracts provide flexibility and programmability, through enabling blockchain users to configure and use the blockchain infrastructure in-line with their business requirements. It is noteworthy that some blockchains (notably the popular blockchain hosting the Bitcoin (BTC) cryptocurrency) do not provide the means for implementing and taking advantage of Smart Contracts. Nevertheless, such a necessity does exist within the context of STAR and therefore this has been taken into account to the selection of a suitable distributed ledger.
- **Blockchain Tokens and Value Transfer:** Several blockchains enable the exchange of “value” among peers in a more abstract sense than money. This value is represented in the form of digital assets, i.e., security or utility tokens. In blockchain applications tokens can indeed comprise value (i.e., they are value carriers). Nevertheless, not all blockchains support inherently tokenization. In STAR there are not tokenization use cases, yet tokenization capabilities could be used to enhance the functionalities of the STAR platform in the future.
- **Immutability:** A blockchain infrastructure is considered immutable, as the impossibility to update (benignly or maliciously) the stored transactions is among its fundamental principles. Theoretically, in some blockchains it could be possible to maliciously shift the commonly acknowledged truth to a direction proposed by a single entity, thus forcing a rollback to changes proposed by others and already accepted locally by their neighbouring nodes (a situation known as “fork”). Nevertheless, in practice manipulation of the network is very uneconomical and unattractive due to the giddy amount of resources that would be required to falsify the consensus.
- **Strong Security Features:** Blockchains as a technology proposition get very strong points as far as integrity and availability of the recorded data are concerned, since they take advantage of battle-hardened cryptographic algorithms and methodologies in a more generic sense. Nevertheless, confidentiality remains an issue of debate since in most cases all network participants may access in theory the contents of the various data blocks where transactions have been recorded. This is evidently a characteristic that can be counted as a setback as far as industrial applications are concerned, since both organizations and the Law are strict on behaviours pertaining corporate espionage and intellectual property. On such occasions the type of blockchains employed might prove to be a game-changer; for instance, in public blockchains (like Bitcoin) confidentiality is not expected to be protected and is traded for elevated transparency (with an adequate level of pseudonymity to be exact). On the other hand, private and permissioned blockchains are built on the principle that only a list of authorized stakeholders may retrieve, persist and validate the current global state of the shared ledger.
- **Cryptocurrency as incentive:** A fundamental aspect of some of the most successful blockchain protocols to-date is the fact that the validators of transactions, who gain the right to do so by proving they have invested something of value in the

communal effort (such as electricity, currency, tokens, etc.), are being incentivised to contribute by receiving cryptocurrencies as rewards. In other words, the stakeholders that go the extra mile to help with transaction validations and overall network robustness against attacks are rewarded for their efforts. Cryptocurrency generation is an extremely useful feature of blockchains at the core of a variety of use cases, mostly around the financial sector. Nevertheless, cryptocurrencies are not a necessity in the context of the STAR blockchain.

2.1.3 Blockchain Types

The different properties and characteristics of blockchain networks are also reflected in the various blockchain types available. Specifically, different types of blockchains exhibit different sets of characteristics. Some of the most prominent blockchain types follow:

- **Public blockchains:** These are publicly accessible blockchain networks i.e., anyone can join them by maintaining a peer blockchain node. This is the reason why they are sometimes called permission-less blockchains and differentiated from permissioned blockchains which are described in following paragraphs. All participants to the blockchain network maintain a copy of the blocks of the chain locally and have the right to create and validate transactions. Moreover, all participants to a public blockchain can participate in the consensus mechanism.
- **Private blockchains:** Contrary to public blockchains, private ones are joined by a closed group of participants. The latter are organizations or individuals that have agreed to form a blockchain network. In this case the contents of the ledgers are only shared within the limited number of participants of the private network.
- **Semi-private blockchains:** As the name indicates, semi-private blockchains comprise a private and a public part. The public part of the ledger is open to anyone, while access to the private part is restricted by a closed group. It can be thought as a combination of a private and a public blockchain in a single infrastructure.
- **Permissioned Blockchain:** A permissioned blockchain infrastructure comprise members that are known and trusted. Hence, it provides fine-grained control over the parties that can participate in the blockchain. Only known and trusted members can participate. One of the main characteristics of a permissioned blockchain is that it can operate on lightweight agreement protocols, rather than based on computationally expensive distributed consensus mechanisms. This allows permissioned blockchains to offer much better performance than other ledgers, in terms of the number of transactions per second that they can support. The performance and membership control characteristics of permissioned blockchains makes them an excellent choice for blockchain-based industrial applications. Note that permissioned blockchains can be either private (i.e., restricted to members of a closed group) or public (i.e., open to specific trusted and approved members).
- **Customized, fully private and proprietary blockchains:** These are special types of blockchains that are customized to enable data sharing with authenticity guarantees within a specific organization. They are useful for data sharing across different departments of a single organization.
- **Tokenized blockchains:** These blockchains employ cryptocurrencies as part their consensus mechanisms and to incentivise fair play. The generation of cryptocurrencies takes place through mining or as part of the initial distribution of the chain.
- **Tokenless blockchains:** These are “fake” blockchains that do not support transfer

of value between the participants. They are used in applications where there is no need for transferring value between the participants, but rather the primary goal is to share data between the nodes of the blockchain.

Arguably the most important classification of Blockchains, at least from the point-of-view of enterprise stakeholders, is the governance model employed by each protocol. The four primary types of Blockchain governance models [Sovrin2020] are exhibited in *Figure 2* below:

		Validation	
		Permissionless	Permissioned
Access	Public	<ul style="list-style-type: none"> Anyone may operate a node or validate transactions Anyone can create transactions on the ledger <p>(e.g., Bitcoin, Ethereum)</p>	<ul style="list-style-type: none"> Permission from some governing entity is required to operate a node or validate transactions Anyone can create transactions on the ledger <p>(e.g., Sovrin)</p>
	Private	<ul style="list-style-type: none"> Anyone may operate a node or validate transactions Single centralized organization restricts ability to write to ledger, read permissions either public or restricted <p>(e.g., Hyperledger Sawtooth)</p>	<ul style="list-style-type: none"> Permission from some governing entity is required to operate a node or validate transactions Single centralized organization restricts ability to write to ledger, read permissions either public or restricted <p>(e.g., Ethereum Enterprise, Hyperledger Fabric)</p>

Figure 2: Blockchain Governance Models

Another classification of the various blockchains can be based on the evolution of their functionalities, which are used to categorize blockchain platforms in different generations as follows:

- 1st Generation of Blockchains - Blockchain 1.0:** This refers to the first blockchains introduced and used along with the Bitcoin network. They are used for supporting the various cryptocurrencies, through applying blockchain protocols (e.g., consensus mechanism) to avoid the double-spending problem. Along with cryptocurrencies, the first generation of blockchains was also used to support transfer of value and financial transactions like payments.
- 2nd Generation of Blockchains - Blockchain 2.0:** This refers to blockchain networks that support Smart Contracts. They were used to support more complex financial transactions, including representation of financial assets like derivatives and bonds.
- 3rd Generation of Blockchains - Blockchain 3.0:** This generation leverages Smart Contracts to support decentralized application in sectors other than digital finance. They are used, for example, to support industrial applications in areas like manufacturing, energy and supply chain management.

It is evident that Smart Contracts play a key role in the implementation of decentralized applications such as the ones considered in STAR. This is the reason why we briefly introduce Smart Contracts in the following paragraph.

2.1.4 Smart Contracts

Smart Contracts provide the means for extending distributed ledgers from being a record keeping system to a middleware, a kind of “machine” or “platform” in the way the Internet is, executing complicated decentralized applications over blockchain. Smart Contracts can be simply defined as secure and unstoppable programs that implement automatically executable and enforceable agreements between various participants. In the context of Ethereum, for example, they directly control the transfer of digital currencies and assets between parties once certain conditions are met. Smart Contracts not only define the rules related to an agreement in the same way that a traditional contract does, but they can also automatically enforce those obligations, as long as the common properties of a traditional contract such as confidentiality, payment terms, and enforcement rules in a trusted way. [Daskalakis20]

The main properties of a Smart Contract are:

- **Automatic executability:** i.e., the rules that constitute the agreement between shareholders are automatically adhered to as long as the preconditions are met.
- **Enforceability:** i.e., the rules of a Smart Contract are unfailingly enforced, without a third impartial entity overlooking the process.
- **Unstoppability** i.e., the contract once set in motion cannot be stopped (or reversed for that matter).
- **Semantical soundness and univocalness:** i.e., being code, a smart contract is characterized by semantic integrity and is unequivocal in terms of the rules that it hosts. In some cases, it is important for it to be readable and comprehensible by both computers and people.

A Smart Contract has its own state and can take custody over assets on the blockchain. One of its most noteworthy roles is to establish relationships driven by data. It maintains a unique address within the blockchain network, which messages/transactions can evoke to trigger actuations. A properly written Smart Contract should describe all possible outcomes of a real-world scenario including “unhappy paths”; for instance, what happens if a party fails to provide the amount of money they have committed to or what happens once a party requests data they are not authorized to see. Additionally, a Smart Contract is deterministic; in other words, the same input must always produce the same output [Christidis16].

Since the Smart Contract resides on the Blockchain, all network participants have access to its code should they feel the need to inspect it (note: this is also a potential vulnerability if the code is poorly written and security auditing has been neglected). Furthermore, all network stakeholders receive a cryptographically verifiable record of the contract's operations history.

Nick Szabo, the American computer scientist with legal background who coined the term “Smart Contract” way back in 1994, recognized amongst their most important benefits the following [Szabo17]:

- Smart contracts reduce mental transaction costs, that is, doing what the human mind

cannot do efficiently, accurately, or dispassionately on its own.

- Smart contracts, drafted in a mathematical language executable by machines, are characterized by increased predictability thus enabling more accurate loss expectations and risk management.
- Traditional contracts tend to leave holes and are disconnected from actual control over assets, whereas Smart Contracts are based on control over assets and can provide broad security in business dealings.

It's worth noting that the concept of Smart Contracts predates the invention and popularization of Blockchains. As a result, Smart Contracts have been discussed and evolved independently of blockchain technology for many years. Nevertheless, correlating them with the notion of Blockchains has been a breakthrough, as the notion of Smart Contracts blends nicely with distributed consensus mechanisms. Blockchain 2.0 and Blockchain 3.0 infrastructures such as Ethereum and Hyperledger Fabric are examples of blockchains that provide the foundation for the development and deployment of Smart Contracts.

2.2 Blockchains for Industrial Applications

2.2.1 Rationale Behind the Use of Blockchains in Industry

Since its inception, blockchain technology presented important advantages for the implementation of cryptocurrencies. This is reflected on the momentum of blockbuster cryptocurrencies like bitcoin and ether, as well as on the fact that these cryptocurrencies represent the largest scale blockchain applications nowadays. Nevertheless, several blockchain applications for other sectors have been developed, including applications for energy management and energy trading (e.g., [Sanseverino17], [Li18a], [Mylrea17]), industrial automation (e.g., [Isaja18], [Barenji18], [LiLiu18]) and 3D printing copyright protection in manufacturing [Kennedy17], as well as for various supply chain management applications (e.g., [Bhardwaj18], [Tian16], [Tian17]).

The industrial applications discussed in the aforementioned academic sources, benefit from their underlying distributed ledger infrastructures in two ways; first by leveraging its inherent enhanced security and elevated robustness against attacks; second by taking advantage of innovative business models that do not require a single party to act as an administrator of operations and guarantor of integrity in trustless business environments. However, proof-of-concept and pilot distributed ledger deployments in different industrial contexts have also, expectedly one can say, unveiled some of the limitations of said technologies for such type of applications. For example, the latency in the conclusion of blockchain transactions is in several cases unacceptable for industrial applications that handle vast amounts of data and require real-time responsiveness. Likewise, blockchain technologies lag in user friendliness of tools and in community support, when compared to mainstream centralized architectures, such as those hosting a centralized data lake within a cloud infrastructure. Last but not least, there is always the debate on the trade-off between confidentiality and security of recorded information already discussed in a previous section. Hence, industrial end-users and practitioners are sometimes conservative and reluctant to consent to the use of DLT in industrial processes. Based on past experiences and lessons learnt, blockchain tends to be a suitable solution in industrial use cases when [QU4LITY-D3.11]:

- There is a need for a shared data storage.

- There is a need for a data storage with multiple writers.
- The network participants interact for a common business cause but do not fully know and trust each other's motives.
- Events are taking place automatically once (and only if) certain conditions are met. In addition, they ought to be unstoppable, and a reaction of the other party is expected to follow. For example, in retail a purchase is expected to be recompensated with money and delivery failure is expected to be penalized with a refund.
- The existence of a central globally trusted overseeing entity (i.e., Trusted Third Party (TTP)) is inappropriate or undesired or non-feasible or at least is perceived as having high-risk single-point-of-failure potential.
- Interactions between network participants, be them monetary or not, ought to be transparent (always recorded and auditable by all stakeholders, even more regulators and legal supervisors).
- The privacy of data as well as the protection of its ownership is of paramount importance.
- A log or history of interactions must not only be available to all participants but, also, must be safe against tampering.
- There is a need for interactions between transactions in the database i.e., transactions are not atomic and isolated.

When most of the above conditions are met, a blockchain is expected to add considerable value towards the goals dictated by the business model. On the other hand, the use of distributed ledgers is ill-advised in industrial contexts where:

- All parties fully trust each other e.g., departments of the same organization, government organizations in the same country, people who often meet physically and so on.
- A third party is unanimously trusted to ensure trustworthiness and fair motives across the ecosystem with impartiality and with no personal gain.
- The system must respond with actuations in (almost) real-time.
- Over-the-top scalability is required (e.g., big data recorded every few seconds).
- The use case requires reversibility of transactions.
- A battle-hardened conventional data storage is already in place and unfailingly provides the service it is expected to.

In these cases, the use of a centralized solution is a viable and, in most cases, a better alternative than a blockchain. Generally speaking, blockchains ought not to be frivolously thought of as miraculous solutions to problems, neither be considered just as a-bit-more-than-average secure databases. The selection of the STAR blockchain use cases takes into account the above-listed criteria/conditions, to ensure that the project benefits from blockchain capabilities in cases where conventional solutions are lacking in effectiveness. Note that industrial stakeholders exchanging information are faced with the need to interact without always trusting each other. Therefore, many use cases in this domain can be smartly served by blockchain implementations.

2.2.2 Industrial Use Cases for Blockchains

In recent years, the research community has experimented with the use of blockchain in various Industrial/Industry 4.0 applications [Bodkhe20]. Typical Use cases of blockchain functionalities for industrial applications include:

- Proactive Security, Data Protection and Privacy Preservation:** The architecture and principles of a blockchain enable a preventive approach to security and privacy preservation, as blockchains impose several restrictions by design [Noizat15]. The latter restrictions make it much more difficult to compromise a blockchain based on popular attacks (e.g., DDoS, spoofing, data rate alternation), when compared to conventional centralized systems. Safeguarding security and data protection via blockchain technologies can be very important for certain industrial applications such as automotive production [Fraga-Lamas19]. Blockchain offers many features that enable secure industrial applications, including [Hirsh18]: (i) The use of decentralized ledgers and distributed databases that have much lower risk of being compromised and becoming corrupted; (ii) The linking of transactions with cryptographic keys in the scope of an immutable ledger, which makes it almost impossible to change/delete any pieces of information; (iii) Its immutability properties which are empowered by timestamps and a consensus mechanism. Overall, blockchain technology is suitable for providing strong security and privacy preservation in any industrial sector.
- Data Provenance and Traceability in Supply Chain Management:** Blockchains are very commonly used in the scope of supply-chain management applications in various industries including for example the drugs and pharmaceuticals chains [Radanović18]. In these applications blockchain offers trusted data exchange, along with data auditing, traceability, and provenance, without a need for some trusted third party. In general, the use of blockchain in supply chain management is due to the auditability, traceability, and transparency that it offers [Queiroz19].
- Real-Time Distributed databases:** These applications use the power of Smart Contracts to obviate the need for conventional paper-based contracts. The auditing and validation of the contracts is performed by means of distributed consensus and the use of blockchain Smart Contracts. Real-time distributed databases offer seamless integration of Industry 4.0-based applications [Holland18].

As already outlined, based on the above-listed properties and benefits, blockchains are used in various industrial sectors, including:

- Supply Chain Management and Logistics:** Many advanced supply chain management and logistics applications leverage IoT technologies for real-time provenance and traceability of information. Blockchains have been used to enhance the trustworthiness of such systems and ensure that information is shared in a secure way [Tian17]. Applications have been proposed in various types of supply chain, such as pharmaceuticals discussed above, the food supply chain and the agricultural supply chain [Leng18].
- Energy Management:** Many blockchain applications for the energy sector have been developed during recent years, including various practical applications [Zhao19]. One of the most prominent blockchain use cases in energy is distributed/decentralized energy management [Kumar18]. Another prominent use case is the decentralized establishment of microgrids [Goranovic17]. In this direction, many systems have been developed including for example open sources projects like

PowerLedger [Wang19].

- **Manufacturing:** Various blockchain applications have been proposed and deployed in manufacturing plants, spanning the areas of decentralized data sharing at the shopfloor and machine levels [Barenji18], as well as the establishment of overlay peer-to-peer networks in manufacturing environments (e.g., [Angrish18], [Li18b]). There are also applications for decentralized control [Isaja18], in addition to various supply chain management applications.
- **Smart City and Public Infrastructures:** Various blockchain architectures for IoT-enabled applications in smart cities have been proposed (e.g., [Sharma18]). Various approaches that improve performance, efficiency, and security [Rivera17] for various applications like smart parking of vehicles, smart cleaning, smart home, and intelligent transport management have been proposed [Mistry20].

The above-listed applications and sectors is not exhaustive, yet representative of the use of blockchain in industrial applications. A broader set of applications use cases and sectors that take advantage of blockchain technology can be found in [Bodkhe20].

2.2.3 Data Provenance and Reliability with Blockchains

One of the most prominent applications of blockchain in all the above-listed sectors concerns provenance and traceability. Blockchain-based data provenance offers the following advantages in comparison to tracing data in a centralized database:

- **Increased resilience:** Blockchain infrastructures are decentralized and therefore pose severe challenges to malicious parties attempting to hack them in comparison to centrally hosted data storages. Blockchains' inherent properties ensure that changes in the status of data entities are performed when most of the parties behaves in a lawful and ethical way.
- **Decentralized trust across data writers that belong in different trust domains:** Data provenance is usually achieved based on the collaboration of multiple writers that provide information about different data entities. In many industrial applications (e.g., supply chain applications) these writers belong in different trust domains (as a matter of fact they may even be competitors). Blockchain applications help lower the trust barriers towards sharing data traceability information.
- **Tamper-proof operation:** The anti-tampering properties of blockchains make them ideal for tracking and tracing data and their properties (e.g., data configurations and metadata) since trust to their integrity is backed by mathematical proofs.

Given these attributes, many blockchain systems for industrial data provenance have been proposed in the literature. For example, Provchain [Liang17] enables auditing of data operations over cloud storage in real-time, while supporting access control and intrusion detection. It also supports privacy preservation features by preventing a direct correlation between users and provenance records, using a hashed user ID. As another example, the ProductChain [Malik18] keeps track of processes in the food supply chain i.e., deals with the provenance of food related data entities. A permissioned blockchain is employed along with a transaction vocabulary for the target domain. The system provides interfaces that enable consumers and other stakeholders to access food product provenance information, without disclosing information about trade flows. There are also blockchain systems for manufacturing chains (e.g., composite materials traceability) [Mondragon18] and other

agricultural products [Hua18]. The latter are recorded in terms of their identity, species name, planting-time, company-name, greenhouse number, and geographical location. Likewise, provenance records about agricultural processes include information about identity, date & time, person, digital-signature, location, operation type, and company. One more noteworthy real-world example is SmartProvenance [Ramachandran18], a system that uses Smart Contracts and consensus mechanisms to provide reliable data provenance. The system supports privacy preservation using public key encryption and digital signatures.

To conclude, data provenance monitoring is an excellent use case for Industry 4.0 platforms as it can alleviate the trust barriers for data logging, while increasing the security and smooth operation of the data traceability process.

2.2.4 Industrial Blockchain Projects Linked to STAR

The EC has funded several EU projects that use blockchain in various sectors². The STAR partners participate or have participated in blockchain projects in the areas of manufacturing and supply chain management, which are thematically most relevant to STAR. H2020 FAR-EDGE³ project leverages a permissioned blockchain for secure and trusted decentralized management of distributed control and distributed data analytics transactions in manufacturing environments [Soldatos19], [Isaja18]. Furthermore, STAR partners participate in the H2020 QU4LITY⁴ project that develops digital platforms for quality management and zero-defect manufacturing. QU4LITY includes a blockchain component/platform as a clearing house element for supply chain transactions. What is more, H2020 DIGIPRIME⁵ is another project where a blockchain is employed to ensure security and traceability of data sharing between partners on a Circular Economy consortium [Soldatos21]. As illustrated in the following section, STAR takes advantage of prior experiences in these projects in the selection and setup of its blockchain infrastructure, as illustrated in the following section.

2.3 STAR Blockchain Use Cases and Selection of Blockchain Infrastructure

2.3.1 STAR Requirements for Blockchain Deployment and Usage

In-line with the analysis in the previous section, STAR will leverage blockchain to provide high levels of data transparency, integrity and availability in use cases where sophisticated AI systems are developed in production lines. This concept is generally in-line with the use of blockchain infrastructures in conjunction with applications employing Artificial Intelligence algorithms. It is also in-line with the rationale behind using distributed ledgers to support operations in the context of industrial applications. Specifically, the exchange and sharing of metadata on algorithm configurations and data sources, as well as calculation results, between the stakeholders of an industrial ecosystem fulfils the preconditions that make blockchains a suitable choice, as illustrated in the following table (Table 1).

² <https://ec.europa.eu/digital-single-market/en/news/eu-funded-projects-blockchain-technology> (accessed April 2022)

³ H2020 FAR-EDGE at CORDIS: <https://cordis.europa.eu/project/id/723094> (accessed April 2022)

⁴ H2020 QU4LITY at CORDIS: <https://cordis.europa.eu/project/id/825030> (accessed April 2022)

⁵ H2020 DigiPrime at CORDIS: <https://cordis.europa.eu/project/id/873111> (accessed April 2022)

Table 1: STAR Requirements vs. Blockchain Suitability Assessment

Conditions Leading to Blockchain Use	STAR Requirements & Examples
Need for a shared storage of data	Stakeholders of the STAR digital platform have the need to persist, exchange and verify (meta) data regarding data sources, AI processor configurations and processing results.
Multiple Writers	The data and metadata used across the STAR digital platform are produced/recorded by multiple actors and at various geographical locations. In addition, a variety of AI algorithms is being employed, created by different research teams towards different ends.
Non-Trusting Writers	An ecosystem of industrial organizations (both commercial-oriented and research-oriented ones) may cooperate towards a common goal, while having simultaneously reasons to be secretive regarding details of their inner operations (most commonly to protect intellectual property from industrial espionage).
Not appropriate or feasible to rely on TTP	It is not always straightforward within an ecosystem of collaborating industrial actors which one can be agreed upon to act as single regulator that everybody can trust (i.e., a TTP) to control data storages and perform data integrity auditing.
Access limited to Platform shareholders	Access to the (meta) data stored should be restricted to a group of trusted project stakeholders, including new participants that will opt for registering in the STAR platform.
Need for Transparency and Auditability	Updates to the (meta) data storage ought to be recorded in some sort of log and be available for a revision when needed.
Privacy and ownership of data is of paramount importance	Even though the industrial data will not be directly stored on the Blockchain (only pointers to their location and metadata), still the datasets contain sensitive information protected by regulation and corporate confidentiality rules.
Protection against tampering from malicious actors	It is self-explanatory why protection from malicious behaviour is a requirement for all enterprise applications.

2.3.2 Overview of the Use of Blockchain in STAR

According to STAR DoA, the role of blockchain in the STAR platform is mainly focused on providing a decentralized infrastructure for provenance and tracking of industrial data used in AI systems, notably data stemming from various industrial sources including sensors, industrial automation devices, robots and business information systems. In-line with this vision of the DoA, STAR opts for an approach, where:

- **Industrial data (such as those collected from sensors on factory shopfloors) are stored in centralized repositories**, as illustrated in deliverables D2.4 «Data Models and Data Collection - Initial version» and D2.6 «STAR Reference Architecture and Blueprints - Initial version».
- **Metadata (e.g., data sources configuration, AI processors/algorithms configurations, processing results) are stored on a blockchain infrastructure.** In essence STAR leverages distributed ledger technologies for achieving data provenance and traceability, as well as for enhancing the trust on AI-related operations.

In essence, the STAR blockchain services will provide data provenance and traceability functionalities, along with decentralized, trusted, and auditable control of AI algorithms configuration and results. Nevertheless, the STAR blockchain could have been used for storing actual data collected from the edge. The latter is theoretically possible and could be even demonstrated in the scope of STAR. Nevertheless, it is also subject to the following limitations:

- **GDPR and Intellectual Property Regulation Compliance Limitations:** For instance, storing data within the blockchain infrastructure contradicts the “right to be forgotten” principle of the GDPR regulation. Given the anti-tampering property of the blockchain, any data written in the distributed ledger cannot be deleted. Hence, it is not possible for data owners to request and achieve deletion of their enterprise data from the blockchain in-line with their right to be forgotten. This is a much-debated issue regarding blockchains and various clever techniques for overcoming it have been proposed and already implemented in various projects. Nevertheless, there are still doubts about the legal validity and acceptance of these solutions i.e., they constitute a grey area for several blockchain stakeholders (e.g., legal experts, end-users).
- **Usability Limitations:** Participation of industrial actors as peers in a blockchain network expects them to run (or at least have assigned to them) a ledger node as a key component for sharing data with the STAR platform. This approach might complicate things from the usability and deployment perspective, especially if said organizations do not employ developers with prior blockchain experience. STAR opts for an approach that enables organizations in industrial ecosystem to make use of their existing/legacy applications for interacting with the STAR platform.
- **Scalability and Performance Limitations:** Even though blockchain can be used for storing and validating large volumes of data, it does not offer the scalability and Quality of Service (QoS) of state of the art (centralized) Big Data architectural proposals. For instance, the handling of high volume and high velocity data is typically among a blockchain’s worst enemies.
- **Lack of Community Support:** Moreover, the state-of-the-art mainstream Big Data technical propositions (both commercial and open source) have almost always larger developer communities and a more mature ecosystem of tools extending their capabilities, which facilitates development and deployment of applications without “re-inventing the wheel”.

On the other hand, the use of the blockchain infrastructure for provenance and tracking of industrial data used in AI systems in the context of STAR offers the following advantages:

- **Decentralized Trust:** Storages implemented using peer-to-peer networks as base (as opposed to those being hosted on a central server) bear the advantage that

transactions are verified by the entire community, the industrial actors themselves, and therefore there is no need to rely on a third trusted party or a peer with elevated privileges.

- **Safety:** Safety is an inherent feature of the blockchain technology that uses popular encryption techniques. All transactions are encrypted, and it is practically impossible to tamper with the chain of transactions recording metadata on the shared ledger. This will enable industrial actors to treat the blockchain as a single and credible version of the status of recorded information.
- **Irreversibility:** The moment a transaction is validated, and a block is created and recorded on the blockchain, it cannot be cancelled. The only way to reverse a storage transaction is to create a new one with the opposite direction.
- **Traceability of Data Entities:** Any data transactions on the common distributed storage, for example new results stemming from the execution of an AI algorithm or the reconfiguration of an algorithm, will be fully traceable and verifiable through the various blocks of the STAR blockchain.

2.3.3 Non-Functional Requirements

In order to support the implementation of the STAR industrial data provenance and traceability framework, the non-functional properties that are listed in the following Table 2 must be met.

Table 2: Non-Functional Properties of the STAR Blockchain

Requirement	Description
Programmability Capabilities	Support for Smart Contracts to programmatically support business logic must be provided.
Development Operations	Introduction of an additional stakeholder in the existing Blockchain network - a process that in most platforms is quite tedious - must be supported by a sequence of steps that ought to be as highly automated as possible.
Scalability	Undefined number of organizations or nodes (at least three for an POC). Processing of at least 100s of transactions per second.
Latency	Transactions should be completed in the time frame of seconds or milliseconds.
Consensus Algorithm	Since participants in the platform are in essence business partners, malicious behaviour is less likely to take place. Therefore, an efficient consensus mechanism is preferable rather one with built-in inhibitors that render the transaction validation tedious.
Use of Token or Cryptocurrency	Not required.
Energy	For financial, environmental and ethical reasons, the Blockchain infrastructure must not devour excessive energy/electricity.

Maturity	A mature and validated blockchain infrastructure with proven community support is required.
UX Design	Both end-users and platform component developers must interact with the Blockchain in a way that does not have as prerequisite prior experience and expertise with the intricacies of said technology.

The requirements listed on Table 2 have been heavily taken into account and eventually determined the selection of the optimal protocol to serve as the underlying technology of the STAR blockchain infrastructure. Specifically, a variety of distributed ledgers were evaluated and confronted against these requirements towards selecting the one best fitting the needs of the project. In this direction, the project exploited background expertise of the partners, along with readily available benchmarking/comparison processes conducted in prior research projects.

2.3.4 Benchmarking and Comparative Evaluation of State-of-the-Art Blockchain Infrastructures

As part of the STAR blockchain selection process, we have researched evaluations of various blockchain infrastructures. Businesses and regulators have strong requirements on identities and permissioning, which steered us towards a permissioned over a permissionless (that is public) blockchain protocol. Three blockchain protocols have emerged as the primary candidates to implement an enterprise production-quality permissioned chain: Hyperledger Fabric ⁶ (originally by IBM), Enterprise Ethereum ⁷, and R3 Corda ⁸. Fabric and Enterprise Ethereum are both general-purpose tools that may be used in any industry, whereas Corda is specifically developed for the financial sector. Each protocol choice has a distinct pedigree and design focus, but all three have received widespread adoption by serious businesses and governments, and all three are currently running on production-quality systems.

Clarification: There are four major Enterprise Ethereum implementations active in the enterprise scene. Three of them are modified from a public Ethereum client, with a fourth developed from scratch. Those are Consensys Quorum ⁹ (originally by JPMorgan Chase modified from go-ethereum), Hyperledger Besu ¹⁰ (originally by Pegasys newly implemented in Java), Autonity ¹¹ (by Clearmatics modified from go-ethereum) and Strato ¹² (by BlockApps modified from Haskell Ethereum).

At a high level, the differences between Enterprise Ethereum vs Hyperledger Fabric vs R3 Corda can be summarized as follows (Table 3):

⁶ Hyperledger Fabric official webpage: <https://www.hyperledger.org/use/fabric> (accessed April 2022)

⁷ Enterprise Ethereum Alliance official website: <https://entethalliance.org> (accessed April 2022)

⁸ Corda official website: <https://www.corda.net/> (accessed April 2022)

⁹ Consensys Quorum official website: <https://consensys.net/quorum/> (accessed April 2022)

¹⁰ Hyperledger Besu official website: <https://www.hyperledger.org/use/besu> (accessed April 2022)

¹¹ Autonity official website: <https://autonity.io/> (accessed April 2022)

¹² Strato official website: <https://blockapps.net/> (accessed April 2022)

Table 3: Comparative Assessment of Characteristics of Popular Permissioned Enterprise Blockchain Infrastructures [Kaleido19]

Characteristic	Enterprise Ethereum	Hyperledger Fabric	R3 Corda
Node Permissioning	Smart contract-based rules, with file-based per-node rules as local overrides.	Configurable on node, channel and consortium levels.	Trusted network map service complemented by file-based configurations on each node. Corda networks are partitioned into compatibility zones that are governed by separate Certificate Authorities.
Identity	Public keys - distributed, and interoperable between Ethereum based chains. Coupled to PKI via proofs.	Based on PKI with native organizational identity. Organizational identity rather than individual identities used throughout in consensus, and permissioning.	Based on PKI with both individual and organizational identity.
Cryptography	secp256k1	Pluggable (ECDSA with secp256r1 and secp384r1 built-in).	ed25519 secp256r1 secp256k1 RSA (3072bit) PKCS#1 SPHINCS-256 (experimental)
Transaction Consensus	Order -> Execute/Validate	Execute -> Order -> Validate	Execute/Validate -> Order/Notarize
Application Responsibility	Sending signed transactions to one node in the network.	Coordinating directly with all other participants to obtain endorsement, managing optimistic concurrency locking on state, signature, and submission.	CorDapps use the flow framework to coordinate with transaction counter-parties to negotiate proposed updates, obtain signatures, and to finalize with the notary service.
Applied Consensus Algorithms	Proof-of-Authority (BFT). Raft (CFT with trusted leader). Istanbul BFT (BFT with	Kafka/Zab (CFT with trusted leader). Raft (CFT with trusted leader).	Raft (CFT with trusted leader) BFT

	deterministic leader rotation). Tendermint		
Smart Contract Engine	EVM, in-process sandbox	Docker isolation	Deterministic JVM
Smart Contract Languages	DSL (Solidity, Serpent), guaranteed deterministic.	Full languages (Go, Node.js, Java), non-determinism is tolerated.	Java, Kotlin, deterministic by using recommended libraries
Smart Contract Lifecycle	Immutable. Easy to deploy. Stored on-chain.	Requires elaborate process to deploy/change. Stored off-chain.	Requires node-level administrative operations to deploy/update. Stored off-chain. Ongoing work to split consensus-critical code vs. non-consensus-critical code for different storage strategy (on-chain vs. off-chain respectively).
Smart Contract Upgrade	Programming patterns to extend/migrate code & data.	Replacing off-chain code via administrative procedure and upgrade transactions.	Contracts with hash-based constraints are explicitly upgraded via node-level administrative procedures and coordinated flow to authorize and upgrade. Contracts with signature constraints automatically allow new versions to execute, as long as signed according to the constraints and the hash matches.
Tokenization of Assets	Native feature Many token standards: ERC20/ERC721/ERC777 etc.	Possible with custom solution.	Possible with custom solution. Corda Token SDK makes it easier to build.
Multi-chains	Each chain is unique and requires separate	Native feature (channels) with shared	No concept of a chain (shared ledger).

	node runtimes (min or 3 or 4 depending on consensus).	peer runtime, and shared orderer. Built-in governance for creating side-chains with isolated state.	Transactions always explicitly target specific nodes. States are scoped to the designated notary which can be retargeted to a different notary.
Private Transactions	Public hash represents input.	Public hash represents input and private end state.	Inherently all transactions are private. The entire transaction is visible to a validating notary.
Throughput	25-200 tx/s	3.5k-110k tx/s	15-1700 tx/s
Community of Contributors (as of writing)	Go-Ethereum: 732 Quorum: 471 Besu: 123 Autonity: 494	Fabric: 304	Corda: 187
Community Pulse (Month of Nov. 2019)	Go-Ethereum: 15 authors, 98 PRs Quorum: 9 authors, 13 PRs Besu: 23 authors, 66 PRs Autonity: 6 authors, 6 PRs	Fabric: 31 authors, 220 PRs	Corda: 33 authors, 91 PRs

2.3.5 Critical Analysis and Selection Outcome

Considering the above comparisons and benchmark, Hyperledger Fabric (HLF) has been selected as the blockchain infrastructure for STAR. The rationale behind this selection lies in the following arguments:

- Hyperledger Fabric provides versatility in implementing private blockchain networks for enterprise use:** HLF enables the implementation of permissioned blockchain infrastructures, which is the type of blockchains that are best suited for enterprise applications.
- Hyperledger Fabric exhibits the best throughput:** In addition to controlling participation to the networks, permissioned blockchain offer much better performance and throughput than public blockchains. Also, HLF can achieve substantially more transactions per second in comparison to the other enterprise oriented blockchains examined above.
- Hyperledger Fabric provides flexibility in the development of custom**

decentralized applications: HLF provides flexibility in application development thanks to support for custom data models and for Smart Contracts programming. As such it provides a very good basis for the development of custom decentralized applications in STAR. What is more, Smart Contracts can be implemented in already popular programming languages such as JavaScript, Go and Java, rather than blockchain domain-specific ones (as is the case with Enterprise Ethereum).

- **Hyperledger Fabric is generic purposed:** HLF does not specialize in applications addressed to a particular sector as is the case, for example, with R3 Corda which is oriented towards financial business cases.
- **Hyperledger Fabric does not require participants to exchange tokens nor consumes excessive energy:** This advantage derives from the consensus mechanisms used (instead of Proof-of-Work) and the absence of a mining process.
- **Hyperledger Fabric is being supported by business leaders:** Large enterprise firms like IBM, Intel, and Cisco, as well as The Linux Foundation, support Hyperledger Fabric development. This will mitigate the trepidation that organizations that are unsure of the future and potential of Blockchain might share.
- **Hyperledger Fabric has good documentation and a vibrant community of developers:** This is an important requirement that can impact the technological longevity and wider uptake of the STAR blockchain solution, considering that there will be a need for advancing the TRL level and the overall maturity of the solution following the end of the project. Overall, HLF has a more established community and better documentation than other blockchains (e.g., NEO). This is documented in recent blockchain developers' surveys (e.g., [ChainStack21]) and is illustrated in Figure 3 and Figure 4 below.

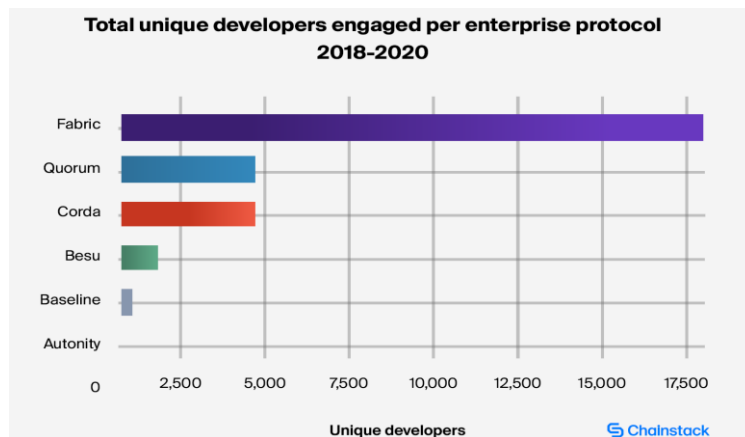


Figure 3: Total Unique Developers Engaging in various Blockchain Infrastructures and Protocols [Chainstack21]

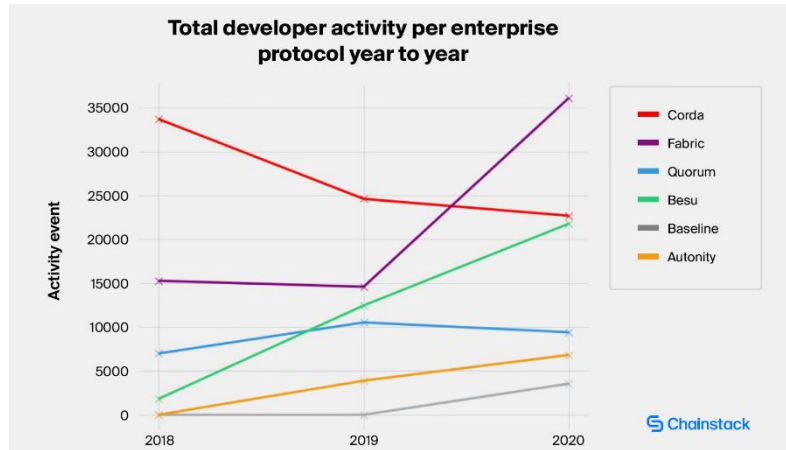


Figure 4: Evolution of Developers' Activity for various blockchain infrastructures [ChainStack21]

Overall, Hyperledger Fabric was prioritized as the blockchain infrastructure to be deployed and use in STAR.

3 STAR Data Reliability Framework

3.1 Framework Placement to STAR Architecture

STAR Architecture, as shown in Figure 5 below and documented in deliverable D2.6, is consisted of three main domains. These domains are the cybersecurity, human robot collaboration and safety domain.

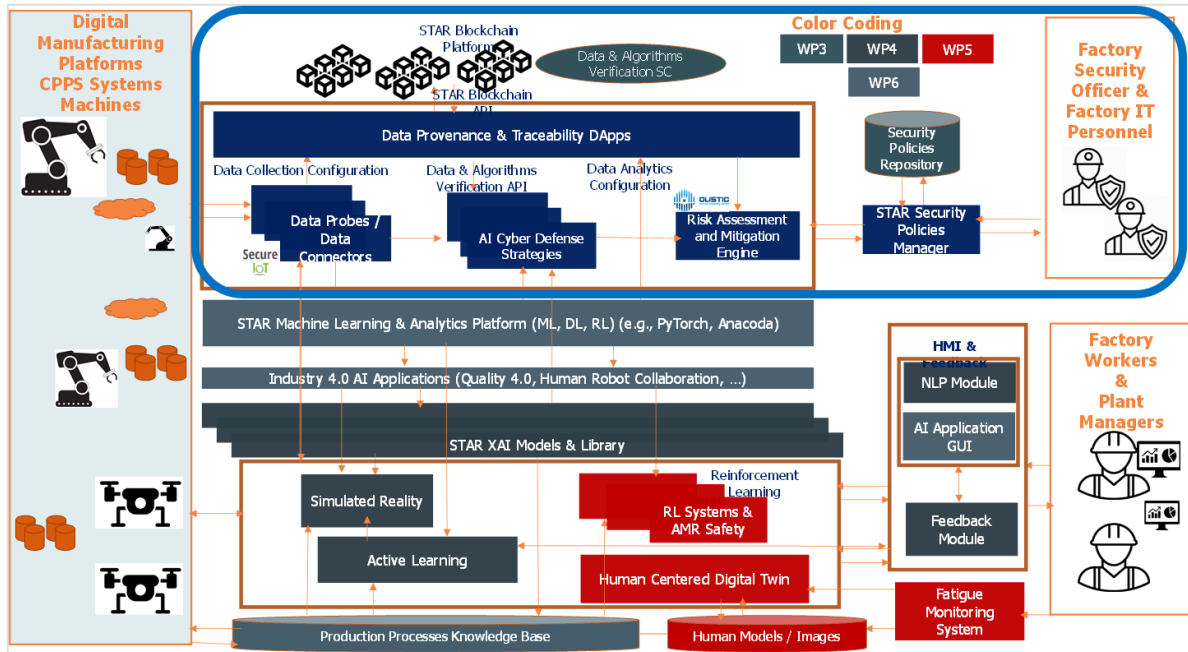


Figure 5 STAR Functional Modules and Logical View of the Architecture [D2.6]

The STAR Distributed Ledger Services for Data Reliability (DLSDR) (depicted in Figure 6 below) are part of the cybersecurity domain which comprises functionalities that are destined to ensure the reliability and security of industrial data, as well as of AI algorithms that are trained and operational based on them.

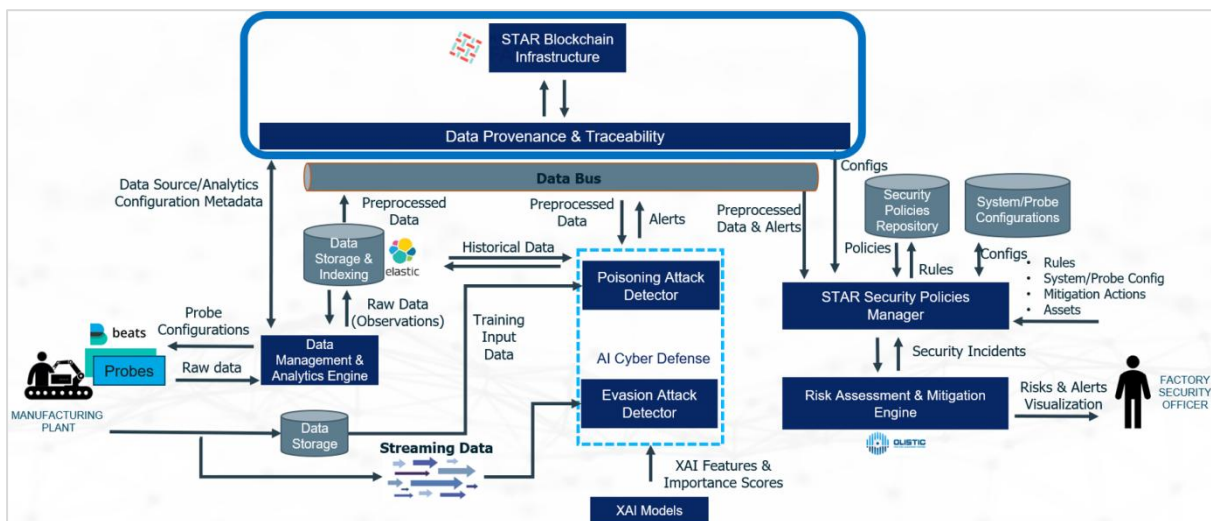


Figure 6 STAR Security and Data Governance for AI Systems in Manufacturing Logical View

DLSDR (Figure 6 above) provides the means for tracking and tracing industrial data for AI algorithms, notably the definitions of the data sources used, the data used to configure STAR AI algorithms and finally the data for persisting their results. To this end, it provides services to the AI algorithms and applications utilizing their results. The DLSDR module is aimed at reinforcing the reliability and the security of the source data used in the STAR system. It records information (i.e., metadata) about the acquired data to facilitate the detection of abuse and tampering attempts against these data. Specifically, data ingested in the DLSDR can be queried by other STAR modules to facilitate the validation of datasets and to ensure that the data that are used have not been tampered.

DLSDR facilitates the communication with other components by exposing appropriate interfaces that enables the AI Algorithms and data consumers to persist and retrieve the data of interest. The core components that interact with the DLSDR are:

- The AI industrial algorithm components (e.g., AI Cyber Defence Strategies) which are using the DLSDR for managing Data Sources, Configurations, and results, and
- The STAR Security Policies Manager which is using DLSDR for AI algorithms results validation.

3.2 DLSDR Framework Architecture

The STAR Distributed Ledger Services for Data Reliability (DLSDR) framework, illustrated in Figure 7, exhibits a rather complex architecture, the assemblage of which requires the use of several interconnected machines each hosting some of its components, thus formulating a private permissioned Blockchain network. The latter is accompanied by additional software applications that operate, in essence, as an interface exposing the network to other STAR platform services in a frictionless and interoperable way.

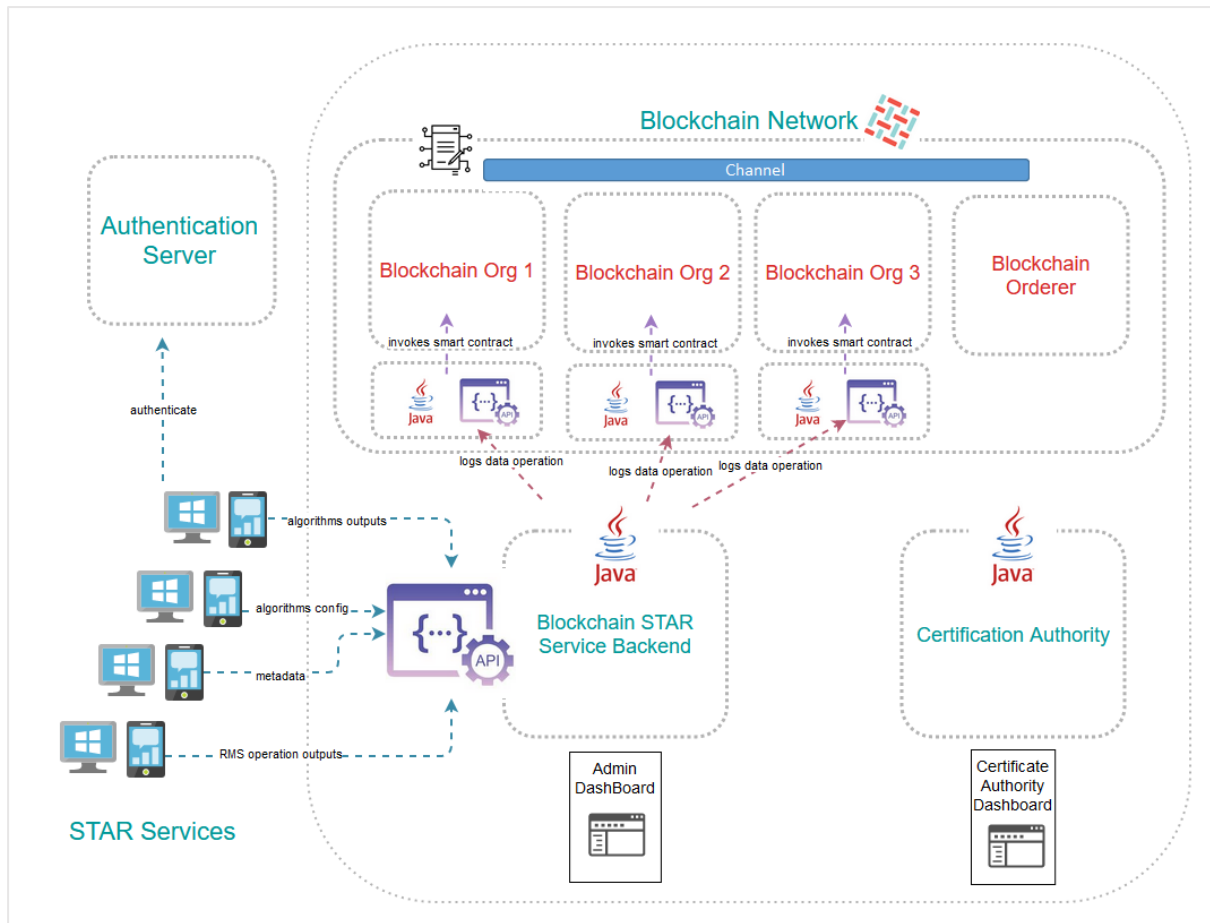


Figure 7: DLSDR Framework Architecture

In the context illustrated above, an Organization refers to a STAR platform industrial shareholder who wants to either persist or retrieve metadata with the ultimate intention of certifying either the provenance of the original data or the authenticity of various assumed configurations and calculation results when those data have been processed by AI algorithms. A Channel is formed between each group of parties who need to share metadata serving both as an avenue of information and an isolation medium from unsolicited readers. One or more Hyperledger Fabric Nodes are maintained by Organizations (representing, for example, different algorithms provided by a particular STAR service or different services provided by a particular STAR shareholder such as the Runtime Monitoring System [RMS]). Nodes can join one or more Channels depending on the data isolation needs of every use case. A Channel is used to install one or more Smart Contracts on, with the latter describing data sources metadata, data processor configurations and data processing results.

Adjacent STAR services will communicate with the Blockchain Data Reliability Service through a multi-level Backend application that will provide numerous APIs to client applications, rather than directly with its middleware and low-level APIs. Another option would be to move those features directly to the platform's adjacent components, which would be more in line with the blockchain decentralization model, but this would have necessitated their developers having substantial experience with dApp development. A final idea is to have Smart Contracts handle everything a back-end process performs, including the role of APIs and authentication, but giving only particular responsibilities to each different component has been deemed to be much cleaner, scalable and controllable.

Moving on, the multiple APIs cater to distinct users; one is for the Authority in charge of creating the certificates that would allow users to use the system. Another API service will be in charge of administrative and monitoring duties. The producer and the recipient of the metadata will be served through the two most significant APIs. All of the above components and dashboards, like all traditional apps, may employ an authentication server, such as Keycloak¹³, or leverage a simpler authentication protocol.

3.3 Framework Services

The STAR infrastructure for decentralized data reliability, will be implemented based on the FAR-EDGE digital automation platform, which comprises a set of ledger services for industrial processes and is implemented on top of the Hyperledger Fabric infrastructure. Through FAR-EDGE, the consortium has access to the cloud deployment of a complete HLF instance, to be used as a lab environment for further development and validation testing. Moreover, FAR-EDGE provides a readily available ledger service that implements a decentralized version of a gateway-level device registry that empowers edge computing operations. STAR will provide the following services that will empower reliable industrial data and trusted algorithms configurations for industrial quality control:

3.3.1 Analytics Engine Configuration (AEC) Service

An **Analytics Engine Configuration (AEC)** Service (concept illustrated in *Figure 8*) that supports Edge Analytics by providing the capability for distributing analytics manifest objects across multiple gateways. Note that an analytics manifest defines how data streams are to be processed by an individual Edge Analytics Engine (EAE) node, using a combination of predefined data processing elements and workflow instructions. Using the Distributed Ledger as the distribution channel, STAR will ensure a truly decentralized but also reliable system, as analytics manifests are "signed, sealed and timestamped" so that no forgery or tampering is possible.

¹³ Keycloak official website: <https://www.keycloak.org/> (accessed April 2022)

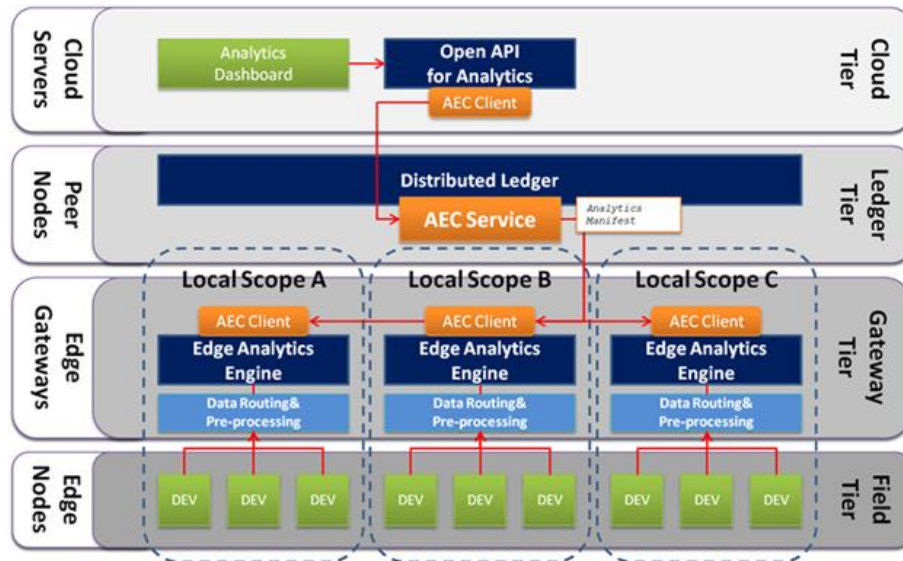


Figure 8: Analytics Engine Configuration (AEC) Service overview

3.3.2 Analytics Results Publishing (ARP) Service

An **Analytics Results Publishing (ARP)** Service (concept illustrated in Figure 9), that makes it possible for edge analytics instances, wherever deployed, to share analytics results on the Distributed Ledger infrastructure, thus contributing to a common data set representing the combined results across the entire distributed system. The virtues of such as workflow lie in immutability and non-repudiation. The ARP service can for example support a machine-as-a-service business model where the OEM is subject to a service level agreement (SLA) as the Distributed Ledger becomes an official registry for performance indicator measurements, where the business-critical information is co-owned by the OEM and the user.

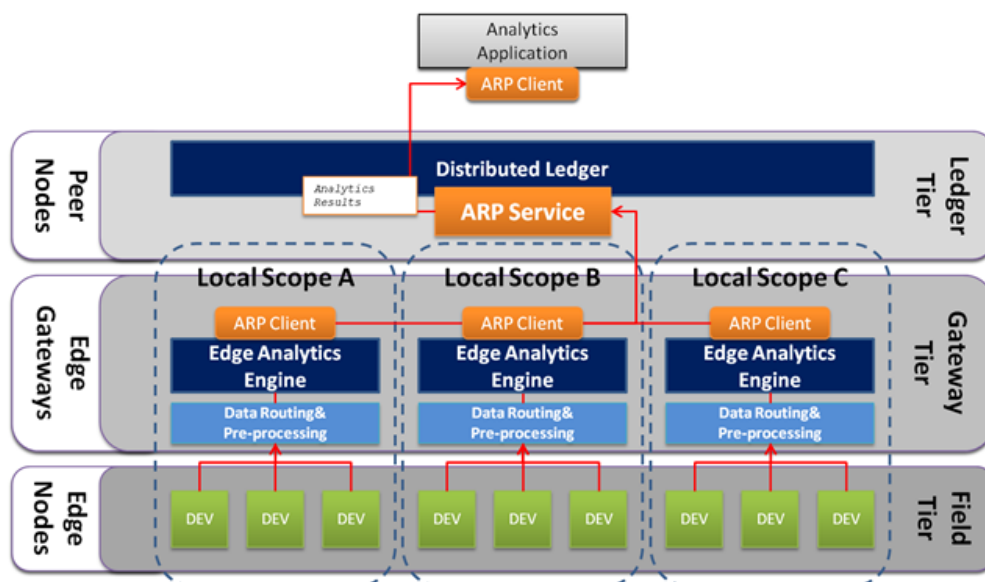


Figure 9: Analytics Results Publishing (ARP) Service overview

3.4 The STAR Hyperledger Fabric Network

3.4.1 Structural Components of the Blockchain Network

A blockchain network is a technical infrastructure that provides ledger and smart contract (which are packaged as part of a “chaincode”) services to applications. Figure 10 illustrates a sample blockchain network infrastructure like the one implemented, deployed and used in the scope of the STAR platform.

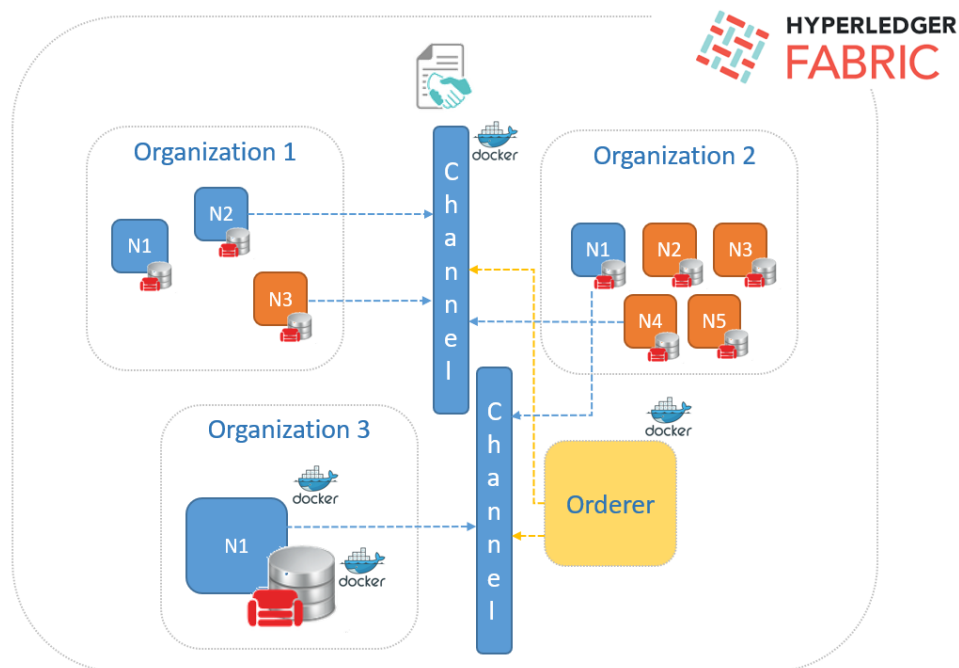


Figure 10: Sample Hyperledger Fabric Network

The latter has been designed so as to conform with the Hyperledger Fabric (HLF) blockchain architectural paradigm and is synthesized by the following components:

- The Distributed Ledger:** In Hyperledger Fabric, a ledger consists of two distinct, though related, parts: a world state and a blockchain. The **world state** is a database that holds **current values** of a set of ledger states. The world state makes it easy for a program to directly access the current value of a state rather than having to calculate it by traversing the entire transaction log (as happens in Bitcoin). Ledger states are, by default, expressed as **key-value** pairs. The **blockchain** is a transaction log that records all the changes that have resulted in the current world state. Transactions are collected inside blocks that are appended to the blockchain – enabling one to understand the history of changes that have resulted in the current world state. The blockchain data structure is very different to the world state because once written, it cannot be modified; it is **immutable** [FabricDocs].
- Nodes (Peers) Deployed Across Various:** The various Nodes belong to different Organizations formulating the STAR Blockchain Network. Each Organization can have one or more Nodes, which are classified in two main types: (i) **Client Nodes** that represent the end user. Those can create transactions and broadcast messages to Orderers (see below) through Channels (coloured blue in the figure above) and (ii) Peer Nodes that receive ordered state updates in the form of transactions from the Orderers, commit transactions and maintain the state of the Ledger [Xu19]. Some of

them can also assume the special role of **endorser** (coloured orange in the figure). As already outlined, HLF is a permissioned network i.e., participating organizations join the blockchain by invitation.

- **Orderers:** A special third type of Node of the network is called Orderer (or “ordering node”) and undertakes the ordering of transactions. In practice, the Orderer is responsible for packaging transactions into Blocks, and distributing them to Anchor Peers across the network. To understand the role of the Orderer in the blockchain, one must review the transaction flow of Fabric, which comprises the following steps: Transaction Proposal, Transaction Packaging and Transaction Validation. Because Fabric’s design relies on **deterministic** consensus algorithms, any block validated by the peer is guaranteed to be final and correct. Ledgers cannot fork the way they do in many other distributed and permissionless blockchain networks.
- **Various Channels:** Channels are destined to support and provide confidential transactions between two or more specific members. They enable the establishment of a private “subnet” between specific nodes of the network. They are similar to topics of a publish/subscribe messaging pattern. Client nodes can join some channels, while not joining others, the existence of which might as well be unknown to them. Clients connected to a channel may broadcast transactions on the channel which are then delivered to all peers within the channel by the orderers.
- **Peer State Databases:** They can store any binary data that is modelled in the Smart Contracts. They support core operations such as getting and setting a key (asset) and querying based on keys. The current options for the peer state database are LevelDB¹⁴ and Apache CouchDB¹⁵, with STAR opting for the latter. As a document object store, CouchDB allows you to store data in JSON format, issue JSON queries against your data, and use indexes to support your queries.
- **Chaincode:** Hyperledger Fabric leverages container technology to host smart contracts (called chaincode) that comprise the application logic of the system. Chaincode can be implemented in programming languages such as Go, JavaScript or Java and is invoked through a transaction proposal. The execution of chaincode is based on the world state stored in the ledger for a channel [Xu19].

3.4.2 Ordering Service

Transactions contain signatures of every Endorsing Peer and are submitted to Ordering service. Transactions are ordered into blocks and are “delivered” from an Ordering service to Peers on a Channel. Peers validate transactions against endorsement policies and enforce the policies. Figure 11 illustrates how Fabric handles a decentralized transaction through the Ordering Service.

The go-to ordering service choice for production networks, the Fabric implementation of the established Raft protocol uses a “leader and follower” model, in which a leader is dynamically elected among the ordering nodes in a channel (this collection of nodes is known as the “consenter set”), and that leader replicates messages to the follower nodes. Because the system can sustain the loss of nodes, including leader nodes, as long as there is a majority of ordering nodes (what’s known as a “quorum”) remaining, Raft is said to be “crash fault tolerant” (CFT). In other words, if there are three nodes in a channel, it can withstand the loss of one node (leaving two remaining). If you have five nodes in a channel,

¹⁴ LevelDB official Github page: <https://github.com/google/leveldb> (accessed April 2022)

¹⁵ Official Apache CouchDB website: <https://couchdb.apache.org/> (accessed April 2022)

you can lose two nodes (leaving three remaining nodes). This feature of a Raft ordering service is a factor in the establishment of a high availability strategy for your ordering service. Additionally, in a production environment, you would want to spread these nodes across data centres and even locations. For example, by putting one node in three different data centres. That way, if a data centre or entire location becomes unavailable, the nodes in the other data centres continue to operate [FabricDocs].

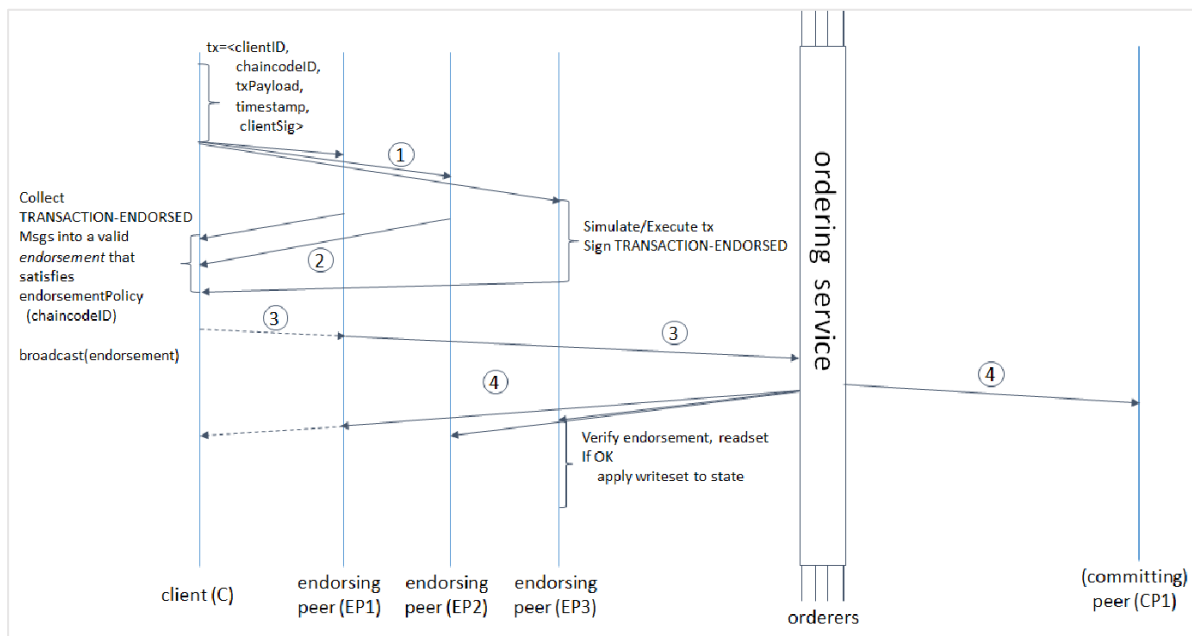


Figure 11: Interfacing to the HLF Ordering Service [FabricDocs]

3.4.3 Mapping Stakeholders to Nodes

To remain faithful to the decentralized architectural paradigm of the blockchain and to remove single points of failure, the distributed ledger network has been envisaged as having multiple nodes hosted in different virtual machines (as opposed to hosting all nodes in a single machine or delegating all operations to a couple of nodes just to take advantage of the blockchain desired properties). However, the exact number of Hyperledger Fabric Organizations and Peer Nodes may vary, depending on the levels of complexity, resilience and performance that are acceptable for the use case at hand. Likewise, organizational deployment limitations and requirements must be considered. In the context of STAR, we opt for a deployment of a minimum of one Peer Node per participating Organization in the circular chain(s) of the STAR pilots. This approach ensures the resilience of the blockchain and enables each stakeholder to withhold control over their data. It is also noteworthy that HLF provides flexibility by permitting the creation of sub-groups of network participants sharing common data and business operations, by leveraging the concept of Channels. This enables the selective sharing of information within subsets of the participating organizations for business and confidentiality reasons. For instance, there are use cases where a sensor belonging to a Manufacturer sends data to a specific AI-enabled service with a third consumer service having the need to validate the sensor data provenance before accepting the AI-enabled service’s output, while every other stakeholder of the platform ought to remain ignorant. For these use cases information exchange can take place through Channels that involve only the participating organizations/platform services rather than the whole set

of peers. Similarly, business logic is shared again only among those participants since HLF chaincode is deployed on said Channels.

Figure 10 illustrates an example of such relationships between Nodes that belong to different Organizations yet share data and business logic through Channels. The upper Channel collects two Nodes from Organization 1 and one from Organization 2, while the lower Channel enables collaboration between the single Node from Organization 3 and another Node from Organization 2.

3.4.4 Smart Contracts vs. Chaincode

Hyperledger Fabric users often use the terms Smart Contract and Chaincode interchangeably. In general, a smart contract defines the transaction logic that controls the lifecycle of a business object contained in the world state. It is then packaged into a chaincode which is then deployed to a blockchain network. Think of smart contracts as governing transactions, whereas chaincode governs how smart contracts are packaged for deployment. A smart contract is defined within a chaincode. Multiple smart contracts can be defined within the same chaincode. When a chaincode is deployed, all smart contracts within it are made available to applications [FabricDocs].

As far as STAR is concerned smart contracts and chaincode will support the three services described in detail in sections 4.2.2, 4.2.3 and 4.2.4: (i) Data Sources Verifiability (ii) Data Processor Configurations Verifiability and (iii) Data Processor Results Verifiability. Chaincode will run in secured Docker¹⁶ containers isolated from the endorsing peer process.

3.4.5 Certificate Authorities

Certificate Authorities (CAs) play a key role in the Fabric network because they dispense X.509 certificates¹⁷ that can be used to identify components as belonging to an organization. Certificates issued by CAs can also be used to sign transactions to indicate that an organization endorses the transaction result – a precondition of it being accepted onto the ledger.

Different components of the blockchain network use certificates to identify themselves to each other as being from a particular organization. That is why there is usually more than one CA supporting a blockchain network – different organizations often use different CAs, as is also the case for the STAR blockchain network. The mapping of certificates to member organizations is achieved via a structure called a Membership Services Provider (MSP), which defines an organization by creating an MSP which is tied to a root CA certificate to identify those components and identities that were created by the root CA.

Certificates issued by CAs are at the heart of the transaction generation and validation process. Specifically, X.509 certificates are used in client application transaction proposals and smart contract transaction responses to digitally sign transactions. Subsequently the network nodes who host copies of the ledger verify that transaction signatures are valid before accepting transactions onto the ledger [FabricDocs].

¹⁶ Docker official website: <https://www.docker.com/> (accessed April 2022)

¹⁷ "X.509: Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks". ITU. Available online: <https://www.itu.int/rec/T-REC-X.509> (accessed April 2022)

3.4.6 Membership Service Providers

Because Fabric is a permissioned network, blockchain participants need a way to prove their identity to the rest of the network in order to transact on the network. Certificate Authorities issue identities by generating a public and private key which forms a key-pair that can be used to prove identity. Because a private key can never be shared publicly, a mechanism is required to enable that proof which is where the MSP comes in. For example, a peer uses its private key to digitally sign, or endorse, a transaction. The MSP on the ordering service contains the peer's public key which is then used to verify that the signature attached to the transaction is valid. The private key is used to produce a signature on a transaction that only the corresponding public key, that is part of an MSP, can match. Thus, the MSP is the mechanism that allows that identity to be trusted and recognized by the rest of the network without ever revealing the member's private key.

The MSP is the mechanism that enables you to join an existing permissioned blockchain network. To transact on a Fabric network a member needs to:

1. Have an identity issued by a CA that is trusted by the network.
2. Become a member of an Organization that is recognized and approved by the network members. The MSP is how the identity is linked to the membership of an Organization. Membership is achieved by adding the member's public key (also known as certificate, signing cert, or signcert) to the Organization's MSP.
3. Add the MSP to a channel.
4. Ensure the MSP is included in the policy definitions on the network.

MSPs occur in two domains in a blockchain network: locally on an actor's node and in channel configuration. The key difference between local and channel MSPs is not how they function – both turn identities into roles – but their scope. Each MSP lists roles and permissions at a particular level of administration [FabricDocs].

3.4.7 Scalability Considerations

An aspect that is strictly related to the consensus algorithm employed by the blockchain protocol is that of scalability. The scalability of any decentralized system will depend directly on the mechanism to replicate the state of each machine (the consensus algorithm). In Figure 12, we can examine how the performance is affected by the increase in the number of nodes in the network.

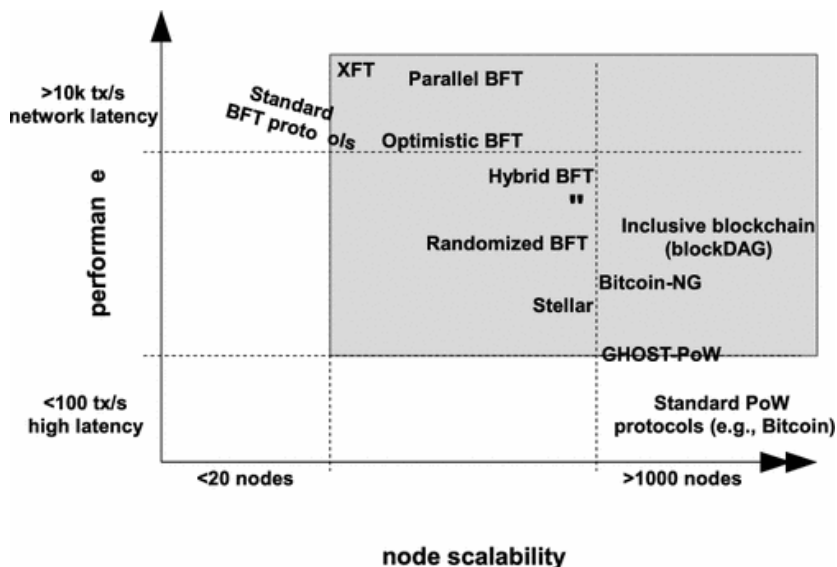


Figure 12: Illustration of Performance and Scalability of Different Families of PoW and BFT protocols [Vukolić15]

In the case of STAR, using democratic consensus algorithms like CFT or BFT with Hyperledger Fabric, will provide good performance while allowing reasonable scalability. By reasonable we mean being able to scale up to a hundred or couple hundred nodes. Such assumption is realistic as the platform is envisaged to be used by multiple industrial sector participants but not thousands, making it suitable for CFT or BFT-like consensus algorithms.

3.5 Algorithms Configuration Parameters Validation

An important step in ensuring data reliability for sensitive configuration files is the ability to monitor and quantify changes in parameter values, but also in the file structure, so as to have an additional safety valve against adversarial tampering with those files, as well as easier traceability in case of malfunctions induced by inappropriate configuration. Our ambition is to add such a validation layer as part of the Data Provenance and Reliability component of the STAR WP3 architecture that will monitor for suspicious changes to the configurations of the AI algorithms used in the STAR platform. These configuration files are expected to contain various kinds of information about the algorithm hyperparameters, its structure and information about the problem addressed such as:

- Type of classification (Multiclass/Binary)
- Input dataset
- High-level hyper-parameters such as:
 - number of epochs
 - batch size
 - input size
 - loss function
 - optimization method
- Model structure
 - layer types and their parameters (e.g., in case of a CNN)

The above can have different subfields for specific hyperparameters (e.g., learning rate for the optimization method, margin for a contrastive loss function), thus giving the file a hierarchical structure that can easily be represented by popular configuration file formats

such JSON, XML, HOCON etc. The Table 4 below shows the complete configuration information for a CNN classifier used in the automatic defect inspection of the Decocap dataset in PCL UC2.

Table 4 CNN configuration information classifier example

Type of classification	Multiclass		
Dataset	Decocap		
Num Of Epoch	15		
Batch Size	30		
Input Layer shape	(400,400,1)		
Hidden Layers	CNN	units	8
		Kernel size	(3,3)
		padding	same
		activation function	relu
		input shape	(400,400,1)
	Maxpooling	pool_size	(2,2)
	Dropout	0,25	
	CNN	units	8
		Kernel size	(3,3)
		padding	same
		activation function	relu
	CNN	units	8
		Kernel size	(3,3)
		activation function	relu
	Maxpooling	pool_size	(2,2)
Dropout	0,25		
Flatten			
Dense	units	128	
	Activation function	relu	
Dropout	0,25		
OutputLayer	Dense	units	3
		Activation function	softmax
Loss	categorical_crossentropy		
Optimizer	adam		
Adversarial Algorithm	Multiclass		
	Decocap		
	Epoch	25	
	Batch	100	

Of course, comparing two different configuration files with the above specification might seem easy at first, but is in fact quite challenging due to their semi-structured format, which could lead to resulting configuration files with very different structures (e.g., parameters about a very deep neural network using transfer learning vs. a shallower, custom one

tailored to a specific problem). For this reason, we deem it necessary to adopt an intelligent method of comparing these files taking advantage of recent research in areas such as document databases, auto-ML, graph theory and graph convolutional neural networks (GCNNs).

3.5.1 Proposed Approach

We plan to tackle this problem in two ways. The first is to find an appropriate method of measuring the distance between two configuration files in a semi-structured, hierarchical format such as JSON. This is aimed at covering configuration and parameter files of a general kind. In the second part of our approach, we will focus more specifically on convolutional neural networks, which are used throughout the project in different scenarios and include vision-based tasks (e.g., defect classification, human pose detection). The rationale behind this choice is that modern CNNs very often have complicated architectures with parallel paths and skip connections which might be hard to model in JSON. Thus, more traditional edit distances might not accurately depict topological differences between two networks, even though these differences might have a large impact on the result. To counter this, we propose reconstructing the neural network from the parameter file and representing it as a graph, which we hope to process with a more sophisticated method, such as the Graph Edit Distance (GED) or using Graph Neural Networks (GNNs). In the diagrams below we outline the flows of the two approaches.

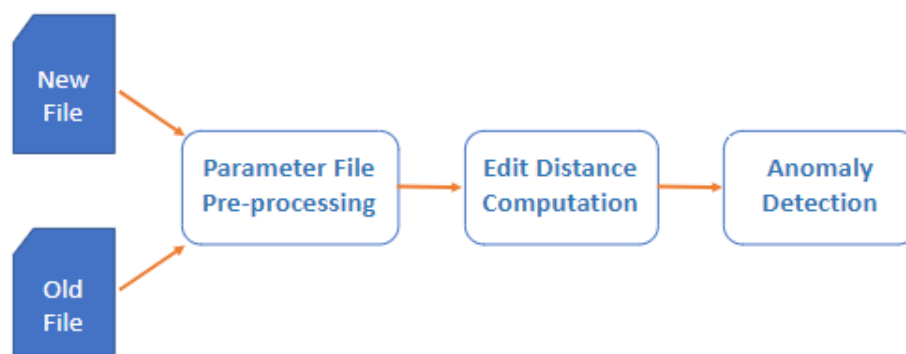


Figure 13 Flow for General Edit Distance Computation

The first diagram showcases the more general flow for parameter files of a semi-structured and hierarchical format. The aim is to quantify the differences between a new updated parameter file coming into the Data Provenance and Reliability module with the existing parameter file stored in the blockchain. The two files go through the following pipeline:

- **Parameter File Pre-processing:** This sub-module will be responsible for extracting the structure and parameter values from the inputs and bringing them to a suitable format for the edit distance computation. Depending on the method chosen for the next step of the pipeline, this step might include building a tree structure or additionally separating the CNN specific parameters to build a graph representation of the network.
- **Edit Distance Computation:** For general parameter file arguments this can be a Tree Edit Distance (see the respective section for more details) and could also include a graph edit distance or a siamese GCNN model in case the CNN architecture parameters are chosen to be treated separately. This is the core module of the pipeline that will output a distance measure between the two files.

- **Anomaly Detection:** This part is the ultimate aim of the pipeline which is to flag the new parameter file in case the distance computation proves suspicious. It can be rule based or timeseries based, taking into account a history of distance measurements.



Figure 14 Training Flow for AI-based CNN-specific Edit Distance Computation

The next flow will be studied because there is a likelihood that traditional Tree and Graph Edit Distances could prove insufficient for quantifying the difference between CNN architectures, especially regarding the correspondence between the change in layer parameters and topology and the change in task performance. For this reason, an attempt will be made to compare it with a more sophisticated AI-based method, leveraging recent advances in Graph Neural Networks. The pipeline for training such a model will consist of the following steps:

- **CNN Architectures Generation:** This step's aim will be to generate a synthetic dataset of Convolutional Neural Networks with different topologies and parameters leveraging developments in AutoML where a similar process takes place for Neural Architecture Search (NAS). Using the parameter-file schema as input might be a good starting to guide the generation and tailor it to the problem at hand.
- **Siamese GCNN:** The last step will be the training of a GNN that learns the distance between graph embeddings coming from pairs of graphs and once trained, can be used as a predictor in the distance computation step of the first diagram.

In terms of our roadmap, the aim is to implement both the Edit Distance and the AI-based methods and compare which of them performs the difference quantification task the best or how they can be both combined into a more effective whole. The next sections will include methods from the scientific literature that can be used to implement the steps described in the two previous diagrams.

3.5.2 Quantifying the Difference between Parameter Files

In the field of database systems there is a large literature on the search and retrieval of hierarchical semi-structured formats such as JSON and XML. A plethora of similarity queries and their related distance measures have been studied especially for the older and simpler XML format [Cobena02] [Finis13]. Most approaches for XML distance calculation are based on the concept of the Tree Edit Distance (TED) [Pawlik16], which is measuring the steps it takes to transform one document to another and by these means respecting their hierarchical structure and parameter values.

```

{
  "title" : "Star Wars -
            A New Hope",
  "running time" : 125,
  "cast" : {
    "Han" : "Ford",
    "Leia" : "Fisher"
  }
}
(a)
    
```

```

{
  "cast" : [
    "Ford",
    "Fisher"
  ],
  "running time" : 125,
  "name" : "Star Wars -
            A New Hope",
}
(b)
    
```

Figure 15 JSON Diff – Same contents but distance calculation is not straightforward [Huetter22]

JSON, which is nowadays more popular, differs from similar data formats in that it supports both ordered and unordered sibling nodes. This needs to be handled both in an appropriate tree representation as well as in the distance function that assesses how similar the representations are. According to [Huetter19] computing the difference between two JSON document is NP-hard, if no restrictions are imposed on the number of node edit operation operations available (insertion, deletion, value modification). Additionally, there is limited support for JSON similarity queries in many existing systems. For example, several stand-alone tools such as [Grossbart21] [Circlecell22] compare documents line by line thereby ignoring hierarchical information. Most database systems also limit their approaches to basic parameter types such as numbers, strings or sets avoiding to compute the distance between full documents [MongoDB20] [PostgreSQL22]. However, there have been recent approaches such as JEDI [Huetter22], which introduces a lossless tree representation for JSON documents on which the edit-based distance can be more efficiently computed. Their evaluation showed significant scalability supporting millions of documents with trees up to tens of thousands of nodes.

In order to understand different approaches and their limitations as well as obtain a better understanding of the problem setting, in the next section we delve deeper into the Tree and Graph Edit distance computations and also examine Siamese Graph Convolutional Neural Networks (GCNNs) as an AI based alternative.

3.5.2.1 Tree Edit Distance

A traditional way of measuring similarity between data represented as a string of characters is the Minimum Edit Distance, which is defined as the minimum number of character insertions, deletions or substitutions to apply to one string so that it becomes identical to another. Despite this measure being extensively researched and utilized in many areas (e.g., genome sequences comparison), its extension to hierarchical data structures is not straightforward. To achieve such an extension the Tree Edit Distance (TED) has been proposed by [Tai79], based on the similar idea of the minimum number of modifications for turning one tree into another. The measure has since been applied to a diverse set of applications ranging from software engineering to natural language processing and bioinformatics.

The typical operations used for TED are node deletion, node insertion and label renaming. These operations can receive different weights according to their context dependent importance and the final distance is generalized as the total cost of modification operations [Tai79] [Zhang89]. Most TED algorithms work by decomposing the input trees into sub-

forests and using dynamic programming to calculate the result from the bottom up. The two mainstream solutions are proposed by Zhang and Shasha [Zhang89] and Chen [Chen01]. The recursive solution also known as Zhang decomposition focuses on deleting the left-most or right-most node to create new sub-forests, where the choice between the two majorly influences the runtime efficiency of the algorithm. Currently the state-of-the-art time complexity for ordered trees is cubic, while for unordered trees the problem is NP-complete.

Recent approaches have focused on different strategies for node deletion based on the Zhang decomposition. While the initial implementation was $O(n^4)$ in time and $O(n^2)$ in space, this has been improved, first by Klein [Klein98] with $O(n^3 \log n)$ time and $O(n^2)$ space complexity. The best asymptotic bound so far has been established by Demaine et al. and also achieved in the RTED[Pawlik15] and AP-TED+ [Pawlik16] algorithms, namely $O(n^3)$ time and $O(n^2 \log n)$ space. The study in [Bringmann17] indicated that this might also be the absolute lower bound. An alternative solution that is most efficient for deep trees with small number of leaves is Chen's algorithm [Chen01], which has shown superior results for both "thin" and "zig-zag" trees in [Schwarz17]

3.5.2.2 Graph Edit Distance

Graph similarity measures can be defined in different ways; the most straightforward is as a mapping $f: G \times G \rightarrow \mathbb{R}$ which computes the similarity measure directly in the graph space. In the case of this mapping also being a graph kernel, dimensionality reduction techniques or Support Vector Machines can be utilized in the distance computation [Neuhaus09] [Gauzere12]. A different avenue is using vector matching techniques over a graph embedding space where a graph embedding is defined as $g: G \rightarrow \mathbb{R}^d$ [Bunke11]. However, we will focus on the computation of a Graph Edit Distance (GED) similar to the Tree Edit distance above operating directly on the graph space.

GED is defined as the minimum total cost of an edit path between two graphs. In this case there are six available operations, name insertion, deletion and substitution both for nodes and for edges. Unfortunately, as with the TED, the computation is expensive and GED is classified as a NP-hard problem [Zheng09]. On the other hand, it is very sensitive to small changes in the graph structure [Stauffer17], something that is crucial for neural network architecture comparison, where small changes can have a significant effect on performance.

Most exact algorithms in the literature [Abu18] [Marsico15] fail to scale well in practice, although they can still be useful for smaller networks. Therefore, the focus has shifted to heuristics trying to discover upper and lower bounds. The basis of these heuristics can be various optimization techniques such as local search [Riesen15], linear programming [Lerouge16] [Lerouge17], the linear sum [Bougleux18] and quadratic assignment [Bougleux17] problems.

3.5.2.3 Siamese Graph Convolutional Neural Networks (S-GCNs)

Graph Neural Networks (GNNs) are neural network models with the ambition of capturing and exploiting graph features through message passing via connected nodes in the graph. This way useful information can be transferred between a node and its neighbours enabling the creation of discriminative graph embeddings. Different variants of GNNs have been developed such as Graph Convolutional Neural Networks (GCNNs) (DeepWalk [Perozzi14], node2vec [Grover16]), Graph Attention Networks (GATs) [Velickovic18] and the older Graph

Recurrent Networks (GRNs) [Frasconi98] all of which have achieved notable performance in various graph learning tasks [Zhou18].

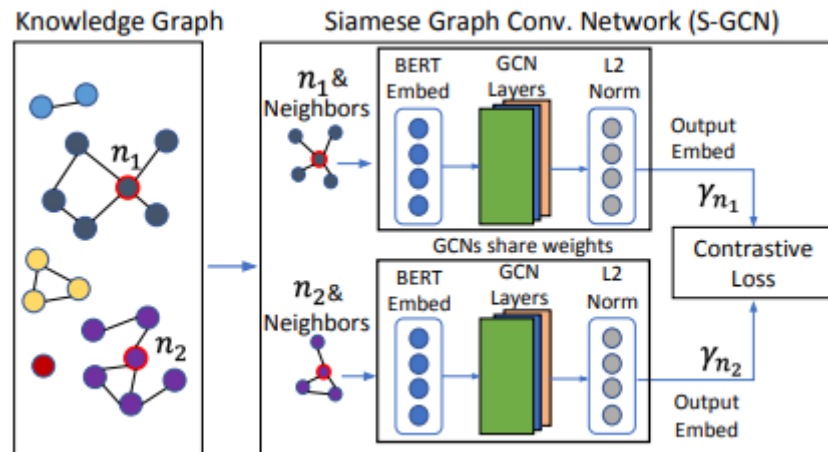


Figure 16 S-GCNN architecture from [Kirosheev21]

Our focus in this section will be on GCNNs and more specifically Siamese GCNNs [Kirosheev21]. In graphs convolutional and pooling layers work similarly to images by combining information from adjacent nodes and then reducing it through pooling. Siamese networks use two parallel GCNN trunks (potentially with shared weights) leading to a distance calculation layer. Distance calculation can work in different ways depending on the application. For instance, in entity matching [Kirosheev21] the network is optimized to produce small distance for nodes of the same entity and large ones for nodes that are unrelated. The distance learned by the siamese network can be more robust to typos or abbreviations. Graph matching is another application, very similar to our envisioned task. In [Ktena17] and [Ma18] the authors use S-GCNNs to match functional brain networks extracted from MRI images. Finally, S-GCNNs have also been successfully leveraged for matching long text documents, by representing the document as a graph between keywords connected through their interactions.

3.5.2.4 Simulating Training Data

As all deep learning applications, GCNN models require large amounts of data, which in our parameter file use-case are hard to obtain. However, since the parameters describe CNN models, which are themselves programmatically generated; it might be good enough for our purposes to create a synthetic dataset by artificially generating CNN architectures. The first type of solutions originates from Graph Generative Models, while alternative solutions, especially when guided generation is needed, are those used in Neural Architecture Search (NAS) to search for the optimal architecture of a neural network for a given dataset, without manual design or intervention.

One of the first generative models, NetGAN [Shchur18] had used random walks to generate graphs by feeding random walks from an input graph to a GAN architecture. More sophisticated approaches such as GraphRNN [You18] utilize Recurrent Neural Networks to generate the graph's adjacency matrix node by node, while others use a sequential decision-making process either based on graph embeddings [Li18] or auto-regressive models [Shi20].

Some approaches also use reinforcement learning to encourage certain properties in generating graphs, such as MolGAN [DeCao18], which tries to produce realistic molecule structures with specific chemical properties. Finally, other works rely on GNN auto-encoders by creating varying graphs from latent representations [Grover19], and by constraining them to ensure semantic validity [Ma18a].

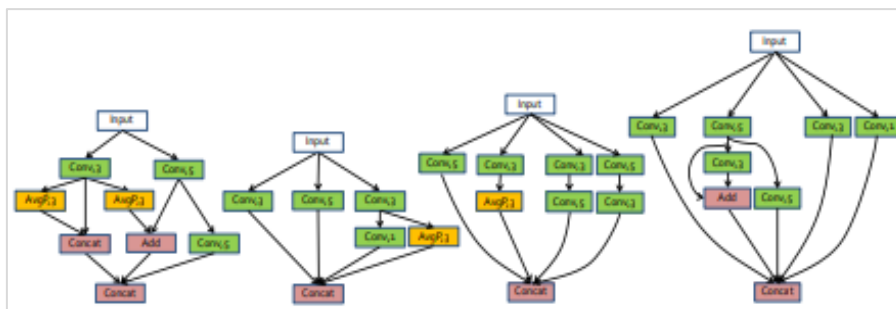


Figure 17 Network architectures generated by variants of Block-QNN [Zhong18]

While Graph Generative Models can produce fairly good results for small graphs and also constrain their output to possess certain properties, they usually require substantial input data, which should either be ready at hand or generated via another method from scratch. This is where AutoML methods can be of value. Neural Network generation methods such as NAS [Baker17] and MetaQNN [Zoph17] rely on reinforcement learning to guide network generation in order to fulfill some performance related objective (e.g., accuracy or runtime). An advantage of these methods is that they are designed specifically for neural network generation and operate on a preset set of node types together with specific hyperparameters making them more applicable to our use case. Of special interest is also a technique called BlockQNN [Zhong18] with uses CNN blocks as the basis for generation.

4 Industrial Data Reliability Framework Specs

4.1 Distributed Ledger Node Management

4.1.1 Registration and Discoverability of the Platform Nodes

Owing to the Blockchain component’s decentralized architecture, with each Organization hosting their proper Nodes in distinct virtual machines, the necessity to make those discoverable by the individual platform services has arisen. A second concern lies in the fact that the various STAR service developers ought to interact with the STAR Blockchain Service as a black-box monolithic system, without having to adjust their design patterns to accommodate different network configurations for the various Hyperledger Fabric protocol participants. To this end, invocations on Smart Contract (chaincode) functions by end-users are carried out through a sequence of HTTP calls, transmitted via a chain of three consecutively exposed RESTful APIs. The architectural subtleties of this configuration are depicted in Figure 7.

Let us assume that a user of a STAR service uses Keycloak to authenticate and thus receives an a JWT access token, also bearing their unique identifier as an additional custom field. Metadata describing a data operation accomplished as part of the normal functionality of the service are expected to be recorded on the blockchain, using, in particular, a Node corresponding to the user, hosted by their Organization. It would have been cumbersome for the service to memorize all the Node ownership information and act each time accordingly. Therefore, it sends an HTTP call to an API exposed by a common Blockchain Service Backend, containing both the data operation to be recorded and the user’s unique identifier. Said Backend hosts a database serving as a Node Registry, thus enabling discoverability of Node ownership relations. Once the Organization and particular Node of the user is discovered, a new HTTP call containing information on the data operation to be recorded is redirected to a second API which, this time, serves the business logic from the viewpoint of the particular Organization. Note that the machines hosting Nodes for the Organization can be hosted separately than the Blockchain Service Backend - which will be hosted under the STAR domain - even withing the Organization’s premises or somewhere on the Cloud.

The Blockchain Node Registry service has been devised to serve as a discoverability service; it employs a simple database with a single table/document storing tuples containing the following information:

User Identifier	Organization Identifier	Org API Virtual Address	Node in Org Identifier
-----------------	-------------------------	-------------------------	------------------------

New records of this format are being added via a RESTful API each time a new Node is manually created by the network administrator. For example:

```
POST /node_registry/ HTTP/1.1
Host: star-mvp.intrasoft-intl.com:80
Content-Type: application/json
Content-Length: 182 {
  "userID": "alice",
  "organizationID": "org1",
  "organizationAddress": { "host": "195.154.51.1", "port": "8080" },
  "nodeID": "org1.node0"
```

4.1.2 Data Model

In this section we describe the Registration and Discoverability services (RD) data model. The core entity of the RD service is the "NodeIdentifier" which is depicted in Figure 18. Every instance of the "NodeIdentifier" entity can provide a mapping of the blockchain node/deployment with an Organization, department and/or user. The XSD schema of the entity is provided in Table 5 of Appendix A below.

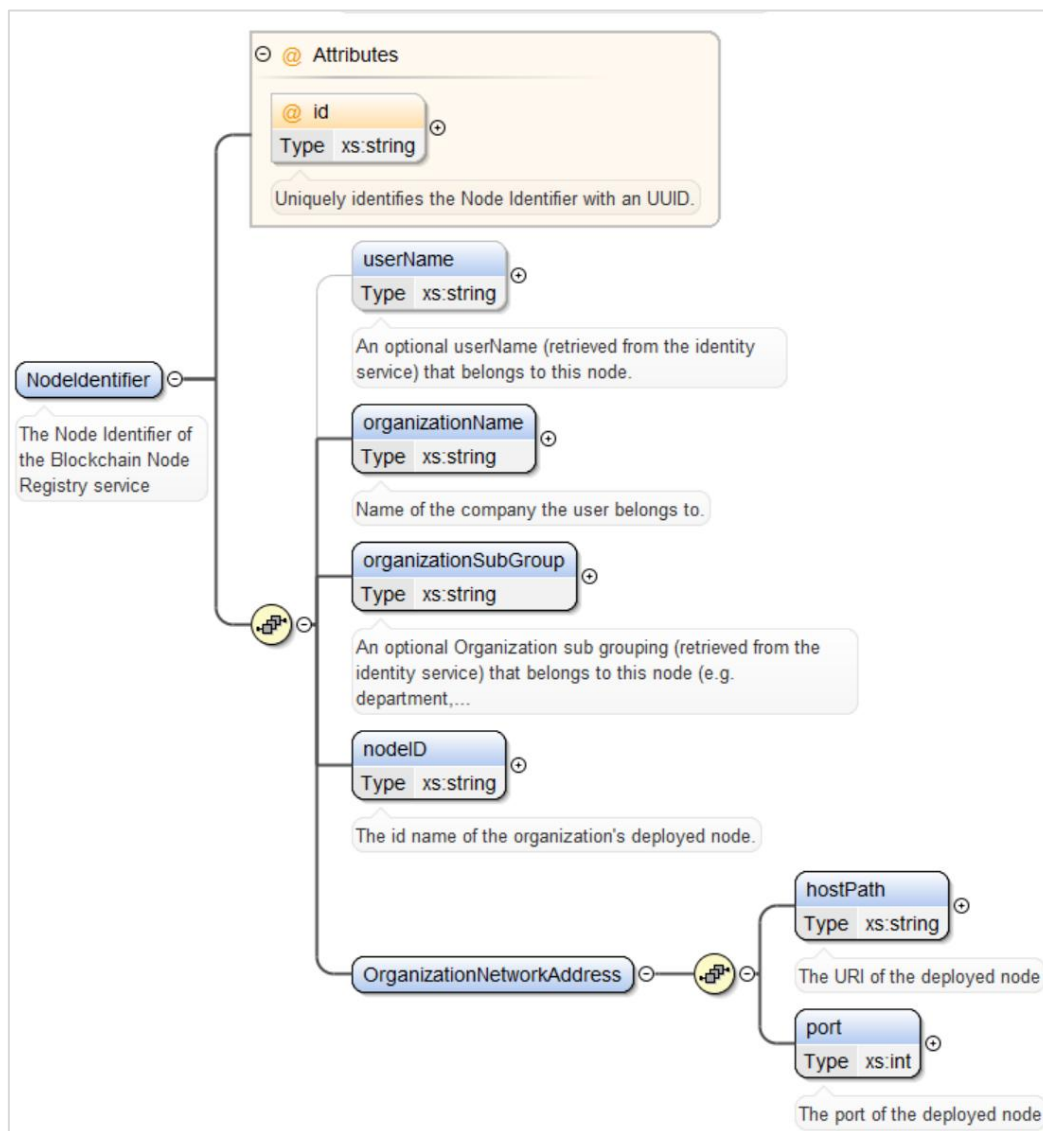


Figure 18: Organization Node Identifier entity graph

As depicted in Figure 18 above the Organization Node Identifier entity has:

- **Id**: Uniquely identifies the Node Identifier with an UUID (Universally Unique Identifier).
- **userName**: An optional userName (retrieved from the identity service) that belongs to this node.
- **organizationName**: Name of the company the user belongs to.

- **organizationSubGroup:** An optional Organization sub grouping (retrieved from the identity service) that belongs to this node (e.g., department, sector, branch, ...).
- **nodeID:** The id name of the organization's deployed node.
- **OrganizationNetworkAddress:** The Organization Network Address consisted of:
 - **hostPath:** The URI of the deployed node
 - **port:** The port of the deployed node

4.1.3 API Specification

Table 5 provides an overview of the Registration and Discoverability services API specification overview which is exposed from the blockchain network as an abstraction of its functionality.

Table 5: Registration and Discoverability services API specification overview

HTTP Method	Service	Input	Output	Functional Description
POST	/node_registry	NodeIdentifier: NodeIdentifier	NodeIdentifier	Used to create a new Node Identifier definition. It returns the registered Node Identifier with an id assigned to it.
PUT	/node_registry /:id	nodeIdentifyerID : String, agreementDefinit ion: NodeIdentifyer	NodeIdentifier	Used to update an existing Node Identifier with the given ID. It returns the new registered Node Identifier with an id assigned to it.
DELETE	/node_registry /:id	nodeIdentifyerID : String	No	Used to delete the specific Node Identifier record.
GET	/node_registry /:id	nodeIdentifyerID : String	NodeIdentifier	Used to retrieve the definition of the Node Identifier with the given ID.
POST	/node_registry /search	NodeIdentifier	<List> NodeIdentifier	Used to discover available Node Identifier instances using a list of their attributes as a search criteria provided in an Node Identifier instance.

4.2 Data Provenance & Traceability Services

4.2.1 Introduction to the Data Entities recorded on the Ledger

Using the STAR Distributed Ledger services three data groups are proposed to be persisted:

1. **Data Source:** Metadata that describe the type of data that are being streamed across the STAR platform along with details on their provenance.
2. **Analytic Algorithm (Processor):** Metadata capable of describing an algorithm type along with its various instantiation configurations across time.
3. **Algorithm Results:** Annotated AI algorithm results whose source can be traced.

The design of the data models for these three data groups has been based on the digital modelling approach of the H2020 FAR-EDGE and PROPHECY projects, which provides the means for representing and configuring streaming data sources in industrial environments¹⁸. The data models have been customized to the requirements for the modelling of STAR data observations. In STAR, the FAR-EDGE/PROPHECY schemas (e.g., XML schema elements like DSD, DI and DK) are used to represent STAR AI algorithm related data. In the sections below we elaborate on the structure of the entities that compose the above-mentioned data groups.

4.2.2 Data Sources Traceability

4.2.2.1 Data Definitions

Data Definition elements are used to define the different data related models. These models are modelling data definitions, interfaces, and processors. These models are analysed in this section.

4.2.2.1.1 Data Kind (DK)

This entity specifies the semantics of the data of the data source, which provides flexibility in modelling different types of data. It can be used to define virtually any type of data in an open and extensible way.

The "DK" has:

- **id:** A required ID which uniquely identifies a DataKind within a STAR deployment
- **name:** An optional human-readable name which uniquely identifies the DataKind
- **description:** Provides an optional description of the DataKind
- **modelType:** Specifies the model type of the Data (i.e. SenML, OM, ...)
- **format:** Specifies the format of the Data (i.e. JSON, XML,...)
- **quantityKind:** A QuantityKind is an abstract classifier that represents the concept of "kind of quantity". A QuantityKind represents the essence of a quantity without any numerical value or unit. (e.g. A sensor -sensor1- measures temperature: sensor1 has quantityKind temperature).
- **AdditionalInformation** (unlimited list): which provides an optional unlimited auxiliary field that may contain any additional information and it is used for further extensions. More specifically it serves as the root of the type definition hierarchy for

¹⁸ <https://github.com/far-edge/digital-models>

any schema and it has the unique characteristic that it can function as a complex or a simple type definition, according to context.

4.2.2.1.2 Data Interface (DI)

The DI entity is associated with a data source and provides the information needed to connect to it and access its data, including details like network protocol, port, network address and more.

The “DI” has:

- **id**: an ID which uniquely identifies a Data Interface within a STAR deployment
- **name**: A human-readable name which uniquely identifies the DataInterface
- **communicationProtocol**: Specifies the protocol Type (i.e. MQTT, JMS, OPCUA, HTTP ...).
- **parameters**: lists the required parameters to set up a specific communication interface. This entity includes:
 - **name**: A human-readable name which uniquely identifies the property for a specific DataInterface.
 - **description**: Provides an optional description of the property
 - **dataType**: A classification of data (i.e. short, int, float, boolean, ...)
 - **defaultValue**: An optional predefined default value for a property.
- **AdditionalInformation** (unlimited list): which provides an optional unlimited auxiliary field that may contain any additional information and it is used for further extensions. More specifically it serves as the root of the type definition hierarchy for any schema and it has the unique characteristic that it can function as a complex or a simple type definition, according to context.

4.2.2.1.3 Data Source Definition (DSD)

This entity defines the properties of a data source in the shop floor, such as a data stream from a sensor or an automation device.

The “DSD” has:

- **id**: A required ID which uniquely identifies a Data Source Definition within a STAR deployment.
- **name**: An optional human-readable name which uniquely identifies the Data Source Definition.
- **description**: Provides an optional description of the Data Source.
- **DataInterfaceReferenceID**: Points to an existing STAR Data Interface definition
- **DataKindReferenceIDs**: contains a list of Data Kind IDs, which references to an existing STAR Data Kind definition.
- **AdditionalInformation** (unlimited list): which provides an optional unlimited auxiliary field that may contain any additional information and it is used for further extensions. More specifically it serves as the root of the type definition hierarchy for any schema and it has the unique characteristic that it can function as a complex or a simple type definition, according to context.

4.2.2.2 Manifests

The Manifest elements are reusable data models capable of manipulating STAR data. These models are the instantiation of the models specified in the Data Definition above and can be used globally (i.e., PdM) or locally (i.e., CPS).

4.2.2.2.1 Data Source Manifest (DSM)

The DSM entity specifies a specific instance of a data source in-line with its DSD, DI and DK specifications. Multiple manifests (i.e., DSMs) are therefore used to represent the data sources that are available in the factory in the scope of the STAR platform.

The “DSM” has:

- **id:** A required ID which uniquely identifies the Data Source Manifest.
- **name:** A human-readable name which uniquely identifies the Data Source Manifest.
- **DataSourceDefinitionReferenceID:** Points to an existing Data Source Definition which is instantiated by adding the parameters below from this DSM.
- **DataSourceDefinitionInterfaceParameters:** contains the data Source Definition Interface configuration parameters in a key-value pair form. The Key is specified from the DI parameters referenced by the DSD which is referenced by the DSM.

4.2.2.3 API Definitions

The main API primitives that will be supported by the STAR Distributed Ledger Services for Data Reliability (DLSDR) for the Data Sources and will enable their persistence and retrieval are listed in Table 6 below.

Table 6 DLSDR for Data Source metadata API specification overview

HTTP Method	Service	Input	Output	Functional Description
POST	/data_source/ dsm	dsmDefinition: DSM	DSM	Used to create a new Data Source Manifest record to the Blockchain network. It returns the DSM instance with an assigned ID.
PUT	/data_source /:id/dsm	dsmID: String, dsmDefinition: DSM	DSM	Used to virtually update an existing DSM. The service generates a new DSM and flags as deleted the id of the provided one. It returns the new registered DSM with a new id assigned to it.
DELETE	/data_source	dsmID: String	No	Used to flag the

	/:id			specific DSM as deleted.
GET	/data_source/:id	dsmID: String	DSM	Used to retrieve an existing DSM Instance.
POST	/data_source/search	dsmAttributes: DSM	<List> DSM	Used to discover available DSM instances using attributes of a DSM as search criteria.

4.2.2.4 Data Sources persistence interaction

Figure 19 below provides the high-level sequence of interactions for a user/client to persist a data source to the distributed ledger network and model/registry repositories of STAR solution.

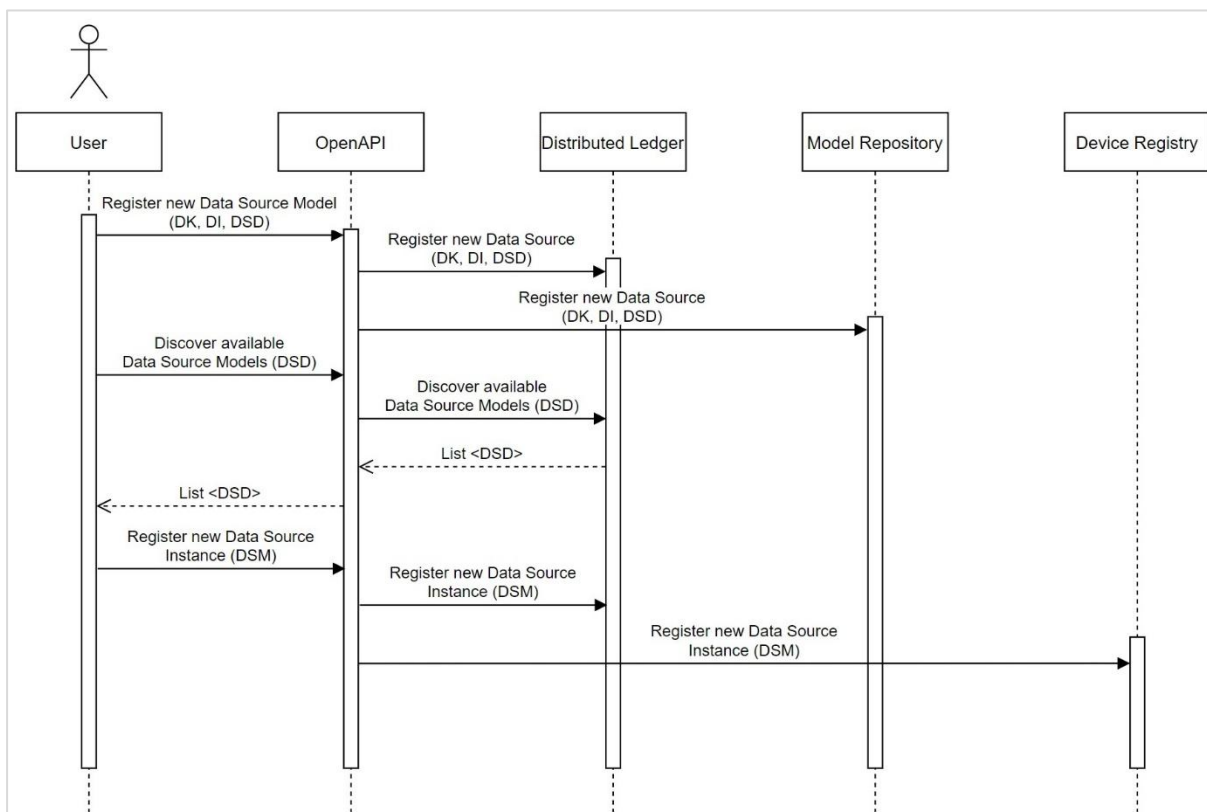


Figure 19 Data Source persistence to Distributed Ledger network

As we can see in Figure 19 above the data sources (DK,DI,DSD & DSM) are persisted in parallel to the Distributed Ledger network and the conventional repositories. This is because the Distributed Ledger is used auxiliary to the existent repositories and persistence mechanisms for provenance and validation of the specified data sources.

4.2.3 Processors Configuration Traceability

4.2.3.1 Data Definitions

Data Definition elements are used to define the different data related models. These models are modelling data definitions, interfaces and processors. These models are analysed in this section.

4.2.3.1.1 Processor Definition (PD)

This entity specifies a processing function to be applied on one or more data sources. It can be used to set up a data routing flow and to utilize analytics algorithms as well.

The "PD" has:

- **id:** is the unique ID of a STAR Processor Definition object.
- **name:** is an optional human recognisable name of the STAR Processor Definition object.
- **processorType:** provides the processor type.
- **version:** provides the processor version information.
- **copyright:** which provides the processor ownership information.
- **description:** provide an optional description of the STAR Processor Definition object.
- **processorLocation:** which provides the physical or relative location of the processor (i.e., a service endpoint or a fat application on a local path).
- **AdditionalInformation** (unlimited list): which provides an optional unlimited auxiliary field that may contain any additional information and it is used for further extensions. More specifically it serves as the root of the type definition hierarchy for any schema and it has the unique characteristic that it can function as a complex or a simple type definition, according to context.
- **Parameters:** provides a list of Parameter entities which describes the required parameters that needs to be instantiated in order the processor to function. This entity is analysed with more details in the sections below.

Parameter

Processor Parameter entity describes a parameter which is required to be instantiated in order for a processor to run, i.e., a parameter could be a numerical value which can be used as input for a processor operation or a URL where a processor could push data streams to.

The "Parameter" has:

- **name:** A human-readable name which uniquely identifies the property for a specific processor.
- **description:** Provides an optional description of the property.
- **dataType:** A classification of data (i.e. short, int, float, boolean, ...)
- **defaultValue:** An optional predefined default value for a property.

4.2.3.2 Manifests

The Manifest elements are reusable data models capable of manipulating STAR data. These models are the instantiation of the models specified in the Data Definition above and can be used globally (i.e., PdM) or locally (i.e., CPS).

4.2.3.2.1 Processor Manifest (PM)

This entity represents an instance of a processors that is defined through the PD. The instance specifies the type of processors and its actual logic through linking to a programming function.

The “PM” has:

- **id:** a required unique identification of the Processor instance.
- **ProcessorDefinitionReferenceID:** The Reference ID of the Processor Definition (PD) this PM is used for instantiation.
- **DataSink:** Data Sink provides the description of the data produced from the processor with the help of a Data Source Manifest which it references. A PM is capable of producing one data source so in contains one:
 - **DataSourceManifestReferenceID:** The Reference ID of the Data Source Manifest which describes the output of this PM.
- **DataSources:** this entity provides the Processor instance data sources. A processor instance is capable of handling many data sources so it includes a list of the following entities:
 - **DataSource:** specifies a single Processor Data Source this is achieved by referencing an appropriate Data Source Manifest:
 - **DataSourceManifestReferenceID:** The Reference ID of the Data Source Manifest which identifies a data source of the PM.
- **Parameters:** this entity contains a list of the PD instance configuration parameters. The configuration parameters consist of a key-value pair where the key specifies the Processor Definition parameters this instance refers to (see ProcessorDefinitionReferenceID above)

4.2.3.3 API Definitions

The main API primitives that will be supported by the STAR Distributed Ledger Services for Data Reliability (DLSDR) for the Processor Configuration traceability and will enable their persistence and retrieval are listed in Table 7 below.

Table 7 DLSDR for Processor Configuration metadata API specification overview

HTTP Method	Service	Input	Output	Functional Description
POST	/processor_config/pm	pmDefinition: PM	PM	Used to create a new Processor Manifest (PM) instance to the Blockchain network. It returns the PM instance with an assigned ID.

PUT	/processor_config/:id/pm	pmID: String, pmDefinition:PM	PM	Used to virtually update an existing PM. The service generates a new PM and flags as deleted the id of the provided one. It returns the new registered PM with a new id assigned to it.
DELETE	/processor_config/:id	pmID: String	No	Used to flag the specific PM as deleted.
GET	/processor_config/:id	pmID: String	PM	Used to retrieve an existing PM Instance.
POST	/processor_config/search	pmAttributes: PM	<List> PM	Used to discover available PM instances using attributes of a PM as search criteria.

4.2.3.4 Data Sources persistence interaction

Figure 20 below provides the high-level sequence of interactions for a user/client to persist a Processor configuration to the distributed ledger network and model/registry repositories of STAR solution.

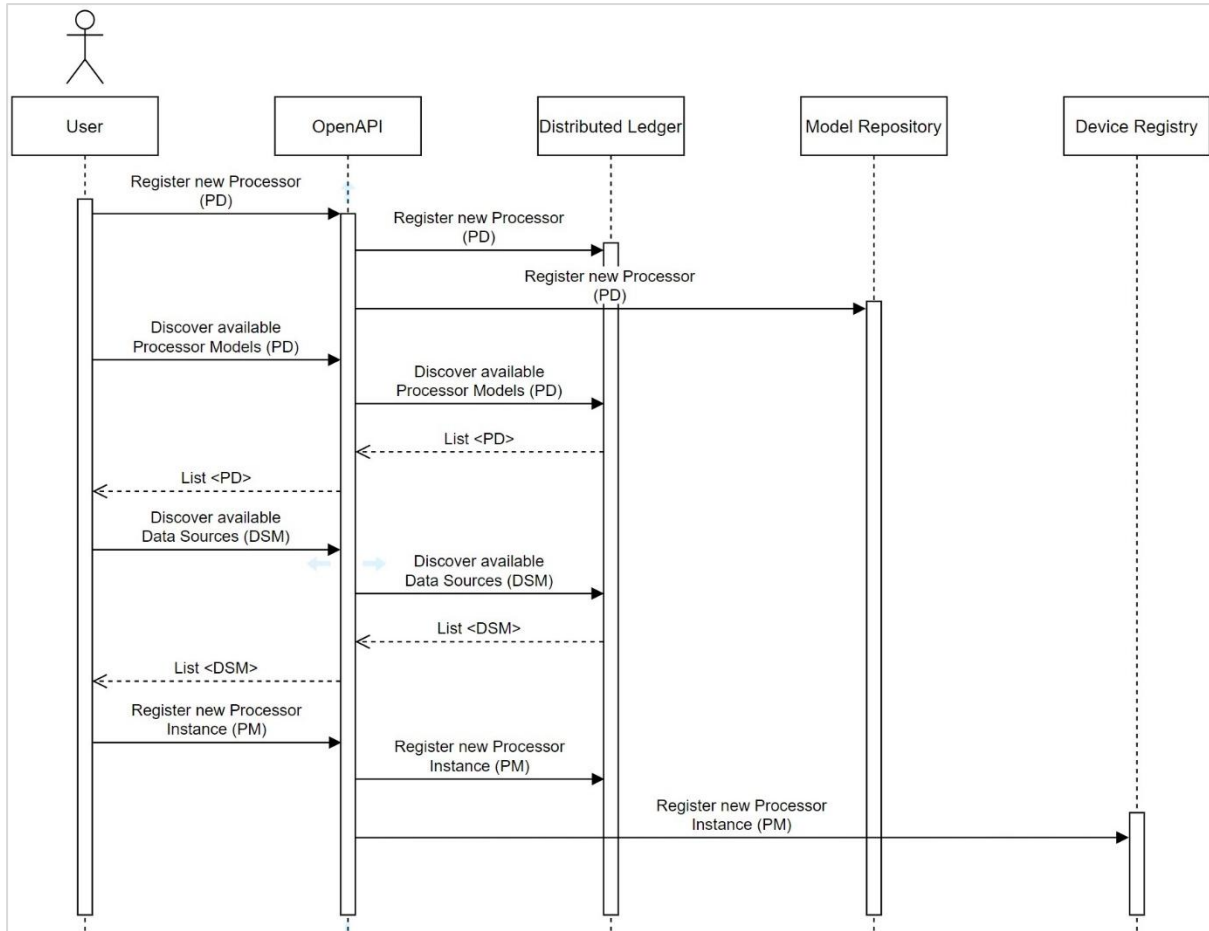


Figure 20 Processor persistence to Distributed Ledger network

As we can see in Figure 20 above the Processor configuration is persisted in parallel to the Distributed Ledger network and the conventional repositories. This is because the Distributed Ledger is used auxiliary to the existent repositories and persistence mechanisms for provenance and validation of the specified data sources.

4.2.4 Algorithm Results Traceability

4.2.4.1 Data Definitions

Data Definition elements are used to define the different data related models. These models are modelling data definitions, interfaces and processors. These models are analysed in this section.

4.2.4.1.1 Observation

The Observation entity models and represents the actual dataset that stems from an instance of a data source that is represented through a DSM. Hence, it references a DSM, which drives the specification of the types of the attributes of the Observation in-line with the DK. An Observation is associated with a timestamp and keeps track of the location of the data source in case it is associated with a mobile (rather than a stationary) data source. Hence, it has a location attribute as well.

The "Observation" has:

- **id:** A unique required ID which is assigned to every observation when captured from the STAR system.
- **dataSourceID:** The ID of the Data Source Manifest (physical or virtual) these observations refer to.
- **dataKindID:** which provides information about the Observation supported Data Kind by referencing an existing DK instance.
- **timestamp:** The timestamp indicating the instance in which a measurement was acquired by the STAR system.
- **Location:** which provides the geographical or virtual location an incident took place. The "Location" has:
 - geolocation: which provides the coordinates (longitude and latitude) of a physical location.
 - virtualLocation: which provides information about a virtual location (it could be the ID of a resource or subsystem).
- **value:** which provides the value of the measurement. The value can be of simple (e.g. a float number depicting temperature) or complex (e.g. a weather station measurements which is consisted by multiple measurements) structure. The type and structure of the value is described in the Data Kind entity.

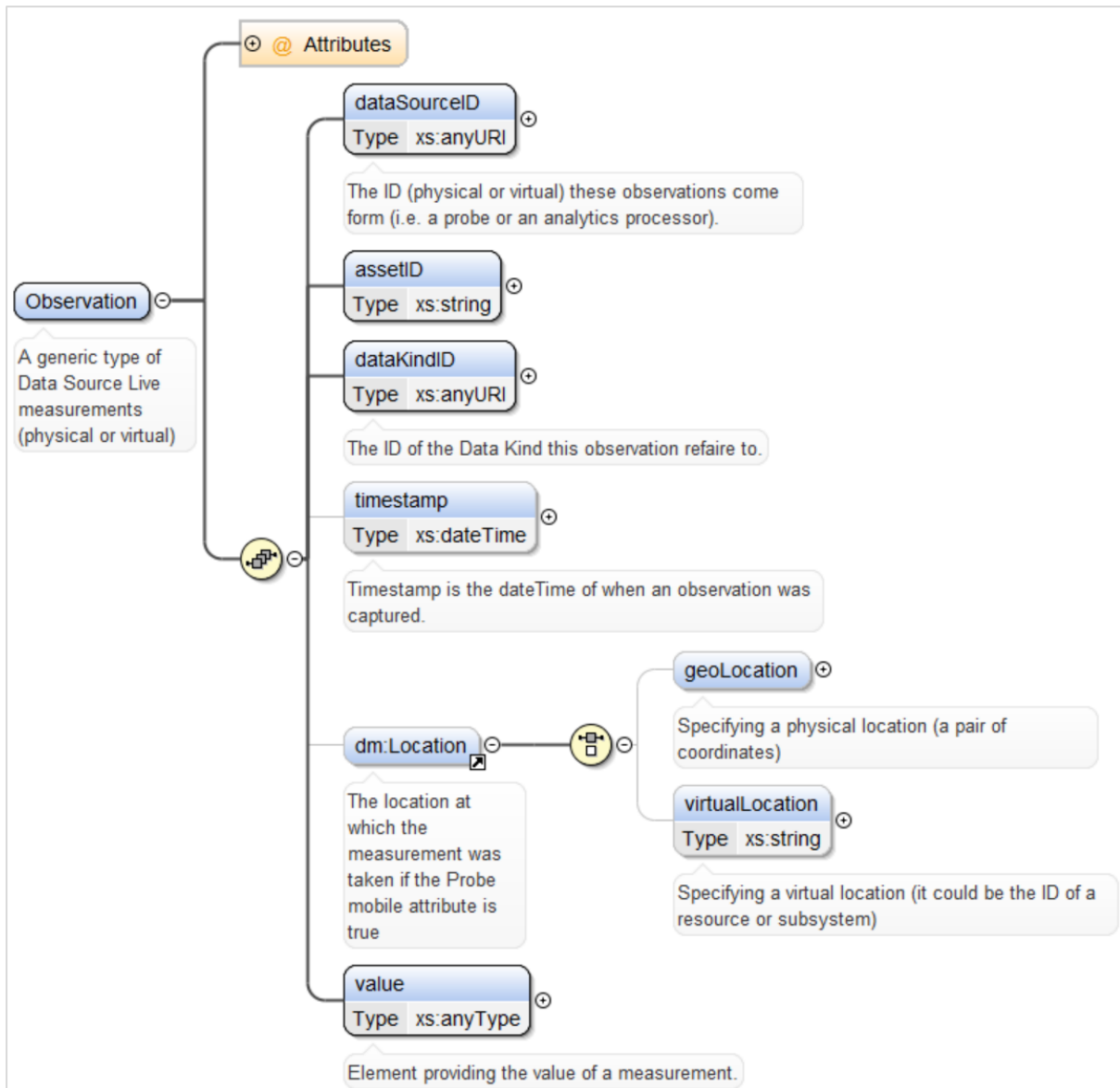


Figure 21 Observation entity of the data model

4.2.4.2 API Specification

The main API primitives that will be supported by the STAR Distributed Ledger Services for Data Reliability (DLSDR) for the algorithm results (Observations) traceability and will enable their persistence and retrieval are listed in Table 8 below.

Table 8 DLSDR for algorithm results API specification overview

HTTP Method	Service	Input	Output	Functional Description
POST	/results/observation	observation: Observation	Observation	Used to upload one Observation instance to the Blockchain network. It returns the Observation

				instance with an assigned ID.
POST	/results/observations	observations: <List> Observation	<List> Observation	Used to upload a list of Observation instances to the Blockchain network. It returns an Observation list with an ID assigned to each one.
GET	/results/:id	observationID: String	Observation	Used to retrieve an existing Observation Instance.
POST	/results/search	observationAttributes: Observation	<List> Observation	Used to discover available Observation instances using attributes of an Observation as search criteria.

4.2.5 Recording Traceability Data on the Blockchain

Figure 22 provides an overview of the sequence of interactions that take place once a client service invokes the STAR distributed ledger services within the ledger network to persist metadata (on data sources, processors, observations described in the previous sections).

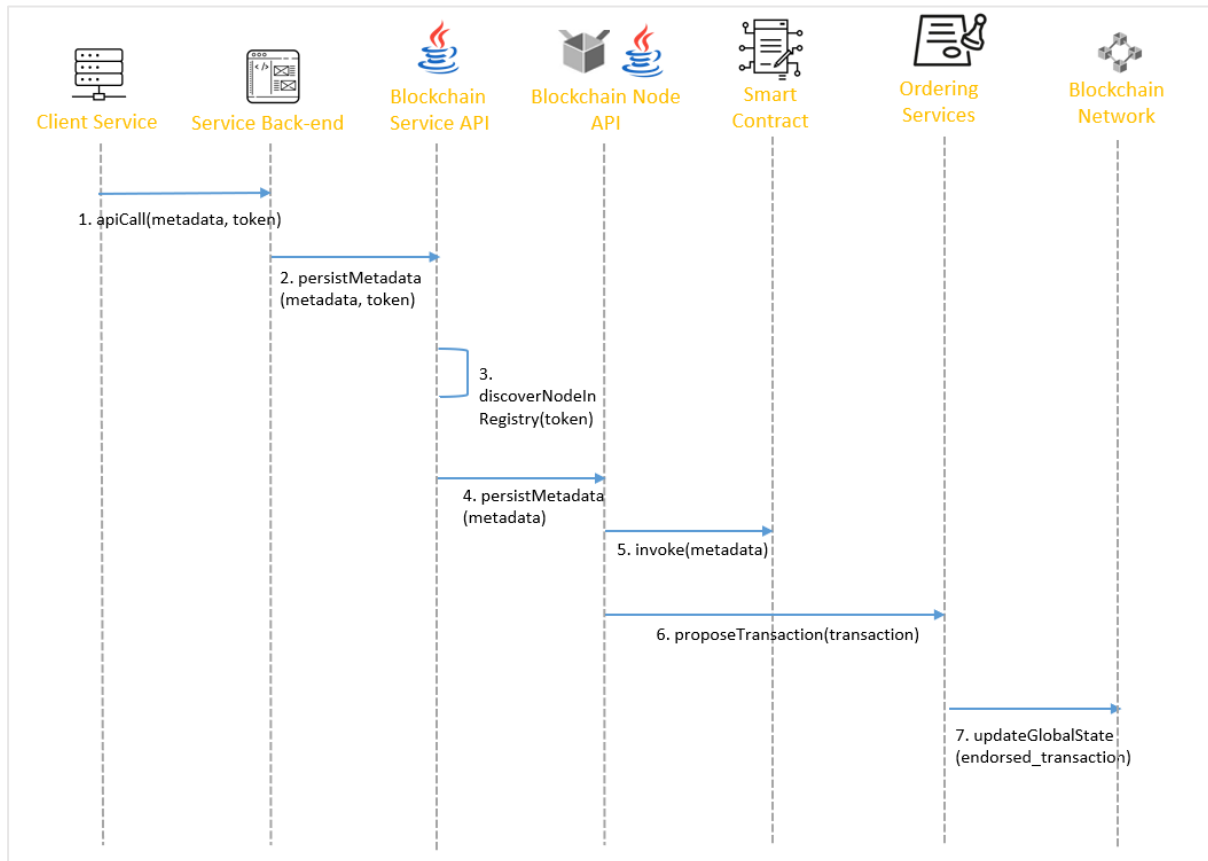


Figure 22: Metadata Persistence using the STAR Blockchain

The execution of the metadata recording on the Ledger is triggered by another platform service that acts as a “client”. Example of such a service can be a service employing AI to perform a numeric estimation, one that can use the Blockchain to record the AI algorithm’s configurations and estimation results.

The Blockchain Service Backend RESTful API is invoked in line with the concept presented in Figure 7. The HTTP call contains metadata on the entities to be persisted as well as the unique identifier of the service performing the call. The API acts as a Façade to the blockchain operations and relies on the Nodes Registry to associate the instigator of a data operation with the Node (and the enveloping dApp) they own on the network.

It subsequently initiates a second HTTP call, identical to the previous (with the user identifier which is no longer needed having been substituted by the Node identifier). The recipient is an inner API serving the business on the level of a particular Organization. The latter communicates with a third API that triggers the execution of the HLF Smart Contract (chaincode). The latter is executed in the blockchain networks and - assuming it does not correspond to a simple query but to an invocation - results in changes in the blockchain. Specifically, following the execution of the Smart Contract the global state of the blockchain

is updated with the metadata. The transaction is assembled into a block (not depicted) after it has been endorsed following the ministrations of an Orderer and then propagated to all Nodes belonging to the same Channel to update their ledger.

In case of high traffic scenarios (i.e., many simultaneous requests), the Blockchain Service API may incorporate a message broker (e.g., an MQTT queue or Apache Kafka) to ensure reliable management and service of requests.

Once the provenance and traceability information have been persisted on the blockchain, they can be queried by any organization/service of the STAR platform in need to verify their authenticity. Such queries might serve different reasons like data audits or cross-verifications before the client service proceeds to some form of actuation on the field. The sequence of interactions following a query is depicted in Figure 23.

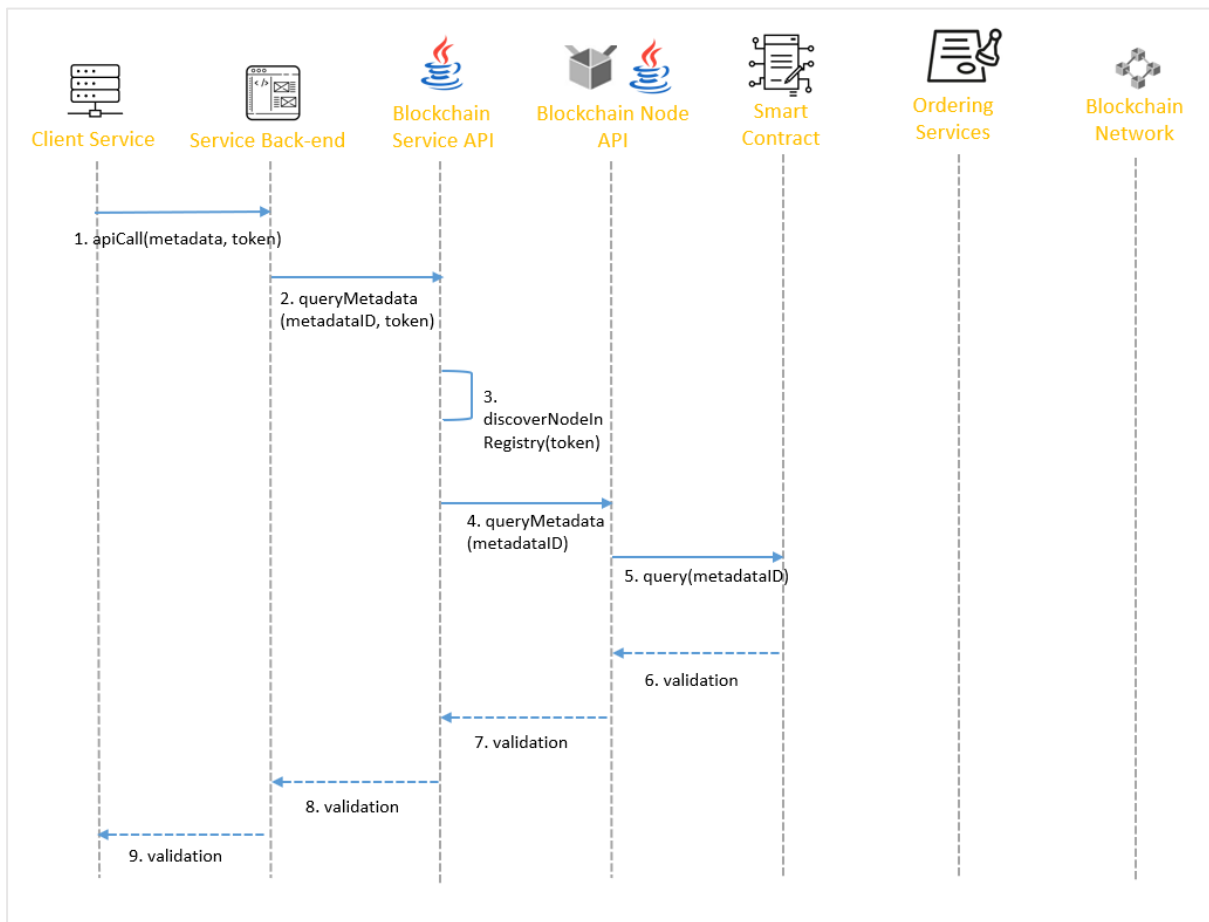


Figure 23: Metadata Validation using the STAR Blockchain

Notice that the first half of the interactions is identical to the process of persisting new data. However, since a simple query does not alter the global state of the distributed ledger, a new transaction is never proposed, the Orderers are never identified and the rest of the network Nodes remain idle.

5 Data Reliability Framework PoC implementation

5.1 Baseline Infrastructure: Hyperledger Fabric

5.1.1 Services Components Deployment in Docker Containers

All the various structural components composing the Blockchain network as well as its integration points with the STAR services that will be using it, will be deployed containerized. Containers are packages hosting applications and their dependencies that can run seamlessly on a variety of locations, such as on-premises, in a public cloud, and/or in a private cloud. They are isolated from one another and bundle their own software, libraries and configuration files; however, they can communicate with each other through well-defined channels. Docker¹⁹ is the platform-as-a-service tool that will be assigned to use OS-level virtualization to deliver the blockchain services' software in such containers.

The Blockchain MVP will assume the existence of three organizations taking part in the circular chain platform. Each will be assumed to maintain a virtual machine hosting their personal Hyperledger Fabric node, as well as its companion applications. To sum up each of those machines will be hosting the following Docker containers:

- A Peer Node²⁰
- A CouchDB where the ledger state is being persisted²¹
- A Certificate Authority (CA)²²
- A Command-Line Interface (CLI)²³
- A Java application exposing an API making available the Node's functionalities to the centralized Blockchain Service Backend and, eventually, the outside world.

One of the machines will be additionally hosting the following Docker containers:

- Multiple instances of the Ordering service
- A Certificate Authority for the above instances
- Fabric Channel(s)
- Chaincode(s)

Images for all those Docker components have been made available by the Hyperledger Fabric development team via DockerHub²⁴. The deployment process has been made semi-automatic by employing Docker Compose²⁵ scripts to pull those images, containerize them and deploy them as Docker Swarm stacks (more details on that are following in the next section). It needs to be highlighted, however, that deployment of Fabric components and

¹⁹ Docker official webpage: <https://www.docker.com/> (accessed April 2022)

²⁰ Peer Node downloadable from DockerHub: <https://hub.docker.com/r/hyperledger/fabric-peer> (accessed April 2022)

²¹ CouchDB downloadable from DockerHub: <https://hub.docker.com/r/hyperledger/fabric-couchdb> (accessed April 2022)

²² Fabric CA downloadable from DockerHub: <https://hub.docker.com/r/hyperledger/fabric-ca> (accessed April 2022)

²³ Fabric CLI downloadable from DockerHub: <https://hub.docker.com/r/hyperledger/fabric-tools> (accessed April 2022)

²⁴ Hyperledger's user account on DockerHub: <https://hub.docker.com/u/hyperledger> (accessed April 2022)

²⁵ Docker Compose official documentation: <https://docs.docker.com/compose/> (accessed April 2022)

the configuration of the network between them is a process more complicated than a simple “docker compose up” command, since rather complex configuration files and TLS certificates ought to have been prepared in advance.

5.1.2 Forming a Cluster Among Distributed Services with Docker Swarm

Hyperledger Fabric is a blockchain protocol constantly gaining in popularity and, therefore, the community of developers forming around it is constantly honing various deployment strategies for production releases. One such approach, used in the context of STAR to create a Proof-of-Concept for its Blockchain-oriented services, is to employ Docker Swarm²⁶ for creating a cluster of interconnected Nodes deployed within Docker containers. A swarm consists of multiple Docker hosts which run in swarm mode and act as **managers** (to manage membership and delegation) and **workers** (which run swarm services). A given Docker host can be a manager, a worker, or perform both roles. When the network administrator creates a service, they define its optimal state (number of replicas, network and storage resources available to it, ports the service exposes to the outside world, and more). Docker works to maintain that desired state. For instance, if a worker becomes unavailable, Docker schedules its tasks on other swarm participants. A task is a running container which is part of a swarm service and managed by a swarm manager, as opposed to a standalone container.

A swarm node - not to be confused with a Fabric node - is an instance of the Docker engine participating in the swarm. One can run one or more nodes on a single physical computer or cloud server, but production swarm deployments typically include Docker nodes distributed across multiple physical and cloud machines. To deploy an application enveloped in a Docker container to a swarm, one submits a service definition to a manager node. The manager node dispatches units of work called tasks to worker nodes. Manager nodes also perform the orchestration and cluster management functions required to maintain the desired state of the swarm. Manager nodes elect a single leader to conduct orchestration tasks. Worker nodes receive and execute tasks dispatched from manager nodes. An agent runs on each worker node and reports on the tasks assigned to it. The worker node notifies the manager node of the current state of its assigned tasks so that the manager can maintain the desired state of each worker.

Swarm manager nodes use the Raft Consensus Algorithm to manage the swarm state. There is no limit on the number of manager nodes. The decision about how many manager nodes to implement is a trade-off between performance and fault-tolerance. Adding manager nodes to a swarm makes the swarm more fault-tolerant. For testing purposes, it is acceptable to run a swarm with a single manager. If the manager in a single-manager swarm fails, the deployed services continue to run, but creation of a new cluster is needed to recover. Hence, it is overall acceptable and possible to test the system based on a single manager node. To take advantage of swarm mode’s fault-tolerance features, Docker recommends administrators to implement an odd number of nodes according to their organization’s high-availability requirements. When one has multiple managers, they can recover from the failure of a manager node without downtime [SwarmDocs]. Figure 24 summarises the concepts described above.

²⁶ Official documentation of Swarm within Docker’s official website: <https://docs.docker.com/engine/swarm/> (accessed April 2022)

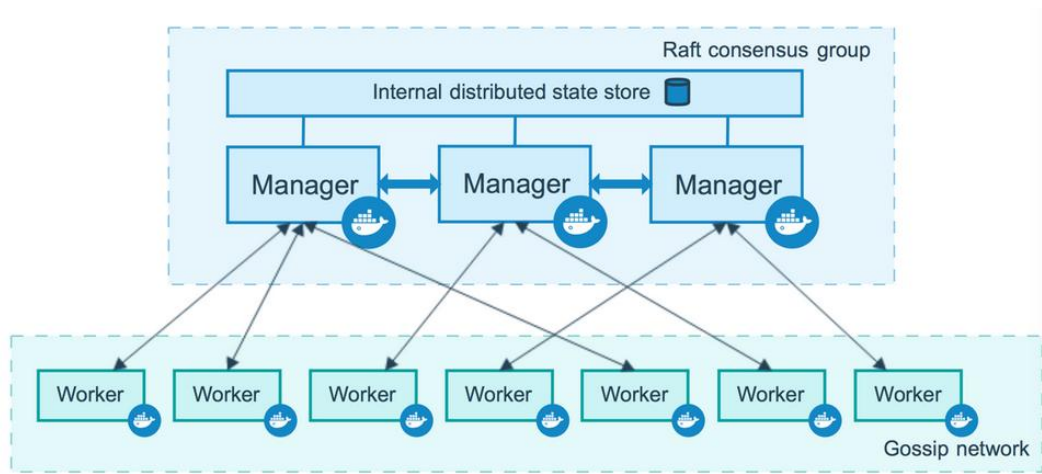


Figure 24: A Docker Swarm Cluster Example [SwarmDocs]

To deploy an application image when Docker Engine is in swarm mode, one creates a service. Frequently a service is the image for a microservice within the context of some larger application. When the service is deployed to the swarm, the swarm manager accepts the service definition as the desired state for the service. Then it schedules the service on nodes in the swarm as one or more replica tasks. The tasks run independently of each other on nodes in the swarm.

The swarm mode public key infrastructure (PKI) system built into Docker makes it simple to securely deploy a container orchestration system. The nodes in a swarm use mutual Transport Layer Security (TLS) to authenticate, authorize, and encrypt the communications with other nodes in the swarm. When a swarm is being created, Docker designates itself as a manager node. By default, the manager node generates a new root Certificate Authority (CA) along with a key pair, which are used to secure communications with other nodes that join the swarm. The manager node also generates two tokens to use when one joins additional nodes to the swarm: one worker token and one manager token. Each token includes the digest of the root CA's certificate and a randomly generated secret. When a node joins the swarm, the joining node uses the digest to validate the root CA certificate from the remote manager. The remote manager uses the secret to ensure the joining node is an approved node. Each time a new node joins the swarm, the manager issues a certificate to the node [SwarmDocs]. Figure 25 summarises the concepts described above.

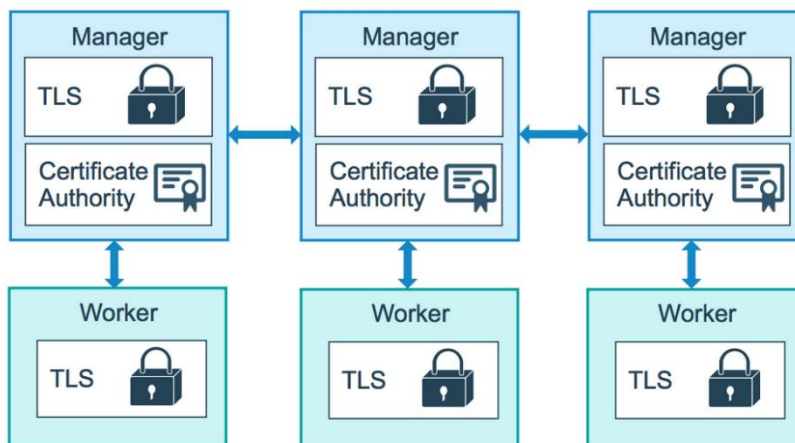


Figure 25: Swarm Security Concept with PKI [SwarmDocs]

The following section discusses the topology devised for the cluster of the Docker containers synthesizing the blockchain services of STAR’s MVP. Even though the Hyperledger Fabric requires additionally its own Raft protocol implementation to achieve consensus - this time between Fabric Nodes while we have just described consensus between Swarm Nodes - and its proper TLS authentication scheme, the concepts are strikingly similar.

5.1.3 Docker Swarm Overlay Topology for STAR PoC Fabric Network

Bringing together the concepts described in the two previous sections, it is time to walk through the deployment plan for the proof-of-concept architecture devised for the needs of the STAR MVP. It has been assumed that only three circular chain stakeholders participate on the network, holding peer roles. To stay faithful to the decentralization principles and to showcase how to handle future scaling, the components belonging to each organization have been deployed in different virtual machines, whose networking and orchestration functionalities are being handled by formulating a Docker Swarm cluster. The distribution of the containerized components into machines is visualised in Figure 26.

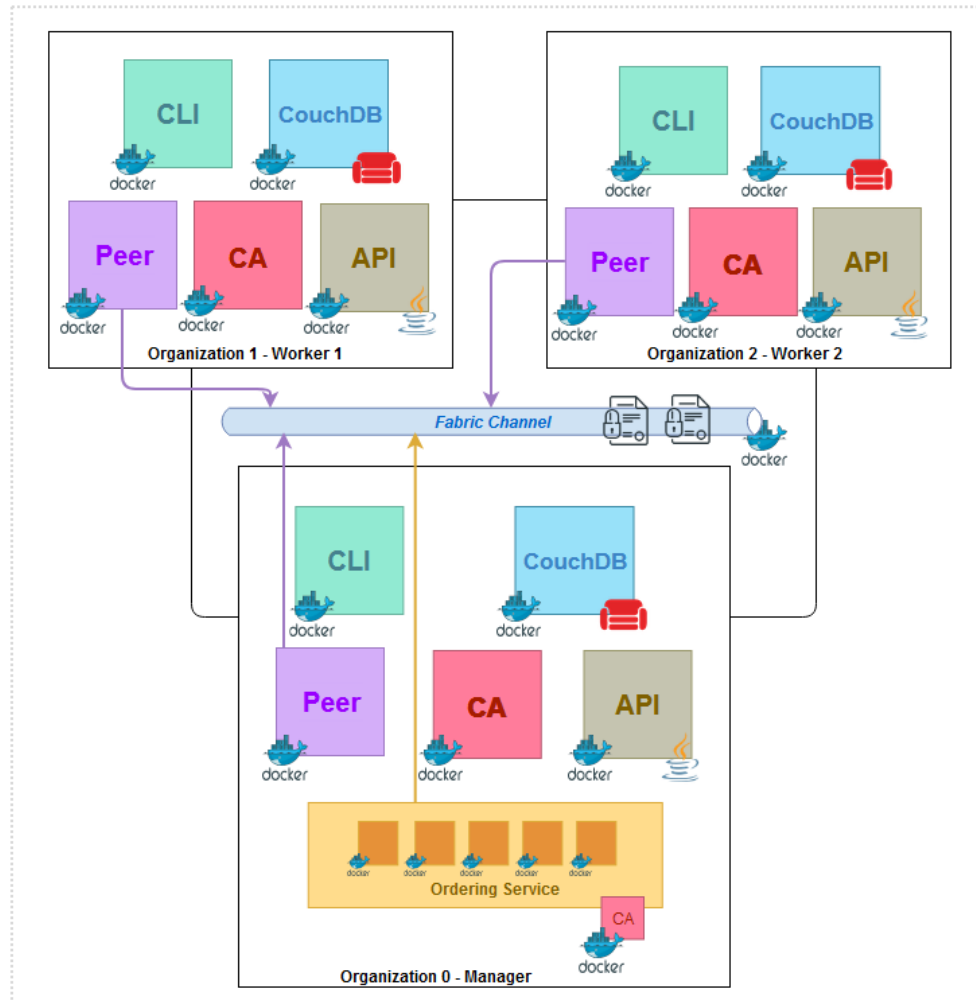


Figure 26: Docker Swarm Overlay Topology for STAR MVP Fabric Network

For the time being the three Fabric Organizations remain anonymous and will be therefore referenced to as "zero", "one" and "two". As far as STAR's blockchain services are concerned we might assume those three use case scenarios:

- Organization "one" offers to the platform a data stream addressed to Organization "two". Metadata on the source of the data are stored on the Blockchain. Organization "two" that needs at a later point to verify the source of the data stream may refer to the Blockchain (see section 4.2.2).
- Organization "one" offers to the platform a service leveraging an artificial intelligence algorithm (data processor). Metadata on the processor's configuration (state) right before its execution are stored on the Blockchain. Organization "two", another stakeholder of the STAR platform, that needs at a later point to verify which processor and under which conditions performed the data processing may refer to the Blockchain (see section 4.2.3).
- Organization "one" offers to the platform a service leveraging an artificial intelligence algorithm (data processor). Metadata on the results of the algorithm's calculations are stored on the Blockchain. Organization "two", another stakeholder of the STAR platform, that needs at a later point to verify a result they might have come across via a different route may refer to the Blockchain to assert that it has not been

tampered (see section 4.2.3).

As evidenced by Figure 26, Organizations “one” and “two” are hosted by two distinct virtual machines carrying the role of Workers within the Docker Swarm network. For the transition to a production-level deployment, the administrator can replicate those Workers to match the number of stakeholders requiring services from Hyperledger Fabric. The Manager within the Docker Swarm network corresponds to a third virtual machine. There resides Organization “zero” and a set of Fabric Orderers (accompanied by its Certification Authority). This configuration is not mandatory for the future full-scale model. The latter could employ an odd number of Managers hosting only replicas of the Fabric Orderers and network monitoring/administration tools. In case of failure of the Leading Manager another could take the mantle of overseeing the network processes. For the MVP the Manager hosts also an Organization just to cut down on resource expenses.

The components that constitute the “Organization” necessities have already been described: the actual Fabric Peer Node, a Certification Authority, a Command-Line Interface for administration tasks, a CouchDB instance to persist the global state and a Java Spring²⁷ Boot²⁸ Application exposing an API with the business logic to the outside world. All shall be deployed in Docker containers. Fabric Channels are, in essence, also materialized through Docker containers. Those can be hosted anywhere but let us assume that they will be hosted also within the Manager machine for clarity. Smart Contracts are also being deployed as Docker containers and associated strictly with Fabric Channels. All Peers and the set of Orderers are then attached to Channels for them to effectively share the global state of the distributed ledger.

5.1.4 Cloud Infrastructure Preparation

The recommended configuration for a Manager workstation that will be running all the services listed in the previous section is the following:

- 160GB of disk storage (excluding OS)
- 16GB RAM
- 4 cores CPU

Accordingly for a Worker workstation:

- 160GB of disk storage (excluding OS)
- 8GB RAM
- 4 cores CPU

For the MVP the virtual machines of choice have been equipped with CentOS Linux 7 operating system. Key preparation points for the machines are the following:

- Preparation for a distributed environment, which enables remote access and security-by-design. Deploying the software components and services on different VMs, makes it easier to horizontally scale the platform by introducing additional resources where necessary.
- Secure communication among the deployed software components achieved by

²⁷ Spring official webpage: <https://spring.io/> (accessed April 2022)

²⁸ Spring Boot official webpage: <https://spring.io/projects/spring-boot> (accessed April 2022)

encrypted communications over TLS/HTTPS protocols.

- Encryption at rest for ensuring data is inaccessible to malicious parties on the event of misplacement of the physical disk units that host the virtual servers.
- Isolation and protection from unauthorized access hosts thanks to established firewall policies and rules.
- Additional Linux servers security best-practices such as: deactivation of password login (keeping SSH connections as the only option, deactivation of root user, protection of ports that are usually targeted by hackers etc.).

5.1.5 Outline of the Network Installation Steps

The present section lists the installation steps for the Fabric network on a high level. More analytical technical instructions will follow on upcoming deliverables such as D3.2 and those of WP6 revolving around the integration of the services and their subsequent installation to the pilot sites.

Step1: Installation of prerequisites, such as git, docker, docker-compose, Hyperledger Fabric images to the appropriate machines.

Step2: Setup of the multi-node cloud Swarm: assignment of Manager and Worker roles, creation of network, human-readable labelling for monitoring purposes.

Step3: (Optional) Installation of monitoring tools, such as Portainer, Swarmpit, and others.

Step4: Preparation of docker-compose scripts for certificate authorities, command-line, CouchDB and peers, as well as the Fabric configuration files.

Step5: Certificate Authorities setup.

Step6: Generation of Certificates for Peers and Orderers and relocation to the appropriate machines/directories.

Step7: Generation of blockchain Genesis block, channel transaction and anchor peers. Relocation of the latter to the appropriate machines/directories.

Step8: Instantiation of Peers, Orderers and CouchDB instances.

Step9: Creation of Channels and Nodes assignment to said Channels.

Step10: Chaincode installation and testing.

Step11: Deployment and testing of the API servers.

5.2 Infrastructure Management

5.2.1 Monitoring the Swarm Network

The command line provides an efficient and useful way for users to control Docker containers and Swarm. This can gradually get a confusing as services and nodes multiply. Therefore, in a production environment managing and monitoring a swarm can be facilitated by a web interface. Two of the most prominent open-source tools are proposed below, however, there are many other high-profile options available as well.

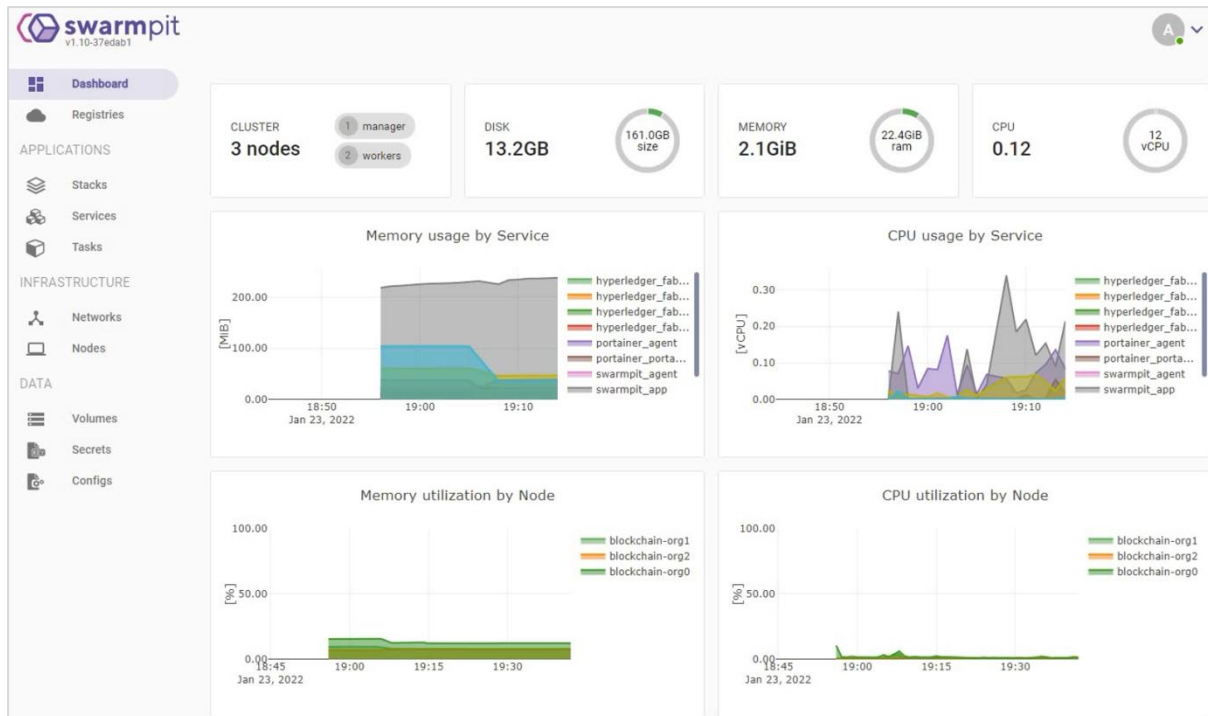


Figure 27: SwarmPit Master Dashboard for Resources Monitoring

5.2.1.1 SwarmPit

SwarmPit²⁹ is a lightweight Docker Swarm cluster management GUI. Among the most prominent features of this open-source tool one can count:

- **Stack management:** composition of a new stack manually or automatically generated from application state.
- **Resource monitoring:** display of information about the use of hardware (CPU, memory, disk) in real-time.
- **Service management:** deployment and management of Swarm services.
- **Smart search:** search for images across public and private registries.
- **Shared access:** multiple users are able to manage a Docker Swarm cluster safely.
- **Private registry:** allows "pull" from private repositories from Docker Hub or custom registries.
- **Service auto-redeploy:** checks whether new service image has been published and updates service accordingly.

²⁹ SwarmPit official webpage: <https://swarmplit.io/> (accessed April 2022)

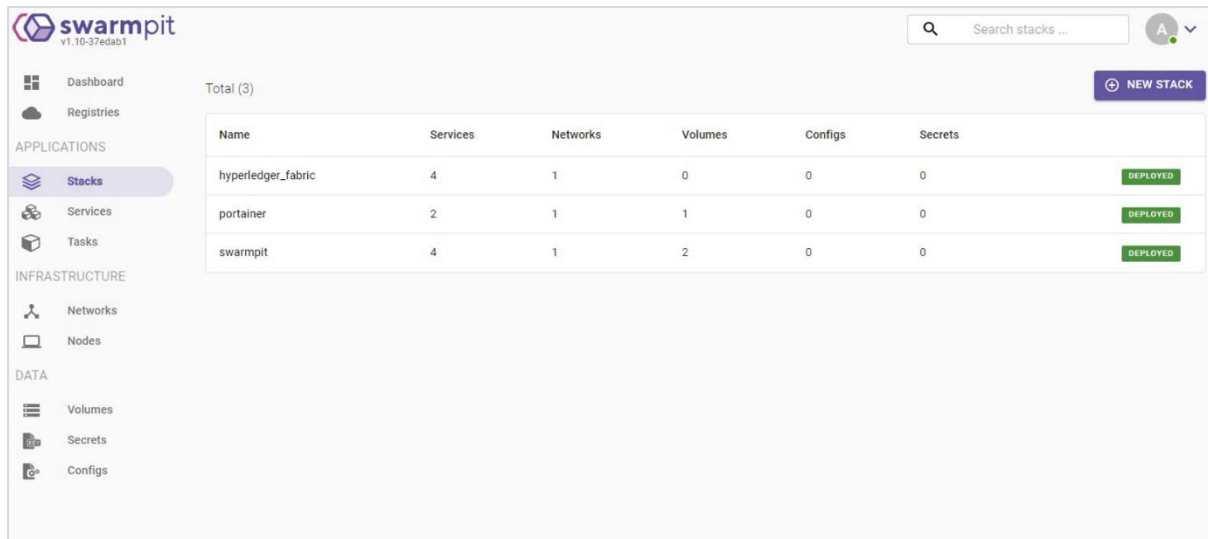


Figure 28: Swarmpit Depicting Deployed Docker Swarm Stacks

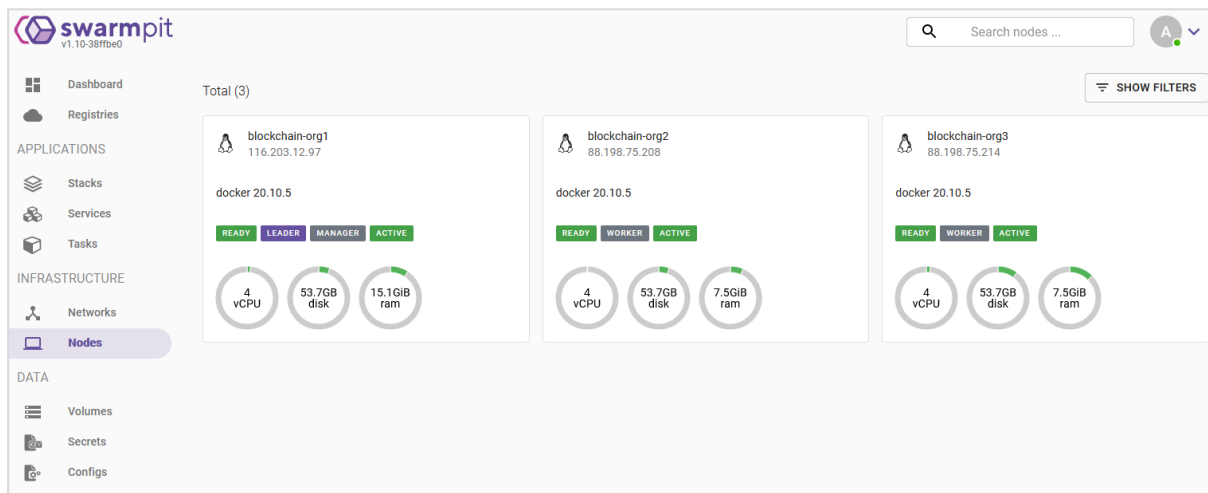


Figure 29: Swarmpit Depicting the Virtual Machines Formulating a Network

The dashboard of Docker Swarm is illustrated in Figure 27, Figure 28 and Figure 29.

5.2.1.2 Portainer

Portainer³⁰ is, similarly, a lightweight management GUI which allows administrators to easily manage different Docker environments (Docker hosts or Swarm clusters). Portainer is meant to be as simple to deploy as it is to use. It consists of a single container that can run on any Docker engine (can be deployed as Linux container or a Windows native container). It can be used to deploy and manage applications, observe the behaviour of containers and provide the security and governance necessary to deploy containers widely. It is both compatible with the standalone Docker engine and with Docker Swarm mode. The dashboard of Portainer is illustrated in Figure 30, Figure 31 and Figure 32.

³⁰ Portainer official website: <https://www.portainer.io/> (accessed April 2022)

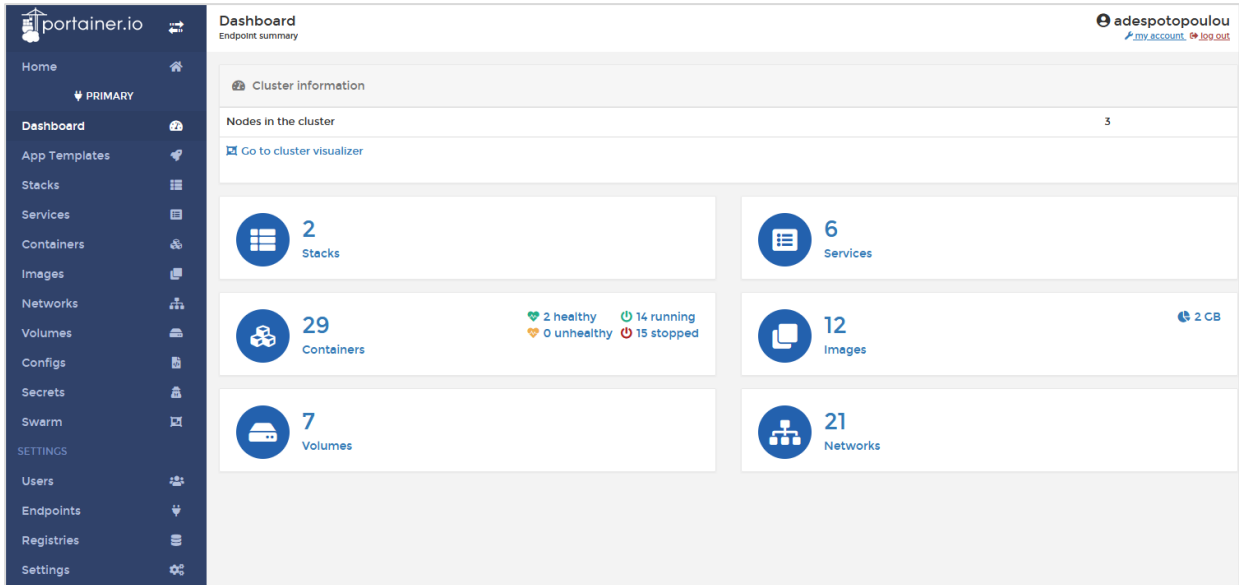


Figure 30: Portainer Landing Page for Containers Administration

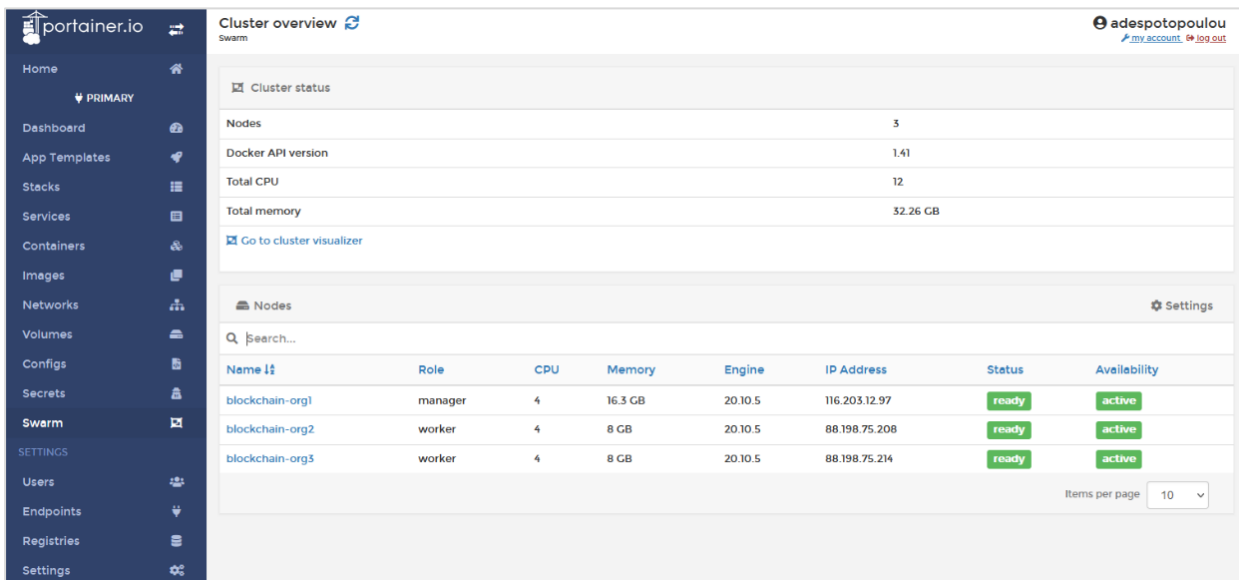


Figure 31: Portainer Swarm Cluster Overview

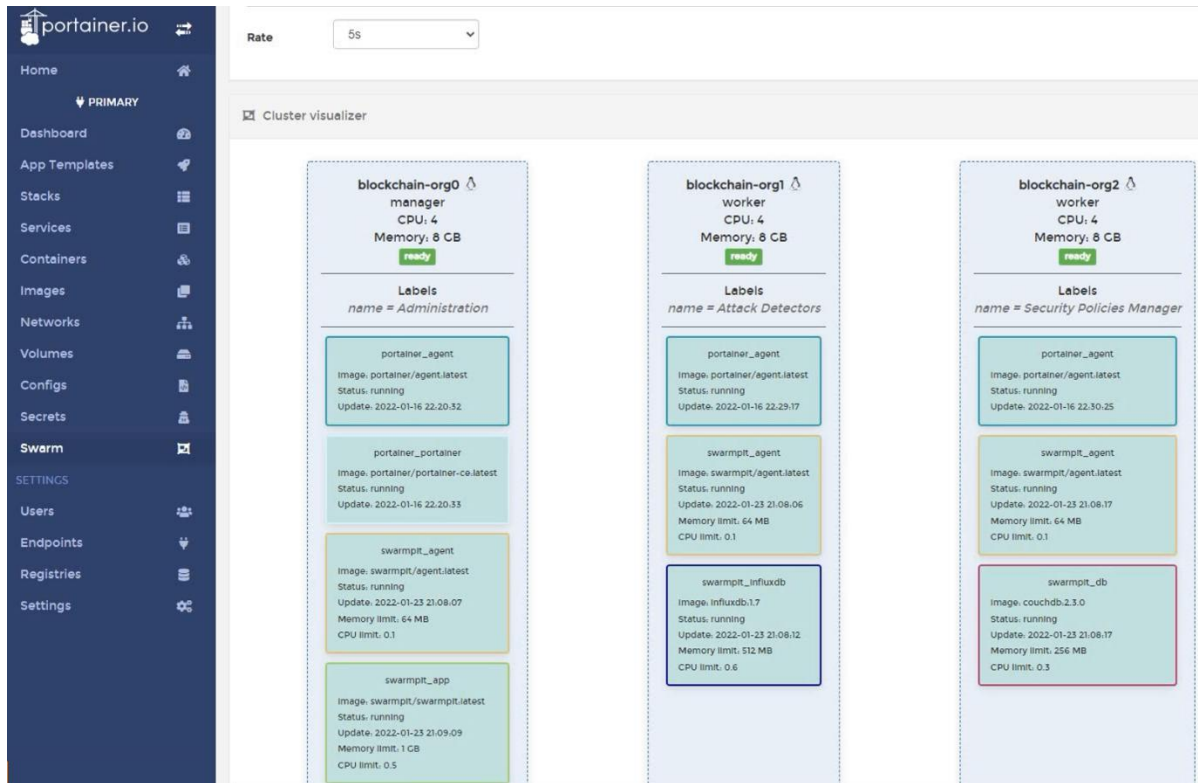


Figure 32: Portainer Swarm Visualizer with Nodes Assigned to STAR Services

5.2.2 Monitoring the Blockchain Network

Swarmpit and Portainer introduced in the previous section address the need of an administrator to monitor resources across the deployed infrastructure that may span multiple virtual machines. What would be additionally useful would be a tool to monitor the actual intricacies of the Hyperledger Fabric components, such as the Nodes deployed, the Channels instantiated, the Chaincode deployed etc.

5.2.2.1 Hyperledger Explorer

Hyperledger Explorer ³¹ is a user-friendly open-source Web application tool used to view, invoke, deploy or query blocks, transactions and associated data, network information (name, status, list of nodes), chain codes and transaction families, as well as any other relevant information stored in the Hyperledger Fabric ledger. The project was contributed by DTCC, Intel, and IBM. Figure 33 illustrates the dashboard of the Explorer. This utility enables a user to [ExplorerDocs]:

- Retrieve the latest status blocks, network and chaincode, view blocks, and transactions.
- Retrieve blocks and transactions metrics by hours, and minutes.
- Search, and filter blocks, transactions by date range and channels.
- Dynamically discover new channels and switch data presentation by channels.
- Get real time notification of new blocks.

³¹ Hyperledger Explorer official website: <https://www.hyperledger.org/use/explorer> (accessed April 2022)

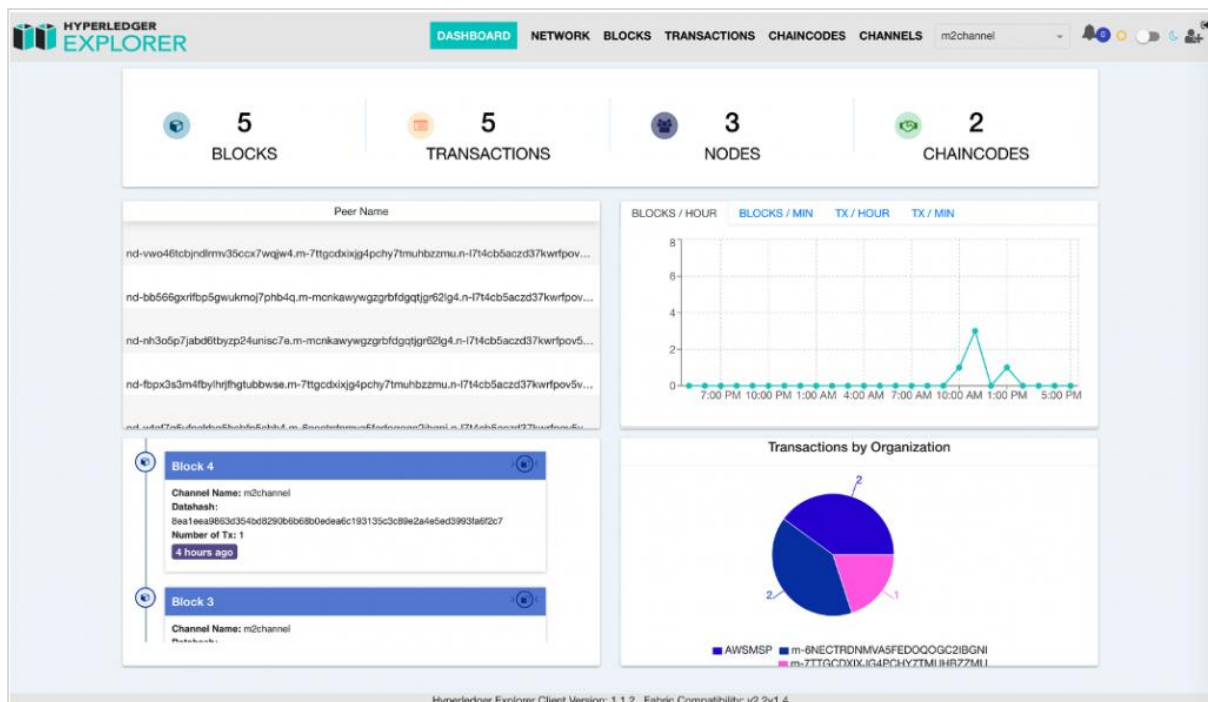


Figure 33: Hyperledger Explorer Dashboard

5.3 Blockchain Service Backend

As discussed at length in Section 4 and depicted in Figure 7, there is an integration point between the various STAR services and the Blockchain network. It acts as a Façade to the blockchain operations, bridging the platform services with the network nodes. It employs information from a database, called **Nodes Registry**, to associate the instigator of a blockchain transaction to the particular Node they are supposed to own.

This architectural decision has been made in order to facilitate the various platform services developers from configuring and maintaining information regarding the topology of the blockchain network. It is important to highlight that this does not violate the decentralized paradigm; the various Fabric Nodes are being deployed across different machines (that could be even maintained within the respective Organizations' premises) while there is a single Service Backend instance deployed along with the other centralized services of the platform to keep the Blockchain network connected to both the authentication service and the pool of STAR services. In case of a malicious attack or a downtime incident on this Façade, the data stored on the blockchain continue to remain safe until the Façade is properly restored.

Technically speaking, this Service Backend is a Java Spring Boot application, also residing in a Docker container and deployed using a Docker Compose script. It is being deployed only once. It exposes a RESTful API to accept HTTP calls, bearing information on the authenticated sender. After mapping this sender to an Organization and Node via the Nodes Registry, the Backend redirects the HTTP call to the API exposed by the particular Node (depicted with golden boxes in Figure 26).

6 Conclusions

Blockchain technology provides powerful functionalities for data reliability in industrial environments. Specifically, it offers decentralized, secure and tamper-proof logging of data, which facilitates the creation and maintenance of a single version of the truth about the status of data entities. The STAR blockchain leverages these properties to provide provenance and tracking of industrial data used in AI systems.

This deliverable has illustrated the benefits of blockchain technologies for data provenance and reliability. It has also shed light on the limitations of blockchain applications when it comes to supporting scalable, high performance, enterprise scale deployments. Moreover, it has presented how permissioned blockchains alleviate the performance limitations of public blockchains for industrial applications. Based on this analysis, the use of permissioned blockchain infrastructure in STAR has been justified and the HLF blockchain has been selected as a baseline distributed ledger infrastructure. The selection has been based on a variety of criteria and following the benchmarking of HLF against other popular permissioned blockchains.

The STAR blockchain solution is aligned to the STAR architecture which was presented in D2.6. The deliverable provides a sound basis for the specification and implementation of the Distributed Ledger Services for Data Reliability of the project.

Overall, the next steps to the implementation and use of the blockchain services in STAR include: (i) The integration of the HLF blockchain infrastructure in the STAR platform and its testing based on pilot scenarios used in the scope of WP6; (ii) The use of the blockchain to persist data from AI Algorithm services in WP3 and WP4, based on the integration of proper service end-points with the APIs of the STAR blockchain.

References

Reference	Name of document
[Abu18]	Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, and P. Martineau, "A parallel graph edit distance algorithm", <i>Expert Syst. Appl.</i> , vol. 94, pp. 41–57, 2018. doi: 10.1016/j.eswa.2017.10.043.
[Angrish18]	A. Angrish, B. Craver, M. Hasan and B. Starly, "A case study for blockchain in manufacturing: 'FabRec': A prototype for peer-to-peer network of manufacturing nodes", <i>Procedia Manuf.</i> , vol. 26, pp. 1180-1192, 2018.
[Atlam18]	Atlam, H.F.; Alenezi, A. et. al.. Blockchain with Internet of Things: Benefits, Challenges, and Future Directions. <i>Int. J. Intell. Syst. Appl.</i> 2018.
[Baker17]	B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in 6th International Conference on Learning Representations, 2017.
[Barenji18]	A. V. Barenji, Z. Li, and W. M. Wang, "Blockchain cloud manufacturing: Shop floor and machine level," in <i>Proc. Eur. Conf. Smart Objects, Syst. Technol.</i> Munich, Germany: VDE, Jun. 2018, pp. 1–6.
[Bhardwaj18]	G. Bhardwaj. (Apr. 2018). Why Pharma's Manufacturing Supply Chain Needs Blockchain Innovation. <i>Pharmaphorum</i> . Accessed: April. 2022. [Online]. Available: https://pharmaphorum.com/views-and-analysis/
[Bhattacharya19]	P. Bhattacharya, S. Tanwar, U. Bodke, S. Tyagi and N. Kumar, "BinDaaS: Blockchain-based deep-learning as-a-service in healthcare 4.0 applications", <i>IEEE Trans. Netw. Sci. Eng.</i> , Dec. 2019.
[Bodkhe20]	U. Bodkhe et al., "Blockchain for Industry 4.0: A Comprehensive Review," in <i>IEEE Access</i> , vol. 8, pp. 79764-79800, 2020, doi: 10.1109/ACCESS.2020.2988579.
[Bougleux17]	S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, and M. Vento, "Graph edit distance as a quadratic assignment problem", <i>Pattern Recognit. Lett.</i> , vol. 87, pp. 38–46, 2017. doi: 10.1016/j.patrec.2016.10.001.
[Bougleux18]	S. Bougleux, B. Gaüzère, D. B. Blumenthal, and L. Brun, "Fast linear sum assignment with error-correction and no cost constraints", <i>Pattern Recognit. Lett.</i> , 2018, in press. doi: 10.1016/j.patrec.2018.03.032.
[Brilliantova19]	V. Brilliantova and T. W. Thurner, "Blockchain and the future of energy", <i>Technol. Soc.</i> , vol. 57, pp. 38-45, May 2019.
[Bringmann17]	Bringmann, K., Gawrychowski, P., Mozes, S., Weimann, O.: Tree edit distance cannot be computed in strongly subcubic time (unless APSP can). <i>CoRR</i> , abs/1703.08940 (2017)
[Bunke11]	H. Bunke and K. Riesen, "Improving vector space embedding of graphs through feature selection algorithms", <i>Pattern Recognit.</i> , vol. 44, no. 9, pp. 1928–1940, 2011. doi: 10.1016/j.patcog.2010.05.016.
[Carletti15]	V. Carletti, B. Gaüzère, L. Brun, and M. Vento, "Approximate graph edit distance computation combining bipartite matching and exact neighborhood substructure distance", in <i>GbRPR 2015</i> , C. Liu, B. Luo, W. G. Kropatsch, and J. Cheng, Eds., ser. LNCS, vol. 9069, Cham:

	Springer, 2015, pp. 188–197. doi: 10.1007/978-3-319-18224-7_19.
[Chainstack21]	Chainstack.com, "Enterprise Blockchain Protocols Evaluation Index", April 2021.
[Chen01]	Chen, W.: New algorithm for ordered tree-to-tree correction problem. <i>J. Algorithms</i> 40(2), 135–158 (2001)
[Christidis16]	K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the IoT", <i>IEEE Access</i> , Volume 4, pp. 2292-2303, 2016.
[Circlecell22]	Circlecell. 2022. JSON Compare. https://jsoncompare.com/ .
[Cobena02]	Gregory Cobena, Serge Abiteboul, and Amelie Marian. 2002. Detecting changes in XML documents. In <i>Proceedings of the 18th International Conference on Data Engineering</i> . IEEE, 41–52.
[Daskalakis20]	N. Daskalakis and P. Georgitseas, "An Introduction to Cryptocurrencies: The Crypto Market Ecosystem", Routledge, pp. 26-27, 2020. ISBN 978-0-367-37078-7.
[DeCao18]	N. De Cao, T. Kipf, MolGAN: an Implicit Generative Model for Small Molecular Graphs, <i>ICML 2018 Workshop on Theoretical Foundations and Applications of Deep Generative Models (2018)</i>
[ExplorerDocs]	Official Hyperledger Explorer Documentation, accessed April 2022. Documents available online: https://blockchain-explorer.readthedocs.io/en/main/index.html
[FabricDocs]	Official Hyperledger Fabric Documentation (version 2.3.2), retrieved in June 2021. Licensed under a Creative Commons Attribution 4.0 International License. Licence details available online: http://creativecommons.org/licenses/by/4.0/
[Fernandez-Carames18]	Fernandez-Carames, T.M.; Fraga-Lamas, P. A Review on the Use of Blockchain for the Internet of Things. <i>IEEE Access</i> 2018.
[Finis13]	Jan P Finis, Martin Raiber, Nikolaus Augsten, Robert Brunel, Alfons Kemper, and Franz Färber. 2013. Rws-diff: flexible and efficient change detection in hierarchical data. In <i>Proceedings of the 22nd ACM international conference on Information & Knowledge Management</i> . 339–348.
[Fraga-Lamas19]	P. Fraga-Lamas and T. M. Fernandez-Carames, "A review on blockchain technologies for an advanced and cyber-resilient automotive industry", <i>IEEE Access</i> , vol. 7, pp. 17578-17598, 2019.
[Frasconi98]	P. Frasconi, M. Gori, A. Sperduti, A general framework for adaptive processing of data structures, <i>IEEE TNN</i> , 9 (1998), pp. 768-786
[Gauzere12]	B. Gaüzère, L. Brun, and D. Villemin, "Two new graphs kernels in chemoinformatics", <i>Pattern Recognit. Lett.</i> , vol. 33, no. 15, pp. 2038–2047, 2012. doi: 10.1016/j.patrec.2012.03.020.
[Goranovic17]	A. Goranovic, M. Meisel, L. Fotiadis, S. Wilker, A. Treytl and T. Sauter, "Blockchain applications in microgrids an overview of current projects and concepts", <i>Proc. IECON-43rd Annu. Conf. IEEE Ind. Electron. Soc.</i> , pp. 6153-6158, Oct. 2017.
[Grossbart21]	Zack Grossbart. 2021. JSON Diff. http://www.jsondiff.com .
[Grover16]	A. Grover, J. Leskovec, node2vec: scalable feature learning for networks, <i>Proceedings of KDD, ACM (2016)</i> , pp. 855-864
[Grover19]	A. Grover, A. Zweig, S. Ermon, Graphite: iterative generative modeling of graphs, <i>Proceedings of ICML (2019)</i> , pp. 2434-2444
[Hirsh18]	S. Hirsh, S. Alman, V. Lemieux and E. T. Meyer, "Blockchain: One emerging technology—So many applications", <i>Proc. Assoc. Inf. Sci.</i>

	Technol., vol. 55, no. 1, pp. 691-693, 2018.
[Holland18]	M. Holland, J. Stjepandic and C. Nigischer, "Intellectual property protection of 3D print supply chain with blockchain technology", Proc. IEEE Int. Conf. Eng. Technol. Innov. (ICE/ITMC), pp. 1-8, Jun. 2018.
[Hua18]	J. Hua, X. Wang, M. Kang, H. Wang, and F. Wang, "Blockchain based provenance for agricultural products: A distributed platform with duplicated and shared bookkeeping," in 2018 IEEE Intelligent Vehicles Symposium (IV), pp. 97–101, June 2018.
[Huetter19]	Thomas Hütter, Mateusz Pawlik, Robert Löschinger, and Nikolaus Augsten. 2019. Effective filters and linear time verification for tree similarity joins. In Proceedings of the 35th International Conference on Data Engineering. IEEE, 854–865.
[Huetter22]	Hütter, T., Augsten, N., Kirsch, C., Carey, M. J., & Li, C. (2022). JEDI: These aren't the JSON documents you're looking for Paper presented at ACM SIGMOD International Conference on Management of Data, Philadelphia, United States.
[Isaja18]	Mauro Isaja, John Soldatos: Distributed ledger technology for decentralization of manufacturing processes. ICPS 2018: 696-701
[Kaleido19]	J. Zhang, "Enterprise Blockchain Protocols: A Technical Analysis of Ethereum vs Fabric vs Corda", Kaleido Blogs, 24 October 2019. Available online: https://www.kaleido.io/blockchain-blog/enterprise-blockchain-protocols-a-technical-analysis-of-ethereum-vs-fabric-vs-corda (accessed April 2022).
[Kennedy17]	Z. C. Kennedy et al., "Enhanced anti-counterfeiting measures for additive manufacturing: Coupling lanthanide nanomaterial chemical signatures with blockchain technology," J. Mater. Chem. C, vol. 5, no. 37, pp. 9570–9578, 2017.
[Kim19]	H. Kim, J. Park, M. Bennis and S. Kim, "Blockchained On-Device Federated Learning," in IEEE Communications Letters. June 2019.
[Klein98]	Klein, P.N.: Computing the edit-distance between unrooted ordered trees. In: Bilardi, G., Italiano, G.F., Pietracaprina, A., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 91–102. Springer, Heidelberg (1998). doi: 10.1007/3-540-68530-8_8
[Krivosheev21]	Krivosheev, E.; Atzeni, M.; Mirylenka, K.; Scotton, P.; Miksovic, C.; and Zorin, A. 2021. Business Entity Matching with Siamese Graph Convolutional Networks. In AAAI, 16054–16056.
[Ktena17]	S. I. Ktena, S. Parisot, E. Ferrante, M. Rajchl, M. C. H. Lee, B. Glocker, and D. Rueckert. Distance metric learning using graph convolutional networks: Application to functional brain networks. CoRR, abs/1703.02161, 2017.
[Kumar18]	N. M. Kumar, "Blockchain: Enabling wide range of services in distributed energy system", Beni-Suef Univ. J. Basic Appl. Sci., vol. 7, no. 4, pp. 701-704, Dec. 2018.
[Leng18]	K. Leng, Y. Bi, L. Jing, H.-C. Fu and I. van Nieuwenhuysse, "Research on agricultural supply chain system with double chain architecture based on blockchain technology", Future Gener. Comput. Syst., vol. 86, pp. 641-649, Sep. 2018.
[Lerouge16]	J. Lerouge, Z. Abu-Aisheh, R. Raveaux, P. Héroux, and S. Adam, "Exact graph edit distance computation using a binary linear

	program”, in S+SSPR 2016, A. Robles-Kelly, M. Loog, B. Biggio, F. Escolano, and R. Wilson, Eds., ser. LNCS, vol. 10029, Cham: Springer, 2016, pp. 485–495. doi: 10.1007/978-3-319-49055-7_43.
[Lerouge17]	J. Lerouge, Z. Abu-Aisheh, R. Raveaux, P. Héroux, and S. Adam, “New binary linear programming formulation to compute the graph edit distance”, <i>Pattern Recognit.</i> , vol. 72, pp. 254–265, 2017. doi: 10.1016/j.patcog.2017.07.029.
[Li18]	Y. Li, O. Vinyals, C. Dyer, R. Pascanu, P. Battaglia, <i>Learning Deep Generative Models of Graphs (2018)</i> , arXiv preprint arXiv:1803.03324
[Li18a]	Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, and Y. Zhang, “Consortium blockchain for secure energy trading in industrial Internet of Things,” <i>IEEE Trans. Ind. Informat.</i> , vol. 14, no. 8, pp. 3690–3700, Aug. 2018.
[Li18b]	Z. Li, A. V. Barenji and G. Q. Huang, "Toward a blockchain cloud manufacturing system as a peer to peer distributed network platform", <i>Robot. Comput.-Integr. Manuf.</i> , vol. 54, pp. 133-144, Dec. 2018.
[Liang17]	Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, Laurent Njilla, “ProvChain: A Blockchain-based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability”, <i>The 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)</i> , May 14-17 2017.
[LiLiu18]	Z. Li, L. Liu, A. V. Barenji, and W. Wang, “Cloud-based manufacturing blockchain: Secure knowledge sharing for injection mould redesign,” <i>Procedia CIRP</i> , vol. 72, no. 1, pp. 961–966, 2018.
[Ma18]	G. Ma, N. K. Ahmed, T. L. Willke, D. Sengupta, M. W. Cole, N. B. Turk-Browne, and P. S. Yu. Similarity learning with higher-order proximity for brain network analysis. <i>CoRR</i> , abs/1811.02662, 2018.
[Malik18]	S. Malik, S. S. Kanhere and R. Jurdak, "ProductChain: Scalable Blockchain Framework to Support Provenance in Supply Chains," 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 2018, pp. 1-10, doi: 10.1109/NCA.2018.8548322.
[Marsico15]	M. D. Marsico, M. A. T. Figueiredo, and A. L. N. Fred, “An exact graph edit distance algorithm for solving pattern recognition problems”, in <i>ICPRAM 2015</i> , Eds., vol. 1, SciTePress, 2015, pp. 271– 278. doi: 10.5220/0005209202710278
[Mistry20]	I. Mistry et al., "Blockchain for 5G-enabled IoT for industrial automation: A systematic review solutions and challenges", <i>Mech. Syst. Signal Process.</i> , vol. 135, 2020.
[Mondragon18]	E. C. Mondragon, C. E. C. Mondragon, and E. S. Coronado, “Exploring the applicability of blockchain technology to enhance manufacturing supply chains in the composite materials industry,” in <i>2018 IEEE International Conference on Applied System Invention (ICASI)</i> , pp. 1300–1303, April 2018.
[MongoDB20]	MongoDB. 2020. White paper: MongoDB Architecture Guide: Overview. Technical Report. 12 pages. mongodb.com
[Mylrea17]	M. Mylrea and S. N. G. Gourisetti, “Blockchain for smart grid resilience: Exchanging distributed energy at speed, scale and security,” in <i>Proc. Resilience Week (RWS)</i> , 2017, pp. 18–23.
[Nakamoto08]	S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf . [Accessed April

	2022].
[Neuhaus09]	M. Neuhaus, K. Riesen, and H. Bunke, "Novel kernels for errortolerant graph classification", <i>Spatial Vis.</i> , vol. 22, no. 5, pp. 425–441, 2009.
[Noizat15]	P. Noizat et al., "Blockchain electronic vote" in <i>Handbook of Digital Currency</i> , San Diego, CA, USA:Academic, pp. 453-461, 2015.
[Pawlik15]	Pawlik, M., Augsten, N.: Efficient computation of the tree edit distance. <i>ACM Trans. Database Syst. (TODS)</i> 40(1) (2015). Article No. 3
[Pawlik16]	Mateusz Pawlik and Nikolaus Augsten. 2016. Tree edit distance: Robust and memory-efficient. <i>Information Systems</i> 56 (2016), 157–173
[Perozzi14]	B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: online learning of social representations, <i>Proceedings of KDD, ACM</i> (2014), pp. 701-710
[PostgreSQL22]	PostgreSQL Documentation. 2021. Additional Supplied Modules. https://www.postgresql.org/docs/current/pgtrgm.html . Accessed: 2022-03-20.
[Queiroz19]	M. M. Queiroz and S. F. Wamba, "Blockchain adoption challenges in supply chain: An empirical investigation of the main drivers in India and the USA", <i>Int. J. Inf. Manage.</i> , vol. 46, pp. 70-82, Jun. 2019.
[Radanović18]	I. Radanović and R. Likić, "Opportunities for use of blockchain technology in medicine", <i>Appl. Health Econ. Health Policy</i> , vol. 16, no. 5, pp. 583-590, Oct. 2018.
[Ramachandran18]	Aravind Ramachandran and Murat Kantarcioglu. 2018. SmartProvenance: A Distributed, Blockchain Based DataProvenance System. <i>Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY '18</i> . Association for Computing Machinery, New York, NY, USA, 35–42. DOI: https://doi.org/10.1145/3176258.3176333
[Riesen14]	K. Riesen, A. Fischer, and H. Bunke, "Combining bipartite graph matching and beam search for graph edit distance approximation", in <i>ANNPR 2014</i> , N. E. Gayar, F. Schwenker, and C. Suen, Eds., ser. LNCS, vol. 8774, Cham: Springer, 2014, pp. 117–128. doi: 10.1007/978-3-319-11656-3_11.
[Rivera17]	R. Rivera, J. G. Robledo, V. M. Larios and J. M. Avalos, "How digital identity on blockchain can contribute in a smart city environment", <i>Proc. Int. Smart Cities Conf. (ISC)</i> , pp. 1-4, Sep. 2017.
[Sanseverino17]	E. R. Sanseverino, M. L. Di Silvestre, P. Gallo, G. Zizzo, and M. Ippolito, "The blockchain in microgrids for transacting energy and attributing losses," in <i>Proc. IEEE Int. Conf. Internet Things (iThings), IEEE Green Comput. Commun. (GreenCom), IEEE Cyber, Phys. Social Computing</i> , 2017
[Schwarz17]	Schwarz S., Pawlik M., Augsten N. (2017) A New Perspective on the Tree Edit Distance. In: Beecks C., Borutta F., Kröger P., Seidl T. (eds) <i>Similarity Search and Applications. SISAP 2017. Lecture Notes in Computer Science</i> , vol 10609. Springer, Cham. https://doi.org/10.1007/978-3-319-68474-1_11
[Sharma18]	P. K. Sharma and J. H. Park, "Blockchain based hybrid network architecture for the smart city", <i>Future Gener. Comput. Syst.</i> , vol. 86, pp. 650-655, Sep. 2018.
[Shchur18]	O. Shchur, D. Zugner, A. Bojchevski, S. Gunnemann, Netgan:

	generating graphs via random walks, Proceedings of ICML (2018), pp. 609-61
[Shi20]	C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, J. Tang, Graphaf: a Flow-Based Autoregressive Model for Molecular Graph Generation, Proceedings of ICLR (2020)
[Soldatos19]	"The Digital Shopfloor: Industrial Automation in the Industry 4.0", John Soldatos, Oscar Lazaro, Franco Cavadini eds), River Publishers Series in Automation, Control and Robotics, Performance Analysis and Applications, ISBN: 9788770220415, e-ISBN: 9788770220408.
[Soldatos21]	J. Soldatos, N. Kefalakis, A.M. Despotopoulou, U. Bodin, A. Musumeci, A. Scandura, C. Aliprandi, D. Arabsolgar and Marcello Colledani, "A digital platform for cross-sector collaborative value networks in the circular economy", Procedia Manufacturing, Volume 54, 2021, Pages 64-69, ISSN 2351-9789, https://doi.org/10.1016/j.promfg.2021.07.011 .
[Sovrin20]	Sovrin Foundation, "Data Privacy Regulation and Distributed Ledger Technology", Whitepaper Collection, 2020.
[Stauffer17]	M. Stauffer, T. Tschachtli, A. Fischer, and K. Riesen, P. Foggia, C. Liu, and M. Vento, "A survey on applications of bipartite graph edit distance", in GBRPR 2017, Eds., ser. LNCS, vol. 10310, Cham: Springer, 2017, pp. 242–252. doi: 10.1007/978-3-319-58961-9_22.
[SwarmDocs]	Official documentation of Docker Swarm, accessed April 2022. Documents available online: https://docs.docker.com/engine/swarm/
[Szabo17]	N. Szabo, "Winning Strategies for Smart Contracts," Blockchain Research Institute Whitepapers, 2017.
[Tai79]	Tai, H.-C.: The tree-to-tree correction problem. J. ACM (JACM) 26(3), 422–433 (1979)
[Tian16]	F. Tian, "An agri-food supply chain traceability system for China based on RFID & blockchain technology," in Proc. 13th Int. Conf. Service Syst. Service Manage. (ICSSSM), 2016, pp. 1–6
[Tian17]	F. Tian, "A supply chain traceability system for food safety based on HACCP, blockchain & Internet of Things," in Proc. Int. Conf. Service Syst. Service Manage. (ICSSSM), 2017, pp. 1–6.
[Velickovic18]	P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, Proceedings of ICLR (2018)
[Vukolić15]	M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication" in International Workshop on Open Problems in Network Security, Springer, Cham, 2015, pp. 112-125.
[Wang19]	N. Wang, X. Zhou, X. Lu, Z. Guan, L. Wu, X. Du, et al., "When energy trading meets blockchain in electrical power system: The state of the art", Appl. Sci., vol. 9, no. 8, pp. 1561, 2019.
[Xu19]	X. Xu, I. Weber and M. Staples, "Architecture for Blockchain Applications", Springer, 2019. ISBN 978-3-030-03034-6.
[You18]	J. You, B. Liu, Z. Ying, V. Pande, J. Leskovec, Graph convolutional policy network for goal-directed molecular graph generation, Proceedings of NeurIPS (2018), pp. 6410-6421
[Zeng09]	Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou, "Comparing stars: On approximating graph edit distance", Proc. VLDB Endow., vol.2, no. 1, pp. 25–36, 2009. doi: 10.14778/1687627.1687631
[Zhang89]	Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance

	between trees and related problems. <i>SIAM J. Comput.</i> 18(6), 1245–1262 (1989)
[Zhao19]	Y. Zhao, K. Peng, B. Xu, Y. Liu, W. Xiong and Y. Han, "Applied engineering programs of energy blockchain in US", <i>Energy Procedia</i> , vol. 158, pp. 2787-2793, Feb. 2019.
[Zhong18]	Zhong, Zhao & Yan, Junjie & Wu, Wei & Shao, Jing & Liu, Cheng-Lin. (2018). Practical Block-Wise Neural Network Architecture Generation. 2423-2432. 10.1109/CVPR.2018.00257.
[Zhou18]	Zhou, Jie & Cui, Ganqu & Zhang, Zhengyan & Yang, Cheng & Liu, Zhiyuan & Sun, Maosong. (2018). Graph Neural Networks: A Review of Methods and Applications.
[Zoph17]	B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in 6th International Conference on Learning Representations, 2017.

Appendix A Data Model Schemata

A.1 Blockchain Data Management

Table 9 STAR Blockchain Data Management XSD schema

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="eu:star:dlsdr" targetNamespace="eu:star:dlsdr"
  elementFormDefault="qualified" vc:maxVersion="1.1" vc:minVersion="1.0"
  xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">
  <xs:group name="BlockchainNodeRegistry ">
    <xs:sequence>
      <xs:element ref="NodeIdentifier" />
    </xs:sequence>
  </xs:group>
  <xs:element name="NodeIdentifier">
    <xs:annotation>
      <xs:documentation>The Node Identifier of the Blockchain Node Registry
        service </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" name="userName" type="xs:string">
          <xs:annotation>
            <xs:documentation>An optional userName (retrieved from the identity
              service) that belongs to this node.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="organizationName" type="xs:string">
          <xs:annotation>
            <xs:documentation>Name of the company the user belongs to.
              </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="organizationSubGroup" type="xs:string">
          <xs:annotation>
            <xs:documentation>An optional Organization sub grouping (retrieved
              from the identity service) that belongs to this node (e.g.
              department, sector, branch, ... ).</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="nodeID" type="xs:string">
          <xs:annotation>
            <xs:documentation>The id name of the organization's deployed node.
              </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="OrganizationNetworkAddress">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="hostPath" type="xs:string"
                maxOccurs="1" minOccurs="1">
                <xs:annotation>
                  <xs:documentation>The URI of the deployed node
                    </xs:documentation>
                </xs:annotation>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

```

```
        </xs:annotation>
      </xs:element>
      <xs:element name="port" type="xs:int">
        <xs:annotation>
          <xs:documentation>The port of the deployed node
        </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="id" type="xs:string">
  <xs:annotation>
    <xs:documentation>Uniquely identifies the Node Identifier with an
      UUID.</xs:documentation>
  </xs:annotation>
</xs:attribute>
</xs:complexType>
</xs:element>
</xs:schema>
```