

**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

### D3.2 - Decentralized Reliability for Industrial Data and Distributed Analytics - Final version

<b>Dissemination level</b>	PU -Public
<b>Type of Document</b>	Report
<b>Contractual date of delivery</b>	31/03/2023
<b>Deliverable Leader</b>	INTRA
<b>Status - version, date</b>	Final – v1.0, 25/11/2023
<b>WP / Task responsible</b>	WP3
<b>Keywords:</b>	Blockchain, AI Algorithms Reliability, Reliability of Industrial Data, Industrial Data Processing, Data Provenance

*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

The present deliverable constitutes the evolved form and continuation of the precedent D3.1, one that discussed how distributed ledger technologies could support traceability of industrial data utilized in AI systems. The latter outlined criteria for selecting a blockchain infrastructure towards this goal and explained the choice of blockchain as a backbone, specifically the Hyperledger Fabric protocol. The document also detailed how AI data-related configurations were planned to be recorded, validated, and accessed through APIs, and concluded with plans for practical deployment and feedback integration in the context of the STAR cybersecurity toolset.

Deliverable 3.2 continues this endeavor by outlining the components of the Hyperledger Fabric network deployed in the context of the project, illustrating their purpose, placement in the STAR architecture and deployment procedure artifacts and instructions of which are shared in the project's official GitHub repository. Leveraging this proposition, other components of the STAR platform have been enabled to mine, process, and validate the reliability of metadata stemming from use cases of Artificial Intelligence (AI) on industrial data. It ought to be highlighted that the architecture proposed is an instance of the STAR reference architecture which implementation is offered as a Proof-of-Concept service; however, there are no scalability restrictions for future deployments under more complex scenarios involving additional stakeholders.



<b>Deliverable Leader:</b>	INTRA
<b>Contributors:</b>	INTRA, UPRC
<b>Reviewers:</b>	RUG, THA
<b>Approved by:</b>	Charalampos Ipektsidis (INTRA)

<b>Document History</b>			
<b>Version</b>	<b>Date</b>	<b>Contributor(s)</b>	<b>Description</b>
0.1	01/02/2023	INTRA	First document release with updated ToC
0.2	14/03/2023	INTRA	Updated background and motivation
0.3	03/04/2023	INTRA	Updated architecture placement and DLSDR specs
0.4	17/10/2023	INTRA	Updated DLSDR PoC deployment environment, Architecture
0.5	30/10/2023	INTRA	Updated DLSDR PoC deployment procedure and monitoring
0.55	06/11/2023	INTRA	Added introduction, executive summary, and conclusions
0.6	10/11/2023	UPRC	Updated algorithm configuration parameter validation
0.7	13/11/2023	INTRA	Updates throughout the document. Preparation for internal review.
0.8	16/11/2023	RUG	Internal quality review 1
0.85	21/11/2023	THA	Internal quality review 2
0.9	23/11/2023	INTRA, UPRC	Address internal reviewers' comments. Preparation of the document for submission
1.0	25/11/2023	INTRA	Final deliverable approval and submission

# Table of Contents

<b>EXECUTIVE SUMMARY .....</b>	<b>2</b>
<b>TABLE OF CONTENTS.....</b>	<b>4</b>
<b>TABLE OF FIGURES.....</b>	<b>7</b>
<b>LIST OF TABLES.....</b>	<b>9</b>
<b>DEFINITIONS, ACRONYMS AND ABBREVIATIONS .....</b>	<b>10</b>
<b>1 INTRODUCTION.....</b>	<b>12</b>
1.1 OVERVIEW AND PURPOSE.....	12
1.2 RELATIONSHIP TO OTHER DELIVERABLES.....	13
1.3 DELIVERABLE STRUCTURE .....	13
<b>2 BLOCKCHAIN: BACKGROUND AND MOTIVATION .....</b>	<b>15</b>
2.1 BACKGROUND ON BLOCKCHAIN TECHNOLOGY AND SMART CONTRACTS.....	15
2.1.1 <i>Blockchain Technology Overview</i> .....	15
2.1.2 <i>Blockchain Properties and Characteristics</i> .....	17
2.1.3 <i>Blockchain Types</i> .....	18
2.1.4 <i>Smart Contracts</i> .....	20
2.2 BLOCKCHAINS FOR INDUSTRIAL APPLICATIONS .....	22
2.2.1 <i>Rationale Behind the Use of Blockchains in Industry</i> .....	22
2.2.2 <i>Industrial Use Cases for Blockchains</i> .....	23
2.2.3 <i>Data Provenance and Reliability with Blockchains</i> .....	25
2.2.4 <i>Industrial Blockchain Projects Linked to STAR</i> .....	25
2.3 STAR BLOCKCHAIN USE CASES AND SELECTION OF BLOCKCHAIN INFRASTRUCTURE.....	26
2.3.1 <i>STAR Requirements for Blockchain Deployment and Usage</i> .....	26
2.3.2 <i>Overview of the Use of Blockchain in STAR</i> .....	27
2.3.3 <i>Non-Functional Requirements</i> .....	28
2.3.4 <i>Benchmarking and Comparative Evaluation of State-of-the-Art Blockchain Infrastructures</i> .....	29
2.3.5 <i>Scalability Considerations</i> .....	33
2.3.6 <i>Critical Analysis and Selection Outcome</i> .....	33
<b>3 RATIONALE BEHIND THE STAR DATA RELIABILITY FRAMEWORK ~ ARCHITECTURE PLACEMENT AND SPECS.....</b>	<b>36</b>
3.1 FRAMEWORK PLACEMENT WITHIN STAR ARCHITECTURE .....	36
3.2 FRAMEWORK SERVICES .....	37
3.2.1 <i>Analytics Engine Configuration (AEC) Service</i> .....	37
3.2.2 <i>Analytics Results Publishing (ARP) Service</i> .....	38
3.3 DISTRIBUTED LEDGER NODE MANAGEMENT .....	39
3.3.1 <i>Registration and Discoverability of the Platform Nodes</i> .....	39
3.3.2 <i>Data Model</i> .....	40
3.3.3 <i>API Specification</i> .....	41
3.4 DATA PROVENANCE & TRACEABILITY SERVICES .....	42
3.4.1 <i>Introduction to the Data Entities recorded on the Ledger</i> .....	42
3.4.2 <i>Data Sources Traceability</i> .....	42
3.4.3 <i>Processors Configuration Traceability</i> .....	46
3.4.4 <i>Algorithm Results Traceability</i> .....	49
3.4.5 <i>Recording Traceability Data on the Blockchain</i> .....	52
3.5 ALGORITHMS CONFIGURATION PARAMETERS VALIDATION .....	54
3.5.1 <i>How DLSDR Asserts that two Configuration Files are Identical</i> .....	54

3.5.2	<i>Proposed Approach</i> .....	56
3.5.3	<i>Quantifying the Difference between Parameter Files</i> .....	58
3.5.4	<i>Implementation of Parameter Files Validation in the STAR project</i> .....	61
<b>4</b>	<b>DATA RELIABILITY FRAMEWORK POC DEPLOYMENT ENVIRONMENT</b> .....	<b>68</b>
4.1	CLOUD HOSTING.....	68
4.2	CERTIFICATE MANAGEMENT PROTOCOLS.....	69
4.3	HARD DISK ENCRYPTION OF CLOUD MACHINES.....	70
4.4	SSH KEY-BASED AUTHENTICATION.....	71
4.5	CONTAINERIZATION OF MICROSERVICES.....	72
4.5.1	<i>Overview</i> .....	72
4.5.2	<i>Docker</i> .....	73
4.5.3	<i>Docker as the Cornerstone of STAR Fabric Network</i> .....	73
4.6	FORMING A CLUSTER AMONG DISTRIBUTED SERVICES WITH DOCKER SWARM.....	74
4.7	DOCKER SWARM OVERLAY TOPOLOGY FOR STAR PoC FABRIC NETWORK.....	76
<b>5</b>	<b>DATA RELIABILITY FRAMEWORK POC ARCHITECTURE DISSECTION AND DEPLOYMENT PROCEDURE</b> .....	<b>78</b>
5.1	INTRODUCTION.....	78
5.2	AUTHORIZERS OF DECENTRALIZED NETWORK PARTICIPANTS.....	78
5.2.1	<i>Certificate authorities in the context of Hyperledger Fabric</i> .....	78
5.2.2	<i>Certificate Authorities in the Context of STAR PoC Architecture</i> .....	80
5.2.3	<i>Configuration and Deployment of the Certificate Authorities</i> .....	81
5.2.4	<i>Verification of the Certificate Authorities installation</i> .....	85
5.3	SECURE DIGITAL COMMUNICATION AMONG NODES REPRESENTING SERVICES.....	87
5.3.1	<i>Cryptomaterial Generation in the Context of Hyperledger Fabric</i> .....	87
5.3.2	<i>Generation of Cryptomaterial for the STAR PoC Architecture</i> .....	89
5.3.3	<i>Verification of Cryptomaterial Generation</i> .....	90
5.3.4	<i>Dissemination of Identities among Nodes of the STAR PoC Network</i> .....	92
5.4	BOOTSTRAPPING THE BLOCKCHAIN.....	93
5.4.1	<i>The Concept of the Orderer Genesis Block in Hyperledger Fabric</i> .....	93
5.4.2	<i>Creating the Genesis Block for the STAR PoC Ordering Service</i> .....	93
5.5	CONFIGURATION OF PRIVATE DATA CHANNELS BETWEEN NETWORK PARTICIPANTS.....	96
5.5.1	<i>The Concept of Channels and Anchor Peers in Hyperledger Fabric</i> .....	96
5.5.2	<i>Instantiation of Channels for STAR Industrial Metadata Use Cases</i> .....	97
5.6	JOINING NODES TO THE NETWORK AND GLOBAL STATE MAINTENANCE.....	99
5.6.1	<i>Key Participants formulating a Hyperledger Fabric Network</i> .....	99
5.6.2	<i>The Concept of Ordering</i> .....	100
5.6.3	<i>Maintenance of a Common Global State in Fabric</i> .....	101
5.6.4	<i>Launching the Network for the STAR PoC Architecture</i> .....	102
5.6.5	<i>Monitoring current Global State using Fauxton</i> .....	106
5.7	ESTABLISHING THE COMMUNICATION BETWEEN STAR PLATFORM SERVICES (NODES).....	108
5.7.1	<i>Instantiation of Channels between Hyperledger Fabric Nodes</i> .....	108
5.7.2	<i>Working with the CLI Containers representing Nodes</i> .....	108
5.7.3	<i>Instantiation of Channels for STAR PoC Architecture</i> .....	111
5.7.4	<i>Join of Peer Nodes to Channel and Update of Anchor Peers</i> .....	113
5.8	RUNNING BUSINESS LOGIC THROUGH SMART CONTRACTS.....	114
5.8.1	<i>Smart Contracts in the Context of Hyperledger Fabric</i> .....	114
5.8.2	<i>Producing Chaincode for the Three Services of The PoC</i> .....	115
<b>6</b>	<b>CONTINUOUS MONITORING OF THE CLUSTER HOSTING THE DLSDR FRAMEWORK</b> <b>117</b>	
6.1	MONITORING THE SWARM NETWORK.....	117

6.1.1 *Swarmpit*..... 117

6.1.2 *Portainer*..... 118

6.2 MONITORING THE BLOCKCHAIN NETWORK ..... 120

**7 CONCLUSION AND FUTURE RESEARCH DIRECTIONS..... 122**

7.1 ACHIEVEMENTS AND KEY FINDINGS..... 122

7.2 FUTURE RESEARCH DIRECTIONS ..... 122

**REFERENCES ..... 124**

**APPENDIX A DATA MODEL SCHEMATA..... 131**

A.1 BLOCKCHAIN DATA MANAGEMENT..... 131

# Table of Figures

FIGURE 1: MCKINSEY’S ANALYSIS OF BLOCKCHAIN OPPORTUNITIES BY INDUSTRIAL SECTOR .....16

FIGURE 2: BLOCKCHAIN GOVERNANCE MODELS .....20

FIGURE 3: ILLUSTRATION OF PERFORMANCE AND SCALABILITY OF DIFFERENT FAMILIES OF POW AND BFT PROTOCOLS [VUKOLIĆ15] .....33

FIGURE 4: TOTAL UNIQUE DEVELOPERS ENGAGING IN VARIOUS BLOCKCHAIN INFRASTRUCTURES AND PROTOCOLS [CHAINSTACK21] .....34

FIGURE 5: EVOLUTION OF DEVELOPERS’ ACTIVITY FOR VARIOUS BLOCKCHAIN INFRASTRUCTURES [CHAINSTACK21] .35

FIGURE 6 STAR FUNCTIONAL MODULES AND LOGICAL VIEW OF THE ARCHITECTURE [D2.6].....36

FIGURE 7 STAR SECURITY AND DATA GOVERNANCE FOR AI SYSTEMS IN MANUFACTURING LOGICAL VIEW .....37

FIGURE 8: ANALYTICS ENGINE CONFIGURATION (AEC) SERVICE OVERVIEW .....38

FIGURE 9: ANALYTICS RESULTS PUBLISHING (ARP) SERVICE OVERVIEW .....38

FIGURE 10: ORGANIZATION NODE IDENTIFIER ENTITY GRAPH .....40

FIGURE 11 DATA SOURCE PERSISTENCE TO DISTRIBUTED LEDGER NETWORK .....45

FIGURE 12 PROCESSOR PERSISTENCE TO DISTRIBUTED LEDGER NETWORK .....49

FIGURE 13 OBSERVATION ENTITY OF THE DATA MODEL .....51

FIGURE 14: METADATA PERSISTENCE USING THE STAR BLOCKCHAIN .....53

FIGURE 15: METADATA VALIDATION USING THE STAR BLOCKCHAIN .....54

FIGURE 16 FLOW FOR GENERAL EDIT DISTANCE COMPUTATION .....57

FIGURE 17 TRAINING FLOW FOR AI-BASED CNN-SPECIFIC EDIT DISTANCE COMPUTATION .....57

FIGURE 18 JSON DIFF – SAME CONTENTS BUT DISTANCE CALCULATION IS NOT STRAIGHTFORWARD [HUETTER22]..58

FIGURE 19 S-GCNN ARCHITECTURE FROM [KIROSHEEV21] .....60

FIGURE 20 NETWORK ARCHITECTURES GENERATED BY VARIANTS OF BLOCK-QNN [ZHONG18] .....61

FIGURE 21: HEATMAP OF DISTANCES BETWEEN CONFIGURATION OF MODELS .....64

FIGURE 22: STAR INFRASTRUCTURE ADMINISTRATOR DASHBOARD ON HETZNER CLOUD .....68

FIGURE 23: HOW TLS 1.3 HANDSHAKE WORKS .....70

FIGURE 24: BASIC PRINCIPLE OF SSH AUTHENTICATION HANDSHAKE .....72

FIGURE 25: A DOCKER SWARM CLUSTER EXAMPLE [SWARMDOCS].....75

FIGURE 26: SWARM SECURITY CONCEPT WITH PKI [SWARMDOCS].....75

FIGURE 27: DOCKER SWARM OVERLAY TOPOLOGY FOR STAR MVP FABRIC NETWORK.....76

FIGURE 28: CERTIFICATE AUTHORITIES IN THE CONTEXT OF THE STAR PoC FABRIC NETWORK (AS RED DOCKER CONTAINERS) .....81

FIGURE 29: MATERIAL FOR CA DEPLOYMENT IN THE INSTALLATION KIT .....83

FIGURE 30: DEFINITION OF A CA FOR ORGANIZATION-ZERO.....84

FIGURE 31: THE RESULT OF “DOCKER STACK PS HYPERLEDGER\_FABRIC” COMMAND.....85

FIGURE 32: THE RESULT OF “DOCKER STACK SERVICES HYPERLEDGER\_FABRIC” COMMAND .....85

FIGURE 33: GENERATED CERTIFICATES AFTER CA INSTALLATION FOR ORGANIZATION ZERO.....86

FIGURE 34: A SUCCESSFUL EXECUTION OF THE FUNCTIONS ENCLOSED IN THE REGISTERENROLL.SH SCRIPT .....90

FIGURE 35: RESULT OF CERTIFICATES CREATION SCRIPT ON “MANAGER” NODE.....92

FIGURE 36: CONFIGURATION FOR THE GENESIS BLOCK FROM CONFIGTX.YAML .....95

FIGURE 37: SUCCESSFUL CREATION OF THE GENESIS BLOCK IN THE CONSOLE .....95

FIGURE 38: GENESIS BLOCK WITHIN THE MANAGER NODE FOLDER STRUCTURE .....95

FIGURE 39: CONFIGURATION FOR THE CHANNEL FROM CONFIGTX.YAML .....97

FIGURE 40: “MYCHANNEL” ON LINE 10 MUST BE CHANGED WITH ACTUAL CHANNEL NAME.....98

FIGURE 41: SUCCESSFUL CREATION OF A CHANNEL IN THE CONSOLE .....98

FIGURE 42: CHANNEL ARTIFACTS WITHIN THE MANAGER NODE FOLDER STRUCTURE.....99

FIGURE 43: INTERFACING TO THE HLF ORDERING SERVICE [FABRICDOCS] .....101

FIGURE 44: DISTRIBUTION OF DOCKER CONTAINERS IN VMS .....103

FIGURE 45: THE RESULT OF STACK DEPLOYMENT THROUGH THE CONSOLE OF THE “MANAGER” .....104

FIGURE 46: THE RESULT OF “DOCKER STACK PS HYPERLEDGER\_FABRIC” COMMAND.....104

FIGURE 47: THE RESULT OF “DOCKER STACK SERVICES HYPERLEDGER\_FABRIC” COMMAND .....105

FIGURE 48: THE DEPLOYED SERVICES OF OUR STACK AS MONITORED VIA PORTAINER .....105

FIGURE 49: THE DISTRIBUTION OF RUNNING SERVICES AMONG NODES AS VISUALISED IN PORTAINER ..... 106

FIGURE 50: REQUEST FOR ADMINISTRATOR CREDENTIALS UPON ACCESSING THE FAUXTON DASHBOARD ..... 107

FIGURE 51: THE DATABASES CREATED BY DEFAULT AS SEEN THROUGH FAUXTON..... 107

FIGURE 52: THE RESULT OF STACK DEPLOYMENT THROUGH THE CONSOLE OF THE "MANAGER" ..... 109

FIGURE 53: THE DISTRIBUTION OF DOCKER CONTAINERS AFTER THE DEPLOYMENT OF CLIS (DEPICTED IN LIGHT GREEN)  
..... 109

FIGURE 54: ACCESSING THE CONSOLE OF THE CLI CORRESPONDING TO ORGANIZATION ZERO VIA PORTAINER ..... 110

FIGURE 55: THE BUTTON TO PRESS SO AS TO ENABLE THE CONSOLE ..... 111

FIGURE 56: THE .BLOCK FILES CREATED AFTER CHANNELS INSTANTIATION ..... 112

FIGURE 57: CODE SNIPPET OF CHAINCODE (SMART CONTRACT) CORRESPONDING TO "DATA SOURCES TRACEABILITY"  
USE CASE..... 116

FIGURE 58: SWARMPIT MASTER DASHBOARD FOR RESOURCES MONITORING ..... 117

FIGURE 59: SWARMPIT DEPICTING DEPLOYED DOCKER SWARM STACKS ..... 118

FIGURE 60: SWARMPIT DEPICTING THE VIRTUAL MACHINES FORMULATING A NETWORK ..... 118

FIGURE 61: PORTAINER LANDING PAGE FOR CONTAINERS ADMINISTRATION ..... 119

FIGURE 62: PORTAINER SWARM CLUSTER OVERVIEW ..... 119

FIGURE 63: PORTAINER SWARM VISUALIZER WITH NODES ASSIGNED TO STAR SERVICES ..... 120

FIGURE 64: HYPERLEDGER EXPLORER DASHBOARD ..... 121

## List of Tables

TABLE 1: BLOCKCHAIN SUITABILITY ASSESSMENT VS STAR REQUIREMENTS.....	26
TABLE 2: NON-FUNCTIONAL PROPERTIES OF THE STAR BLOCKCHAIN .....	28
TABLE 3: COMPARATIVE ASSESSMENT OF CHARACTERISTICS OF POPULAR PERMISSIONED ENTERPRISE BLOCKCHAIN INFRASTRUCTURES [KALEIDO19].....	30
TABLE 4: REGISTRATION AND DISCOVERABILITY SERVICES API SPECIFICATION OVERVIEW.....	41
TABLE 5 DLSDR FOR DATA SOURCE METADATA API SPECIFICATION OVERVIEW .....	44
TABLE 6 DLSDR FOR PROCESSOR CONFIGURATION METADATA API SPECIFICATION OVERVIEW .....	47
TABLE 7 DLSDR FOR ALGORITHM RESULTS API SPECIFICATION OVERVIEW .....	51
TABLE 8 CNN CONFIGURATION INFORMATION CLASSIFIER EXAMPLE .....	55
TABLE 9: EXAMPLE OF INPUT CONFIGURATIONS .....	62
TABLE 10: CA INFORMATION PER VIRTUAL MACHINE .....	84
TABLE 11: STAR BLOCKCHAIN DATA MANAGEMENT XSD SCHEMA .....	131

## Definitions, Acronyms and Abbreviations

Acronym/Abbreviation	Title
<b>API</b>	Application Programming Interface
<b>ARP</b>	Analytics Results Publishing
<b>BFT</b>	Byzantine Fault Tolerance (or Tolerant)
<b>BTC</b>	Bitcoin
<b>CA</b>	Certificate Authority
<b>CE</b>	Circular Economy
<b>CFT</b>	Crash Fault Tolerance (or Tolerant)
<b>CLI</b>	Command-Line Interface
<b>CRL</b>	Certificate Revocation List
<b>CRUD</b>	Create Read Update Delete
<b>dApp</b>	Distributed Application
<b>DLSDR</b>	Distributed Ledger Services for Data Reliability
<b>DLT</b>	Distributed Ledger Technology
<b>DoA</b>	Description of Action
<b>DPT</b>	Data Provenance and Traceability
<b>DSL</b>	Domain-Specific Language
<b>EAE</b>	Edge Analytics Engine
<b>EDM</b>	Ecosystem Data Manager
<b>ERC</b>	Ethereum Request for Comments
<b>GNN</b>	Graph Neural Network
<b>GUI</b>	Graphical User Interface
<b>HA</b>	High Availability
<b>HLF</b>	Hyperledger Fabric
<b>JSON</b>	JavaScript Object Notation
<b>JVM</b>	Java Virtual Machine
<b>MitM</b>	Man-in-the-Middle
<b>MSP</b>	Membership Service Provider
<b>MVP</b>	Minimum Viable Product
<b>NAS</b>	Neural Arcitecture Search
<b>OEM</b>	Original Equipment Manufacturer
<b>P2P</b>	Peer-to-Peer
<b>PKCS</b>	Public Key Cryptography Standards
<b>PKI</b>	Public Key Infrastructure
<b>PoC</b>	Proof-of-Concept
<b>PoS</b>	Proof of Stake
<b>PoW</b>	Proof of Work
<b>RBAC</b>	Role Based Access Control
<b>RMS</b>	Runtime Monitoring System
<b>RSA</b>	Rivest–Shamir–Adleman cryptosystem
<b>SC</b>	Smart Contract
<b>SDK</b>	Software Development Kit
<b>SLA</b>	Service Level Agreement

<b>SSL</b>	Secure Sockets Layer
<b>TED</b>	Tree Edit Distance
<b>TLS</b>	Transport Layer Security
<b>TTP</b>	Trusted Third Party
<b>UBAC</b>	User Based Access Control
<b>URI</b>	Universal Resource Identifier
<b>URL</b>	Universal Resource Locator
<b>UUID</b>	Universally Unique Identifier
<b>XACML</b>	eXtensible Access Control Markup Language
<b>XSD</b>	XML Schema Definition

# 1 Introduction

## 1.1 Overview and Purpose

This current deliverable represents the final iteration and extension of its predecessor, D3.1. In D3.1, we delved into the potential applications of distributed ledger technologies in enhancing the traceability of industrial data used within AI systems. The previous document set out specific criteria for the selection of a blockchain infrastructure for achieving this objective and elucidated our decision to employ a permissioned option, the Hyperledger Fabric protocol as the foundational framework for developing a Proof-of-Concept for STAR's proposed Data Reliability Framework. Furthermore, we provided comprehensive insights into how data configurations related to AI were slated to be recorded, validated, and accessed by other platform components through the use of APIs. The document concluded with a roadmap for the practical deployment of these innovations and their integration into the broader context of the STAR cybersecurity toolset, incorporating mechanisms for feedback and refinement.

Building upon the foundational principles and strategies outlined in D3.1, this evolved deliverable takes us further on our journey to harness the potential of distributed ledger technologies within the realm of industrial data traceability for AI systems. In the subsequent sections, we delve deeper into the technical intricacies of our approach, providing a detailed implementation and deployment description. We also shed light on the synergy between blockchain and AI, emphasizing the robust security and transparency that this combination brings to the table. Our expanded discussions include a more comprehensive analysis of the key components, processes, and technologies involved in recording, validating, and granting access to AI industrial data configurations through APIs.

In this context, in addition to the anti-tampering properties of the blockchain, we adopted and discuss in the present deliverable a dual-pronged approach. This has been performed to address the challenge of ensuring the integrity of configuration files employed in AI-powered components of the STAR platform. The first part focuses on measuring dissimilarity between configuration files in a semi-structured, hierarchical format like JSON, while the second part delves into reconstructing convolutional neural networks as graphs for more accurate representation and analysis. Diagrams illustrating these two approaches are provided for clarity.

Moreover, we place a strong emphasis on the practical aspects of the cluster of blockchain nodes deployment. We elucidate the steps involved in bringing our vision to life, outlining specific milestones. The integration of our advancements into the STAR cybersecurity toolset is a key focal point, as it plays a pivotal role in enhancing the resilience of industrial systems against cyber threats (e.g., data tampering). We delve into the strategies for seamless integration, scalability, and how feedback loops through monitoring tools for our infrastructure could be leveraged to continuously observe, refine and optimize our solution. Ultimately, this deliverable serves as an invaluable resource for stakeholders, offering a comprehensive blueprint for the successful implementation of future distributed ledger-based solutions to bolster industrial data traceability within the dynamic landscape of AI systems.

This deliverable represents the culmination of extensive work conducted within the context of Task 3.1, which is centred around establishing a decentralized infrastructure to track the provenance of industrial data used in AI systems. This task is specifically geared towards handling a diverse range of industrial data sources, including data generated by sensors, industrial automation devices, robots, and business information systems. Its core objectives

involve the design and implementation of decentralized consensus mechanisms, aimed at not only validating the integrity and reliability of the source data but also ensuring the trustworthiness of the distributed data analytics algorithms used in their processing. This approach is further bolstered by optimized performance tailored to the unique demands of industrial deployments. The infrastructure developed within this task serves as a linchpin for enhancing the reliability and trustworthiness of industrial data applied in distributed data analytics and distributed AI mechanisms.

## 1.2 Relationship to other deliverables

Apart from being an advanced iteration and extension of D3.1 “Decentralized Reliability for Industrial Data and Distributed Analytics-Initial version”, 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 D2.10 “Report on Co-Design Workshops and Focus Groups-Final version”. At the same time, it considers the initial specifications of the STAR data models and architecture which are part of deliverables D2.5 “Data Models and Data Collection-Final version”, D2.7 “STAR Reference Architecture and Blueprints-Final Version” respectively.

Moreover, the deliverable is closely related to D3.6 (“Security and Data Governance Infrastructure-Final version”), which is destined to specify the usage of the data reliability framework as part of the security and data governance solution.

In addition, the contents of this research are leveraged in D3.4 “Cyber-Defence Mechanisms against Poisoning and Evasion Attacks-Final version” as a service for managing Poisoning and Evasion Attacks algorithms configuration and results data.

Finally, as expected for all technological propositions developed and evolved in the context of STAR project, the Decentralized Reliability Framework presented here has been validated in the context of the pilot sites participating in the consortium as stakeholders. Therefore, the results and conclusions of said validations appear in their respective deliverables of WP6.

## 1.3 Deliverable Structure

Before closing this introductory chapter, we present an outline of how this document is organized and structured to provide a clear and logical flow of information. This section offers readers a glimpse into the document's layout and what to expect in the chapters that follow.

- **Chapter 2** is a revisit to content also present in D3.1. It delves into the potential applications of distributed ledger technologies in enhancing the traceability of industrial data used within AI systems. It sets out specific criteria for the selection of a blockchain infrastructure for achieving this objective and elucidated our decision to employ the Hyperledger Fabric protocol as the foundational framework for developing a Proof-of-Concept (PoC) for STAR's proposed Data Reliability Framework.
- **Chapter 3** is mostly theoretical and similarly contains information already discussed in D3.1. It explores the purpose of the services of the industrial Data Reliability Framework proposed by STAR. It provides an introduction to the Data Entities recorded on the Ledger, the specifications of the APIs that may be used by other services to interact with them and perform data transactions, as well as the scientific methods used to address the challenge of ensuring the integrity of configuration files employed

in AI-powered components of the STAR platform. The newly introduced Section 3.5.4 illustrates how those methods have been validated in the specific context of STAR pilots.

- **Chapter 4** is where content that has not been discussed before begins. This chapter dissects the deployment environment of the Data Reliability Framework PoC. It delves into the technologies asserting the security of the data-at-rest and of the data-in-transit. Then, it discusses containerization of the Framework's components to ensure interoperability with all mainstream operating systems and their possible individual configurations. Finally, it discusses microservices Orchestration and the formulation of a single cluster hosting the PoC blockchain network.
- **Chapter 5**, the most extended in size, provides detailed insights into the intricacies of the actual component's architecture. It elucidates the deployment procedures, and offer a comprehensive examination of how Hyperledger Fabric's capabilities empower the wider platform's cybersecurity technological solutions to catalyse transformative change within the domains of security, trust, and traceability.
- **Chapter 6** proposes a set of third-party open-source tools, addressed to system maintainers, for monitoring the performance and health of deployments such as our blockchain network.
- **Chapter 7** concludes the present deliverable by highlighting important points and proposing future research directions.

## 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.

The evolution of ledger systems in response to modern business requirements has been remarkable. As business landscapes become increasingly complex, the need for secure, tamper-proof, and resilient record-keeping has grown more pronounced. Traditional ledger systems, even when computerized, sometimes fall short in ensuring data integrity. In contrast, the demand for trustworthy record-keeping, especially in complex, interwoven business ecosystems, continues to rise. These systems need to provide not only security against tampering but also an unwavering ability to recover from unforeseen failures, ensuring the uninterrupted flow of transactions. Furthermore, the need for transparency remains critical, as partners and stakeholders seek access to irrefutable records, fostering trust and collaboration in a highly interconnected global business environment. The ongoing quest to meet these evolving business requirements is driving innovation in ledger technology, promising more robust and reliable solutions in the years to come.

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).

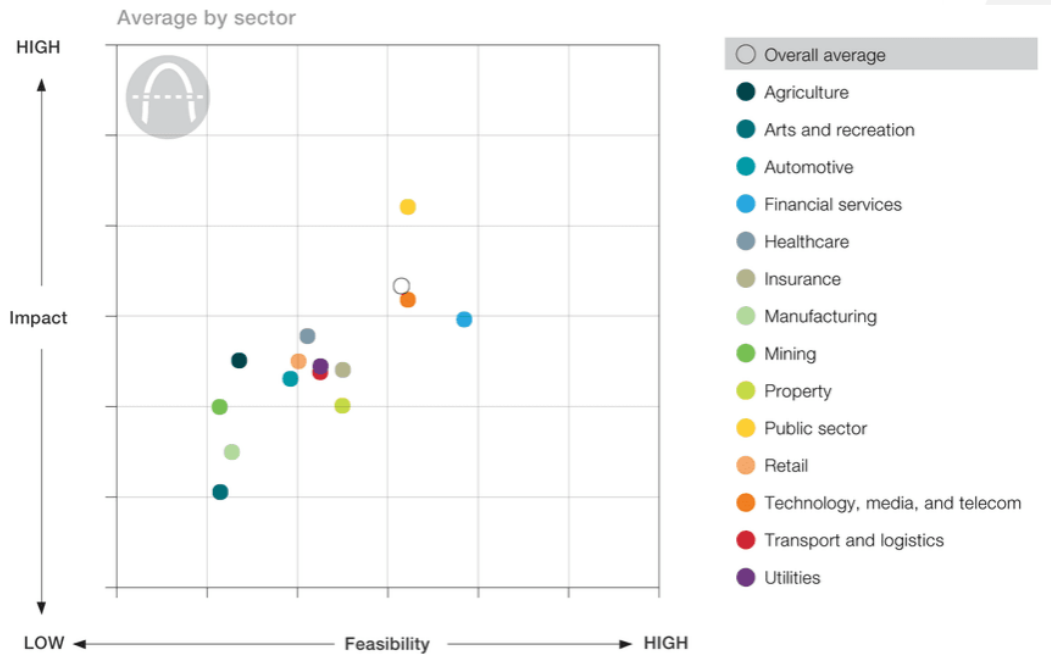


Figure 1: McKinsey's Analysis of Blockchain Opportunities by Industrial Sector<sup>1</sup>

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<sup>2</sup>, 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

<sup>1</sup> Source: "Blockchain opportunities by industrial sector" by Brant Carson, Giulio Romanelli, Patricia Walsh, and Askhat Zhumaev, McKinsey.com, June 19, 2018. Accessed October 2023: <https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/blockchain-beyond-the-hype-what-is-the-strategic-business-value>

<sup>2</sup> Official bitcoin website: <https://bitcoin.org/en/> (Accessed October 2023).

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 cryptocurrencies systems such as Ethereum<sup>3</sup>), 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<sup>4</sup>. A digital asset is anything digital that has value, establishes ownership, and is discoverable. Digital assets now encompass everything, from words to fractionalized ownership in a corporation or real estate through tokenization. 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

<sup>3</sup> Official Ethereum website: <https://ethereum.org/en/> (Accessed October 2023).

<sup>4</sup> Definition of "Digital Asset" from Investopedia: <https://www.investopedia.com/terms/d/digital-asset-framework.asp> (Accessed October 2023).

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

- 1<sup>st</sup> 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.
- 2<sup>nd</sup> 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.
- 3<sup>rd</sup> 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 notional 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 (e.g., [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 sectors<sup>5</sup>. The STAR partners participate or have participate in blockchain projects in the areas of manufacturing

---

<sup>5</sup> <https://ec.europa.eu/digital-single-market/en/news/eu-funded-projects-blockchain-technology> (accessed April 2022)

and supply chain management, which are thematically most relevant to STAR. H2020 FAR-EDGE<sup>6</sup> 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 QU4LITY<sup>7</sup> 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 DIGIPRIME<sup>8</sup> is another project where a blockchain is employed to ensure security and traceability of data sharing between partners on a Circular Economy consortium [Soldatos21]. 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).

*Table 1: Blockchain Suitability Assessment vs STAR Requirements.*

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).

<sup>6</sup> H2020 FAR-EDGE at CORDIS: <https://cordis.europa.eu/project/id/723094> (accessed April 2022)

<sup>7</sup> H2020 QU4LITY at CORDIS: <https://cordis.europa.eu/project/id/825030> (accessed April 2022)

<sup>8</sup> H2020 DigiPrime at CORDIS: <https://cordis.europa.eu/project/id/873111> (accessed April 2022)

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 Description of Action (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.

Essentially, 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<sup>9</sup> (originally by IBM), Enterprise Ethereum<sup>10</sup> and R3 Corda<sup>11</sup>. 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 Consensus Quorum<sup>12</sup> (originally by JPMorgan Chase modified from go-ethereum), Hyperledger Besu<sup>13</sup> (originally by Pegasys newly implemented in Java), Autonomy<sup>14</sup> (by Clearmatics modified from go-ethereum) and Strato<sup>15</sup> (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):

*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.

<sup>9</sup> Hyperledger Fabric official webpage: <https://www.hyperledger.org/use/fabric> (accessed April 2022)

<sup>10</sup> Enterprise Ethereum Alliance official website: <https://entethalliance.org> (accessed April 2022)

<sup>11</sup> Corda official website: <https://www.corda.net/> (accessed April 2022)

<sup>12</sup> Consensus Quorum official website: <https://consensus.net/quorum/> (accessed April 2022)

<sup>13</sup> Hyperledger Besu official website: <https://www.hyperledger.org/use/besu> (accessed April 2022)

<sup>14</sup> Autonomy official website: <https://autonity.io/> (accessed April 2022)

<sup>15</sup> Strato official website: <https://blockapps.net/> (accessed April 2022)

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 deterministic leader rotation). Tendermint	Kafka/Zab (CFT with trusted leader). Raft (CFT with trusted leader).	Raft (CFT with trusted leader) BFT
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 node runtimes (min or 3 or 4 depending on consensus).	Native feature (channels) with shared peer runtime, and shared orderer. Built-in governance for creating side-chains with isolated state.	No concept of a chain (shared ledger). 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 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 3, we can examine how the performance is affected by the increase in the number of nodes in the network.

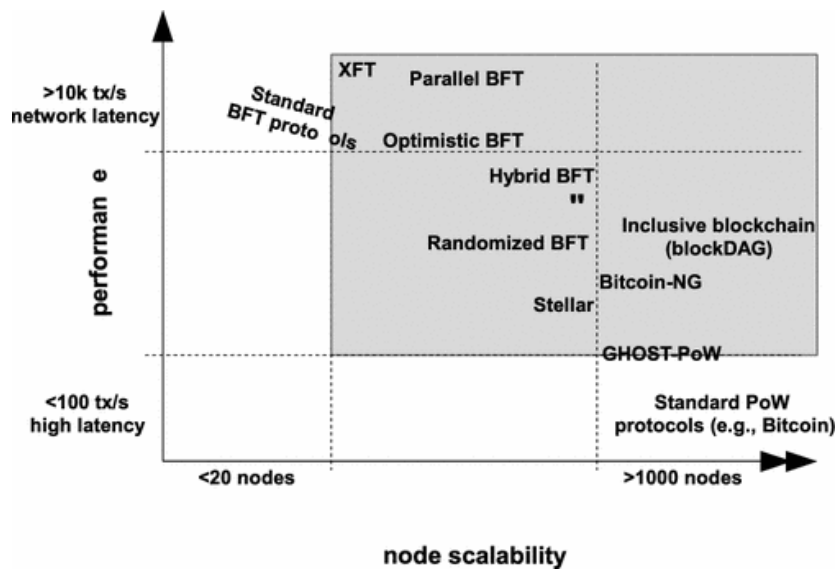


Figure 3: Illustration of Performance and Scalability of Different Families of PoW and BFT protocols [Vukolić15]

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.

### 2.3.6 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 4 and Figure 5 below.

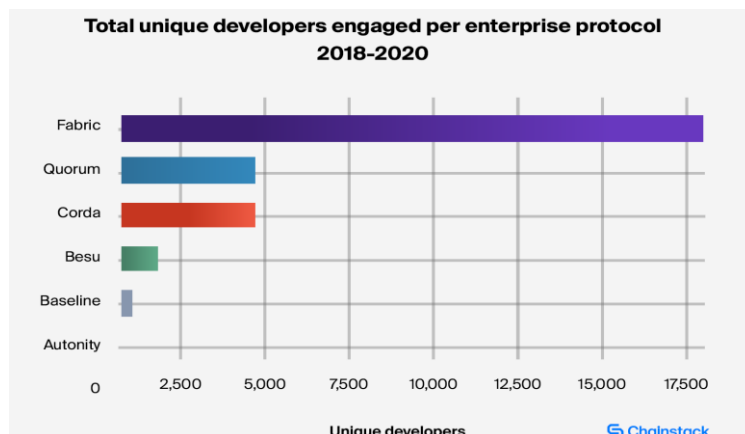


Figure 4: Total Unique Developers Engaging in various Blockchain Infrastructures and Protocols [Chainstack21]

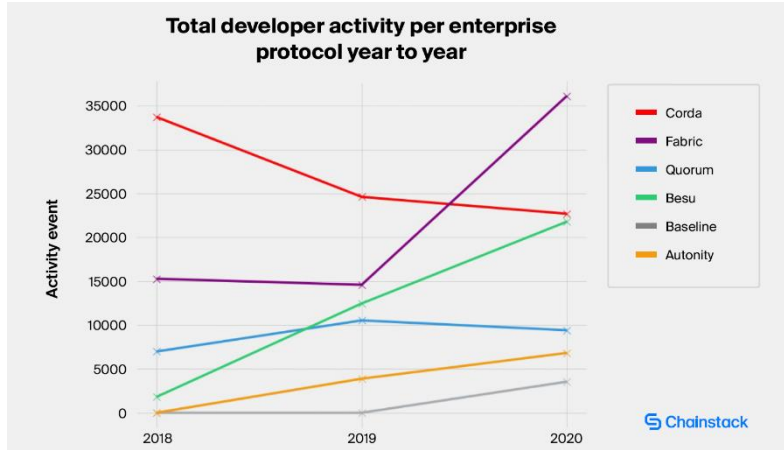


Figure 5: Evolution of Developers' Activity for various blockchain infrastructures [ChainStack21]

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

### 3 Rationale behind the STAR Data Reliability Framework ~ Architecture Placement and Specs

#### 3.1 Framework Placement within STAR Architecture

The Reference Architecture of the STAR platform, as illustrated in Figure 6 below and documented in deliverables D2.6 and D2.7, is consisted of three main domains:

- Cybersecurity Domain:** 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. The functionalities of these domains support and reinforce the trustworthiness of the project’s functions in the other two domains.
- (Trusted) Human Robot Collaboration Domain:** Provides functionalities for the trusted collaboration between human and robots. Leverages cybersecurity functionalities, while being used to reinforce functionalities in the safety domain as well.
- Safety Domain:** Ensures the safety of industrial operations, including operations that involve workers and/or automation systems. For instance, functionalities in this domain reinforce worker safety, while catering for the safe operation of AMRs in industrial sites.

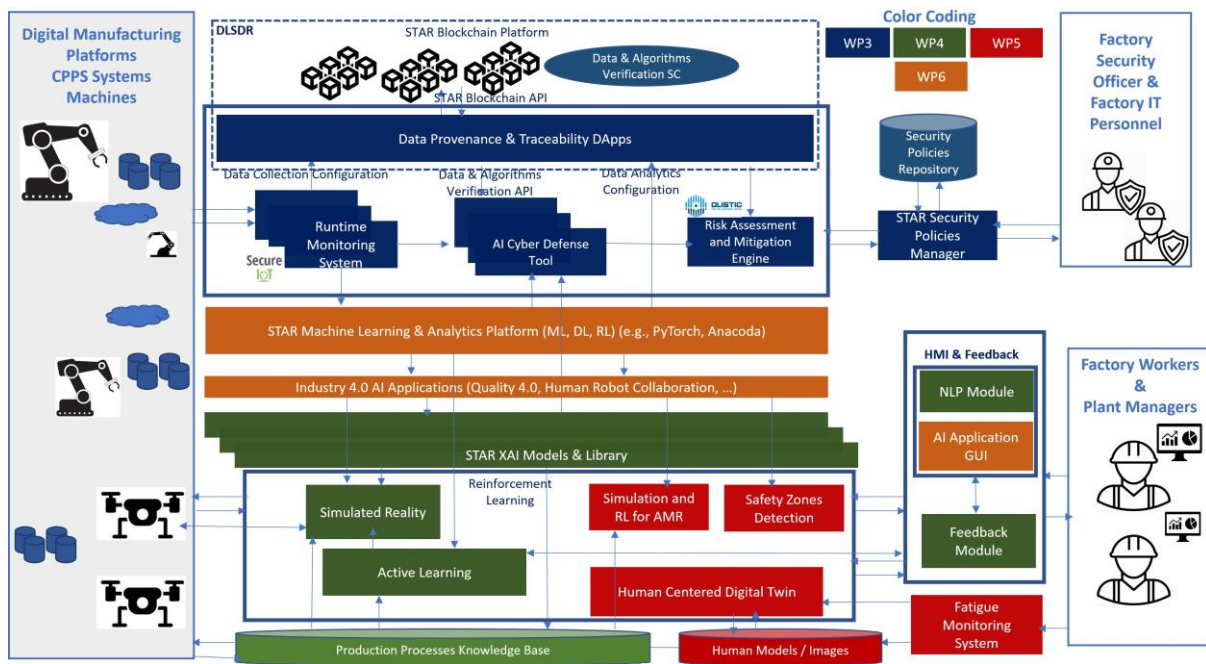


Figure 6 STAR Functional Modules and Logical View of the Architecture [D2.6]

The **STAR Distributed Ledger Services for Data Reliability (DLSDR)** (depicted in Figure 7 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.

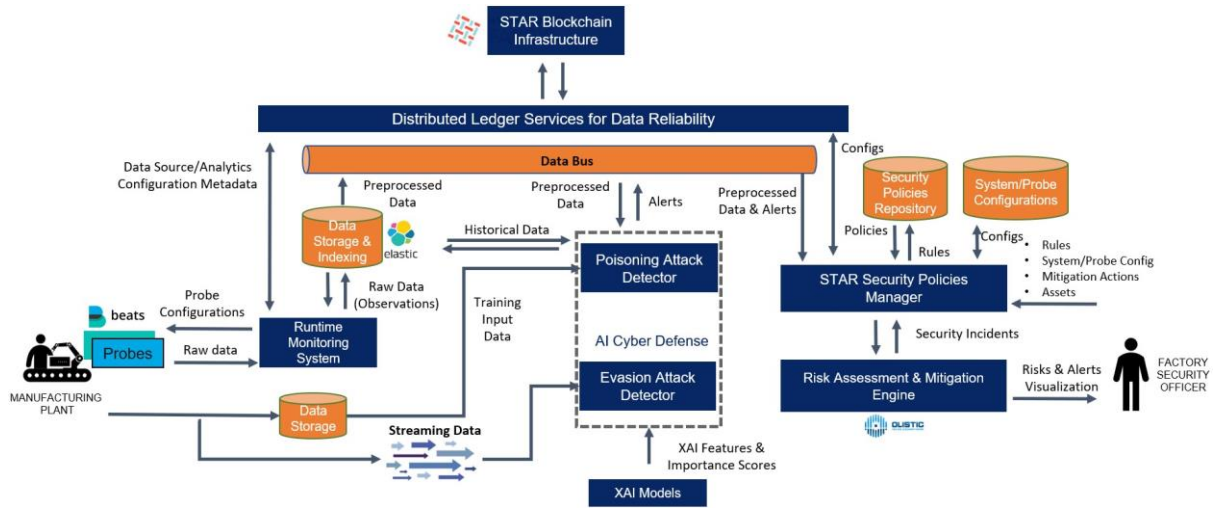


Figure 7 STAR Security and Data Governance for AI Systems in Manufacturing Logical View

DLSDR (Figure 7 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
- The STAR Security Policies Manager which is using DLSDR for AI algorithms results validation.

### 3.2 Framework Services

The **STAR Distributed Ledger Services for Data Reliability** (DLSDR) framework provides the following services that will empower reliable industrial data and trusted algorithms configurations for industrial quality control:

#### 3.2.1 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.

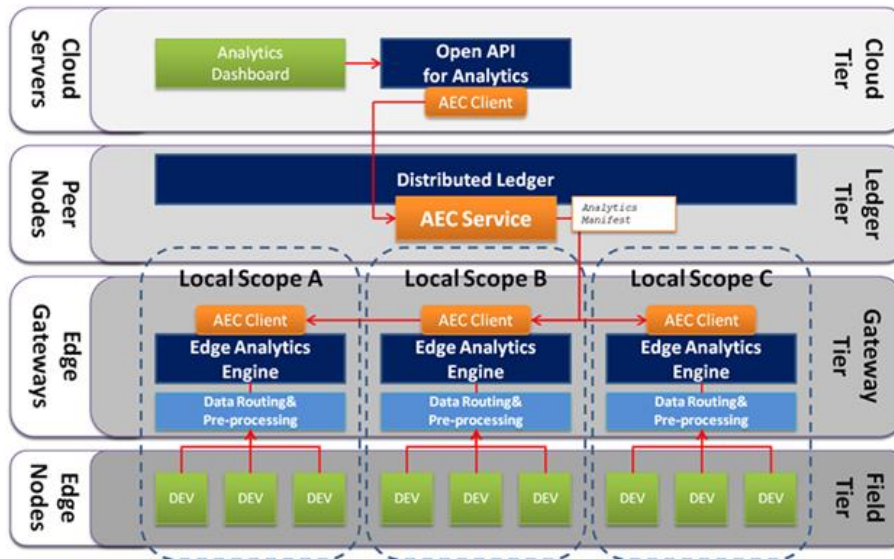


Figure 8: Analytics Engine Configuration (AEC) Service overview

### 3.2.2 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.

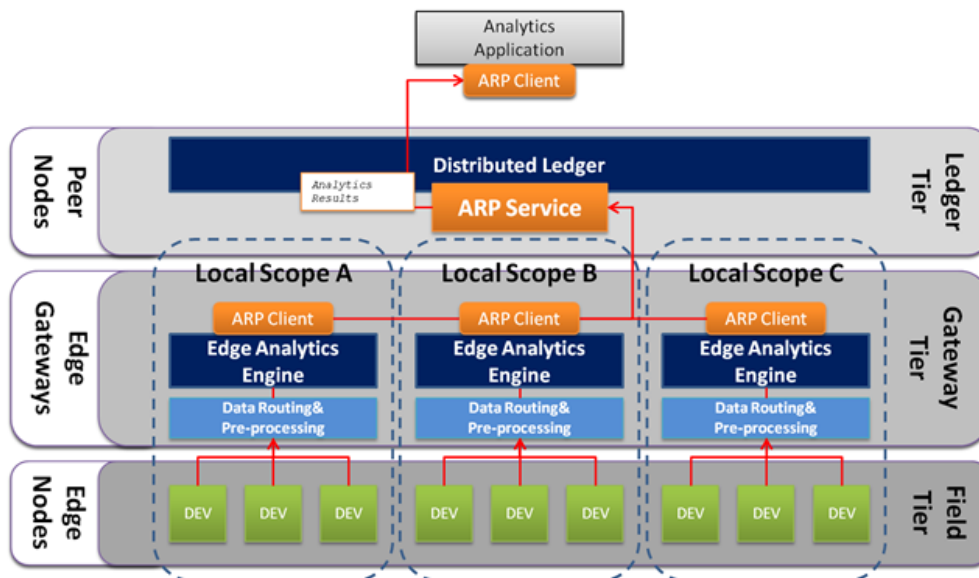


Figure 9: Analytics Results Publishing (ARP) Service overview

### 3.3 Distributed Ledger Node Management

#### 3.3.1 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.

Let us assume that a user of a STAR service uses Keycloak to authenticate and thus receives 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 within 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"
}
```

### 3.3.2 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 10. 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 4 of Appendix A below.

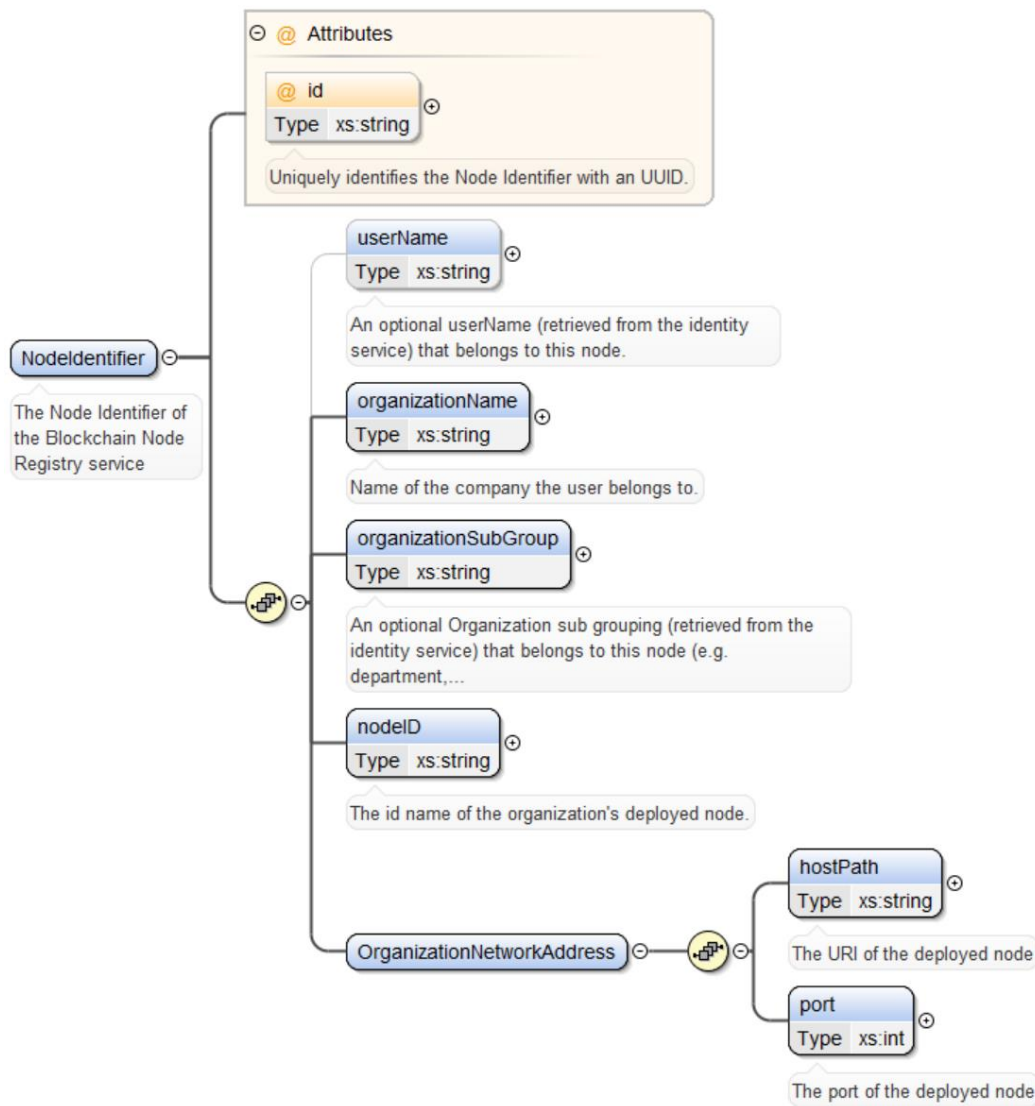


Figure 10: Organization Node Identifier entity graph

As depicted in Figure 10 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

### 3.3.3 API Specification

Table 4 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 4: 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 criterion provided in an Node Identifier instance.

## 3.4 Data Provenance & Traceability Services

### 3.4.1 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 environments<sup>16</sup>. 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.

### 3.4.2 Data Sources Traceability

#### 3.4.2.1 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.

##### 3.4.2.1.1 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 any schema and it has the unique characteristic that it can function as a complex or a

---

<sup>16</sup> <https://github.com/far-edge/digital-models>

simple type definition, according to context.

#### 3.4.2.1.2 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.

#### 3.4.2.1.3 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.

### 3.4.2.2 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).

#### 3.4.2.2.1 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.

### 3.4.2.3 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 5 below.

*Table 5 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/:id	dsmID: String	No	Used to flag the 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.

### 3.4.2.4 Data Sources persistence interaction

Figure 11 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.

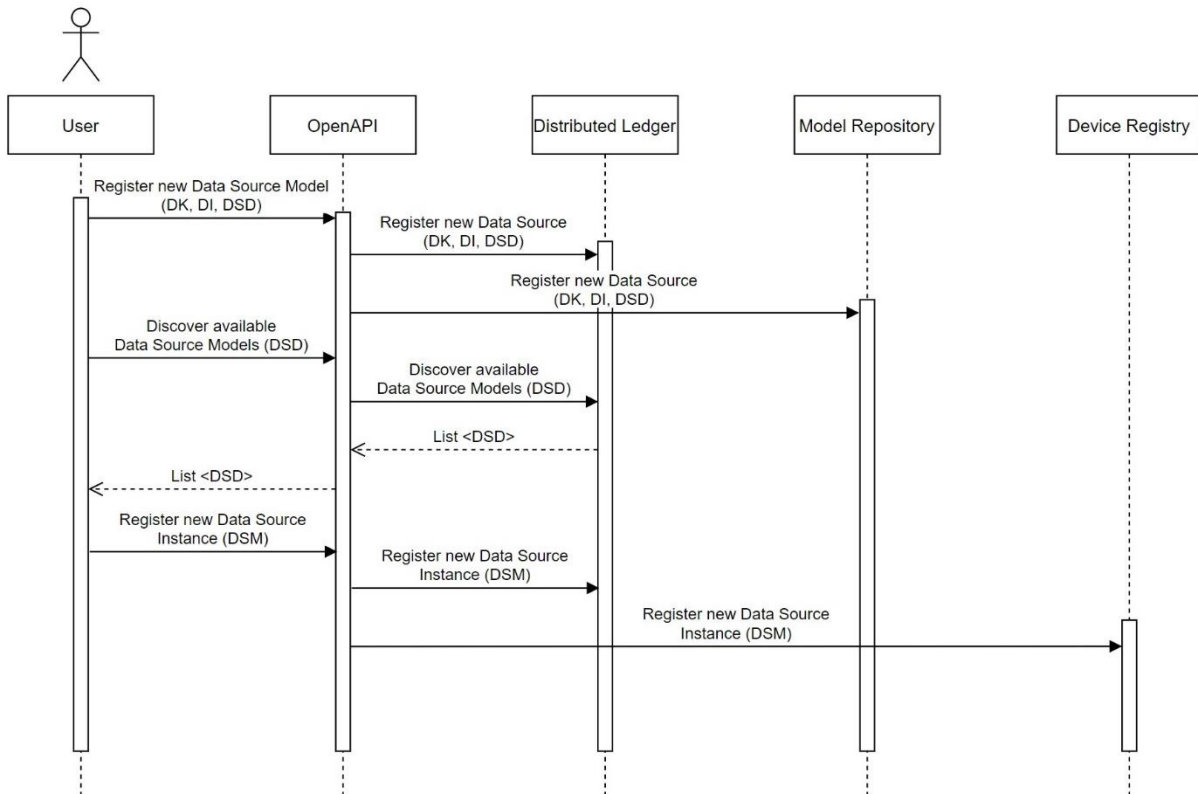


Figure 11 Data Source persistence to Distributed Ledger network

As we can see in Figure 11 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.

### 3.4.3 Processors Configuration Traceability

#### 3.4.3.1 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.

##### 3.4.3.1.1 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.

##### 3.4.3.1.2 Processor 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.

### 3.4.3.2 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).

#### 3.4.3.2.1 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)

### 3.4.3.3 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 6 below.

*Table 6 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.

### 3.4.3.4 Data Sources persistence interaction

Figure 12 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.

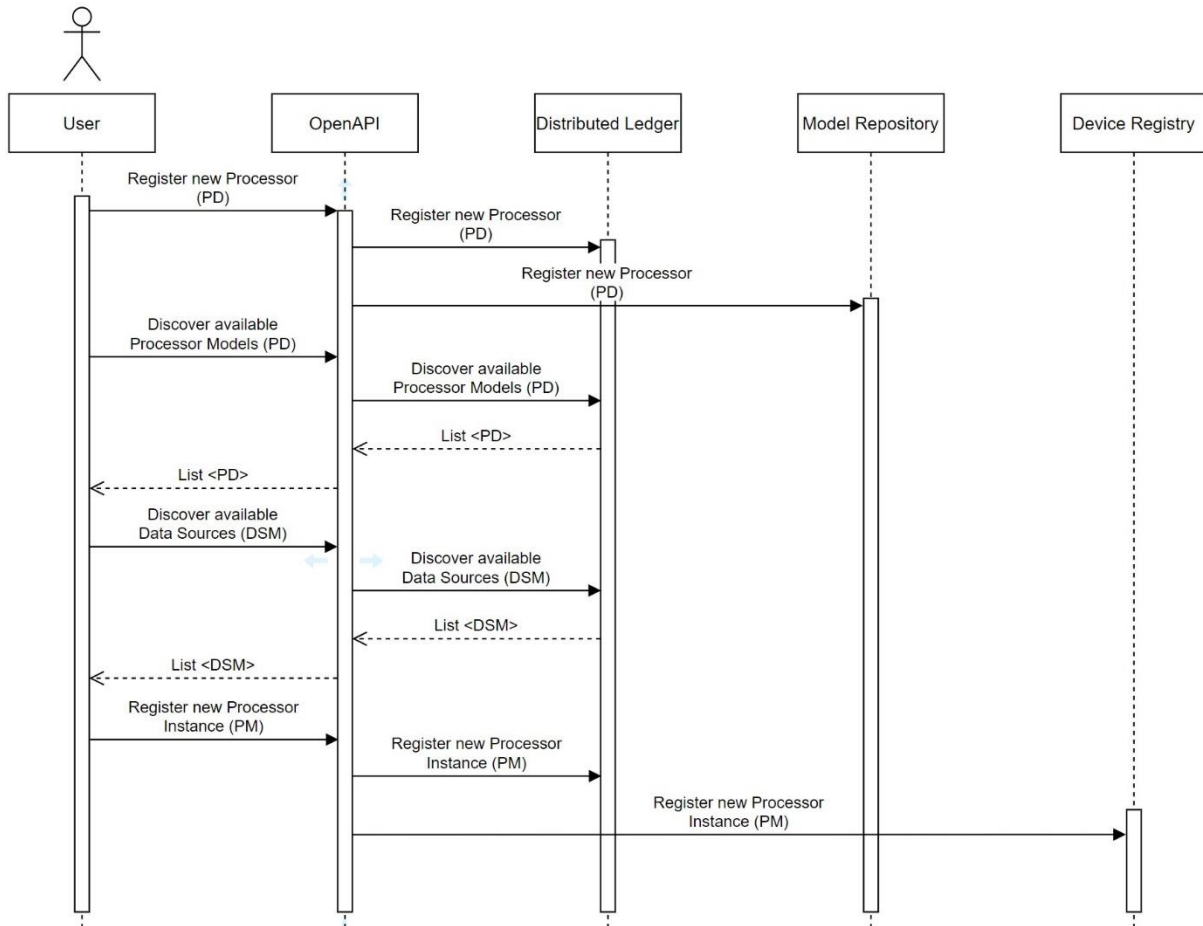


Figure 12 Processor persistence to Distributed Ledger network

As we can see in Figure 12 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.

### 3.4.4 Algorithm Results Traceability

#### 3.4.4.1 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.

##### 3.4.4.1.1 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.

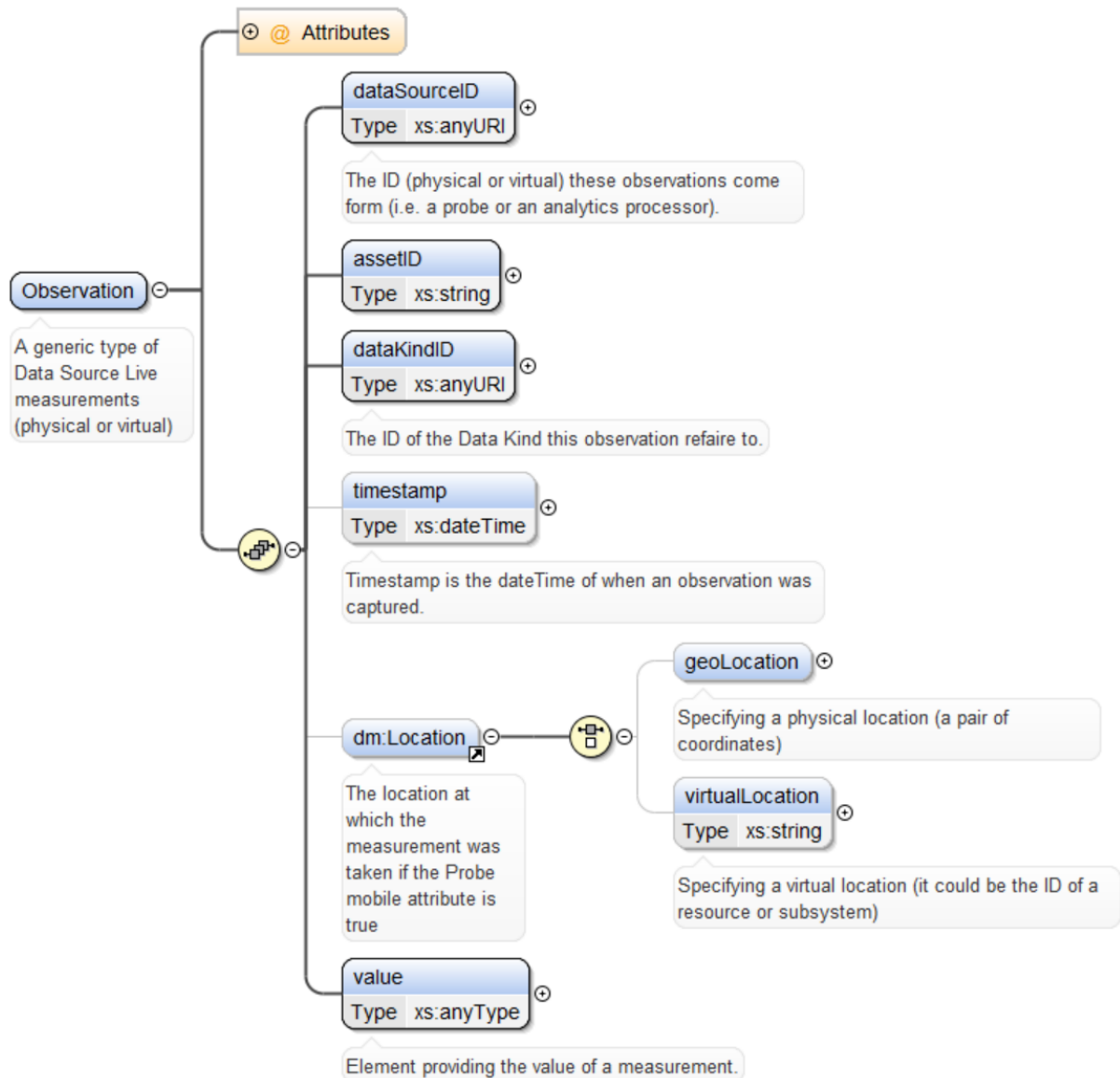


Figure 13 Observation entity of the data model

### 3.4.4.2 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 7 below.

Table 7 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.

### 3.4.5 Recording Traceability Data on the Blockchain

Figure 14 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).

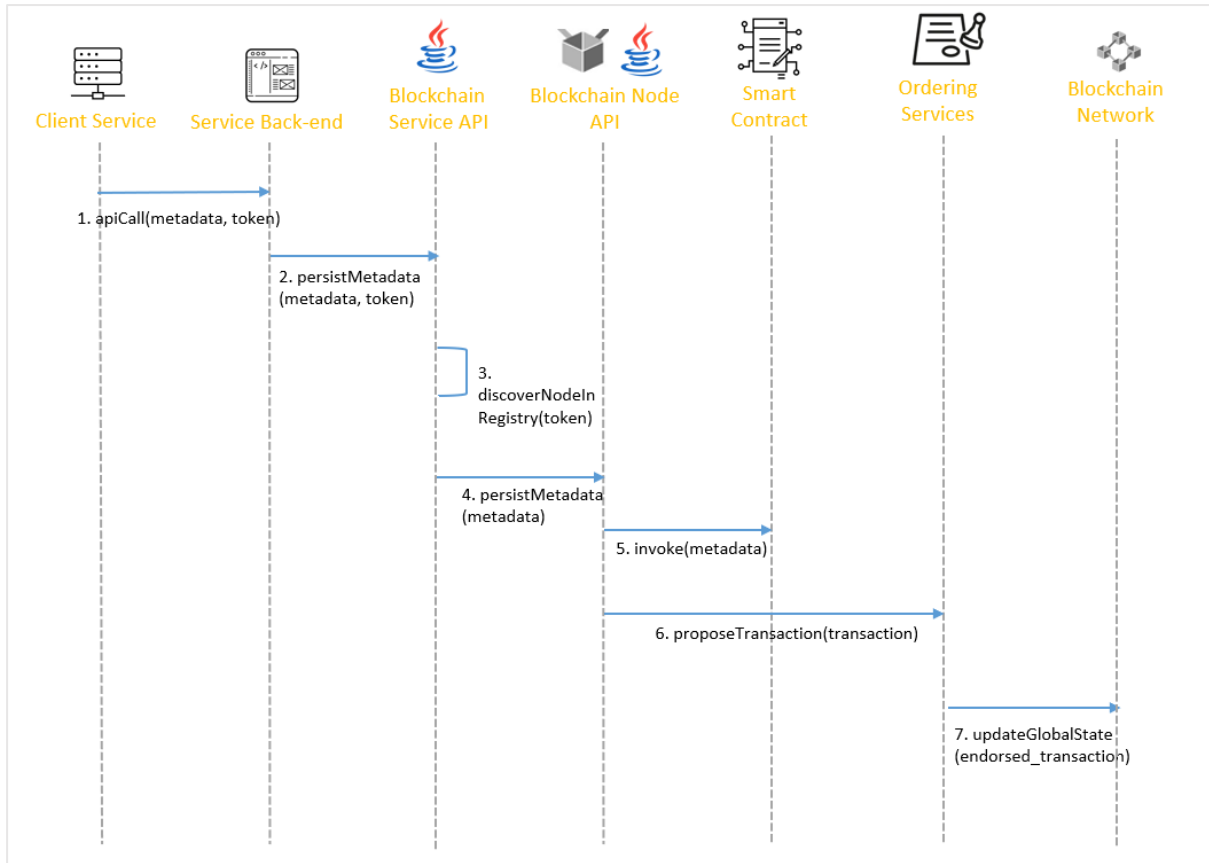


Figure 14: Metadata Persistence using the STAR Blockchain

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 just presented. 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 15.

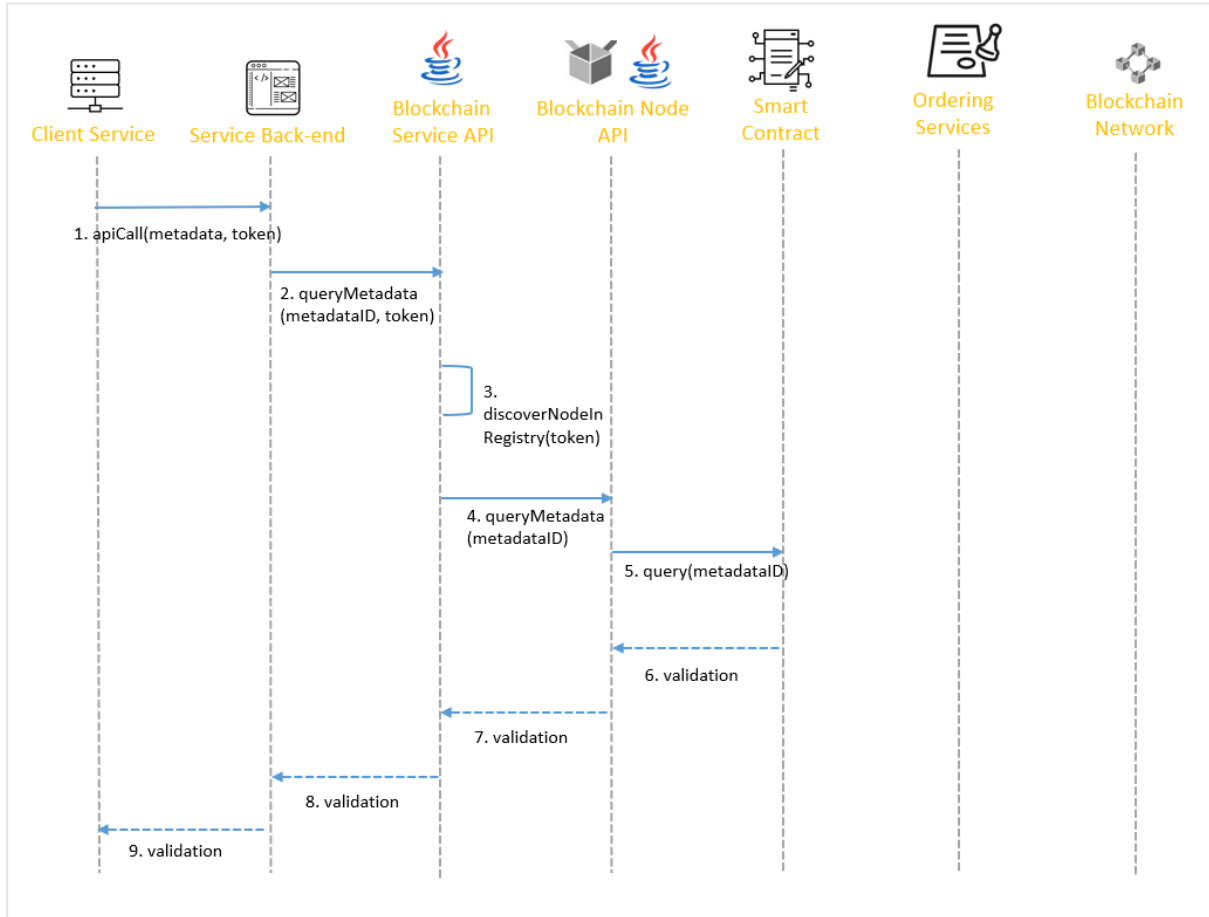


Figure 15: Metadata Validation using the STAR Blockchain

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.

### 3.5 Algorithms Configuration Parameters Validation

#### 3.5.1 How DLSDR Asserts that two Configuration Files are Identical

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 8 ci-dessous shows the complete configuration information for a CNN classifier used in the automatic defect inspection of the Decocap dataset in PCL UC2.

*Table 8 CNN configuration information classifier example*

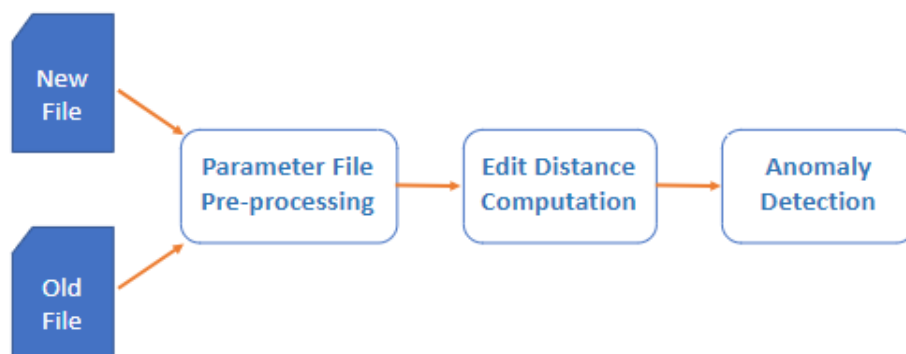
<b>Type of classification</b>	<b>Multiclass</b>		
<b>Dataset</b>	<b>Decocap</b>		
<b>Num Of Epoch</b>	15		
<b>Batch Size</b>	30		
<b>Input Layer shape</b>	(400,400,1)		
<b>Hidden Layers</b>	CNN	units	8
		Kernel size	(3,3)
		padding	same
		activation function	relu
		input shape	(400,400,1)
	Maxpooling	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		

<b>OutputLayer</b>	Dense	units	3
		Activation function	softmax
<b>Loss</b>	categorical_crossentropy		
<b>Optimizer</b>	adam		
<b>Adversarial Algorithm</b>	<b>Multiclass</b>		
	<b>Decocap</b>		
	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).

### 3.5.2 Proposed Approach

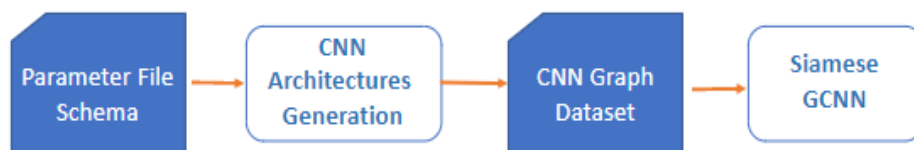
We plan to tackle this problem in two ways. The first is to find an appropriate method of measuring the distance between two configuration files in a semi-structured, hierarchical format such as JSON. This is aimed at covering configuration and parameter files of a general kind. In the second part of 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 16 Flow for General Edit Distance Computation*

The first diagram (see Figure 16 above) 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 17 Training Flow for AI-based CNN-specific Edit Distance Computation*

The next flow (see Figure 17 above) 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.

### 3.5.3 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 18 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.

#### 3.5.3.1 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]

### 3.5.3.2 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.

### 3.5.3.3 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].

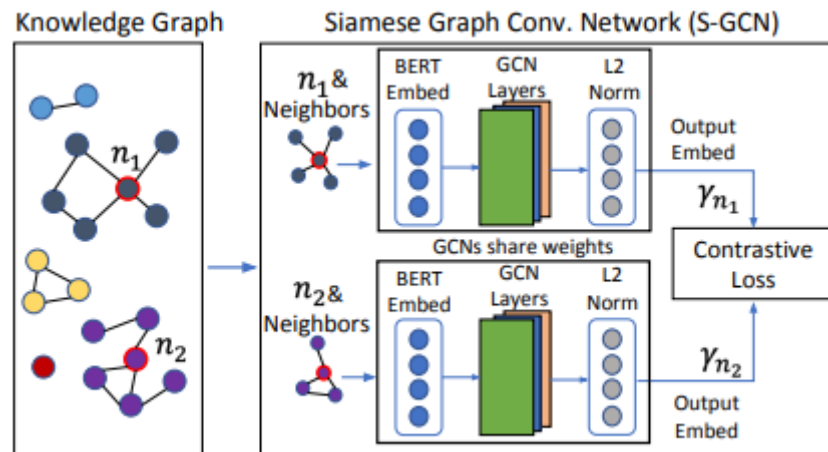


Figure 19 S-GCNN architecture from [Kirosheev21]

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.

### 3.5.3.4 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].

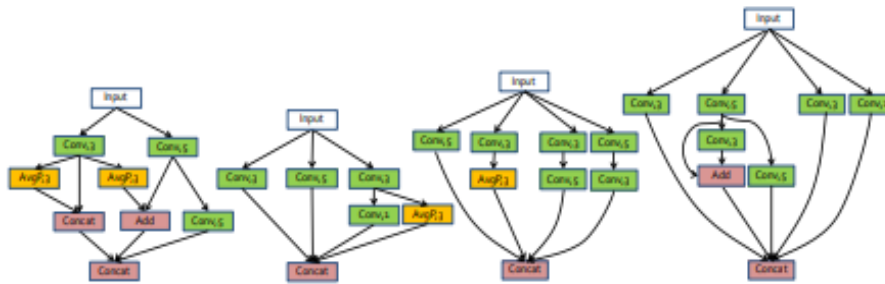


Figure 20 Network architectures generated by variants of Block-QNN [Zhong18]

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] which uses CNN blocks as the basis for generation.

### 3.5.4 Implementation of Parameter Files Validation in the STAR project

With the maturation of STAR components, the finalization of configuration formats and contents and their gradual deployment, we were able, with the help of technical partners, to collect different configuration files. Based on them we developed concrete approaches to guarantee their correct validation using quantifiable measures. As outlined in the previous sections our aim was to utilize statistically determined distance measures to understand which changes in configurations are expected and which should be flagged as anomalous. This is an optional, yet important step to weed out potential adversarial tampering before the configuration files end up in the project's persistent storage. In the following subsections we describe the steps of our methodology, which progressed along two lines:

- Statistical anomaly detection based on exact distance computation;
- Fully approximate detection of anomalies based on Siamese GCNNs

#### 3.5.4.1 Inputs

The extraction of data was performed with meticulous attention in order to identify rows encapsulating useful configuration values, given as input an Excel file containing some

currently existing configurations for the AI algorithms of STAR. Subsequently, these values were parsed into a structured format. Our preliminary analysis indicates the presence of four discrete sets of configurations, each pertaining to different training datasets and methods associated with the use cases of STAR’s pilots PHILIPS and IBER:

- PHILIPS (Shaver dataset) - Adversarial Training
- PHILIPS (Shaver dataset) - Defences Training
- PHILIPS (Soother dataset) - Adversarial Training
- PHILIPS (Soother dataset) - Defences Training
- IBER - Adversarial Training
- IBER - Defences Training

The example input configurations can be seen in the following table (Table 9):

*Table 9: Example of input configurations*

PHILIPS (Shaver dataset)										
Adversarial Training										
Dropout	Activation Function	Dense	Flatten	Image Pixels	Color Mode	Test Size	Epochs	Batch Size	Optimizer	Loss
Range: 0 - 1		Range: 2 - 1024	Range: 0 or 1	Ranges: (50-4096, 50-4096)		Range: 0 or 1	Range: 4 or 128	Range: 1 or 1024		
Value: 0.25	Value: relu	Value: 32	Value: 0	Value: (356, 420)	Value: gray	Value: 0.2	Value: 20	Value: 16	Value: adam	Value: categorical_crossentropy
Defences Training										
Dropout	Activation Function	Dense	Flatten	Image Pixels	Color Mode	Test Size	Epochs	Batch Size	Optimizer	Loss
Range: 0 - 1		Range: 2 - 1024	Range: 0 or 1	Ranges: (50-4096, 50-4096)		Range: 0 or 1	Range: 4 or 128	Range: 1 or 1024		
Value: 0.75	Value: sigmoid	Value: 64	Value: 0	Value: (356, 420)	Value: grayscale	Value: 0.2	Value: 25	Value: 16	Value: adam	Value: binary_crossentropy
PHILIPS (Soother dataset)										
Adversarial Training										
Dropout	Activation Function	Dense	Flatten	Image Pixels	Color Mode	Test Size	Epochs	Batch Size	Optimizer	Loss
Range: 0 - 1		Range: 2 - 1024	Range: 0 or 1	Ranges: (50-4096, 50-4096)		Range: 0 or 1	Range: 4 or 128	Range: 1 or 1024		
Value: 0.5	Value: relu	Value: 128	Value: 0	Value: (256, 256)	Value: grayscale	Value: 0.175	Value: 15	Value: 32	Value: adam	Value: categorical_crossentropy
Defences Training										
Dropout	Activation Function	Dense	Flatten	Image Pixels	Color Mode	Test Size	Epochs	Batch Size	Optimizer	Loss
Range: 0 - 1		Range: 2 - 1024	Range: 0 or 1	Ranges: (50-4096, 50-4096)		Range: 0 or 1	Range: 4 or 128	Range: 1 or 1024		
Value: 0.50	Value: sigmoid	Value: 64	Value: 0	Value: (256, 256)	Value: grayscale	Value: 0.175	Value: 20	Value: 32	Value: adam	Value: binary_crossentropy
IBER										
Adversarial Training										
Dropout	Activation Function	Dense	Flatten	Image Pixels	Color Mode	Test Size	Epochs	Batch Size	Optimizer	Loss
Range: 0 - 1		Range: 2 - 1024	Range: 0 or 1	Ranges: (50-4096, 50-4096)		Range: 0 or 1	Range: 4 or 128	Range: 1 or 1024		
Value: 0.25	Value: relu	Value: 128	Value: 0	Value: (512, 256)	Value: grayscale	Value: 0.175	Value: 15	Value: 16	Value: adam	Value: categorical_crossentropy

Defences Training										
Dropout	Activation Function	Dense	Flatten	Image Pixels	Color Mode	Test Size	Epochs	Batch Size	Optimizer	Loss
Range: 0 - 1		Range: 2 - 1024	Range: 0 or 1	Ranges: (50-4096, 50-4096)		Range: 0 or 1	Range: 4 or 128	Range: 1 or 1024		
Value: 0.75	Value: sigmoid	Value: 64	Value: 0	Value: (512, 256)	Value: grayscale	Value: 0.175	Value: 20	Value: 16	Value: adam	Value: binary_crossentropy

Given the structural uniformity across the datasets, the successful parsing of one dataset shall enable the analogous application of the logic to the remaining sets.

### 3.5.4.2 Parameter File Pre-processing

Configuration names, values and ranges were extracted from the relevant Microsoft Excel file taking into account the structural differences for each configuration field. The configurations extracted are as follows:

- Configuration for PHILIPS (Shaver dataset) - Adversarial Training:
- Dropout: 0.25
- Activation Function: relu
- Dense: 32
- Flatten: 0
- Image Pixels: (356, 420)
- Color Mode: grey
- Test Size: 0.2
- Epochs: 20
- Batch Size: 16
- Optimizer: adam
- Loss: categorical\_crossentropy

During the extraction of the parameter ranges, sensible values were provided to augment the configurations for parameters devoid of explicitly defined ranges (e.g., range for loss is now: ['binary\_crossentropy', 'categorical\_crossentropy']):

- Dropout: [0, 1] (a continuous range from 0 to 1)
- Activation Function: ['relu', 'sigmoid'] (a discrete set of options)
- Dense: [2, 1024] (a continuous range from 2 to 1024)
- Flatten: [0, 1] (a discrete set of options)
- Image Pixels: ([50, 4096], [50, 4096]) (a continuous range for each dimension)
- Color Mode: ['grayscale', 'gray'] (a discrete set of options)
- Test Size: [0, 1] (a discrete set of options)
- Epochs: [4, 128] (a continuous range from 4 to 128)
- Batch Size: [1, 1024] (a continuous range from 1 to 1024)
- Optimizer: ['adam'] (a single option used across all configurations)
- Loss: ['binary\_crossentropy', 'categorical\_crossentropy'] (a discrete set of options)

### 3.5.4.3 Edit Distance Computation

The computation of "Tree Edit Distance" (TED) is generally reserved for tree data structures. In contrast, the configurations at hand are represented as flat dictionaries, and thus, the concept of TED is not directly applicable. A custom distance measure could be proposed, wherein the number of differing values for each key is tallied, or, for a more nuanced approach, numerical values could entail the calculation of the actual difference, and categorical values could be treated as binary distinctions.

Distances were visualized through a heatmap containing the pairwise distances between configurations, so as to give an idea of the distribution of edit distance between currently existing configurations. The heatmap (Figure 21) conveys similarity through a gradient of colours - darker tones indicating closer similarity.

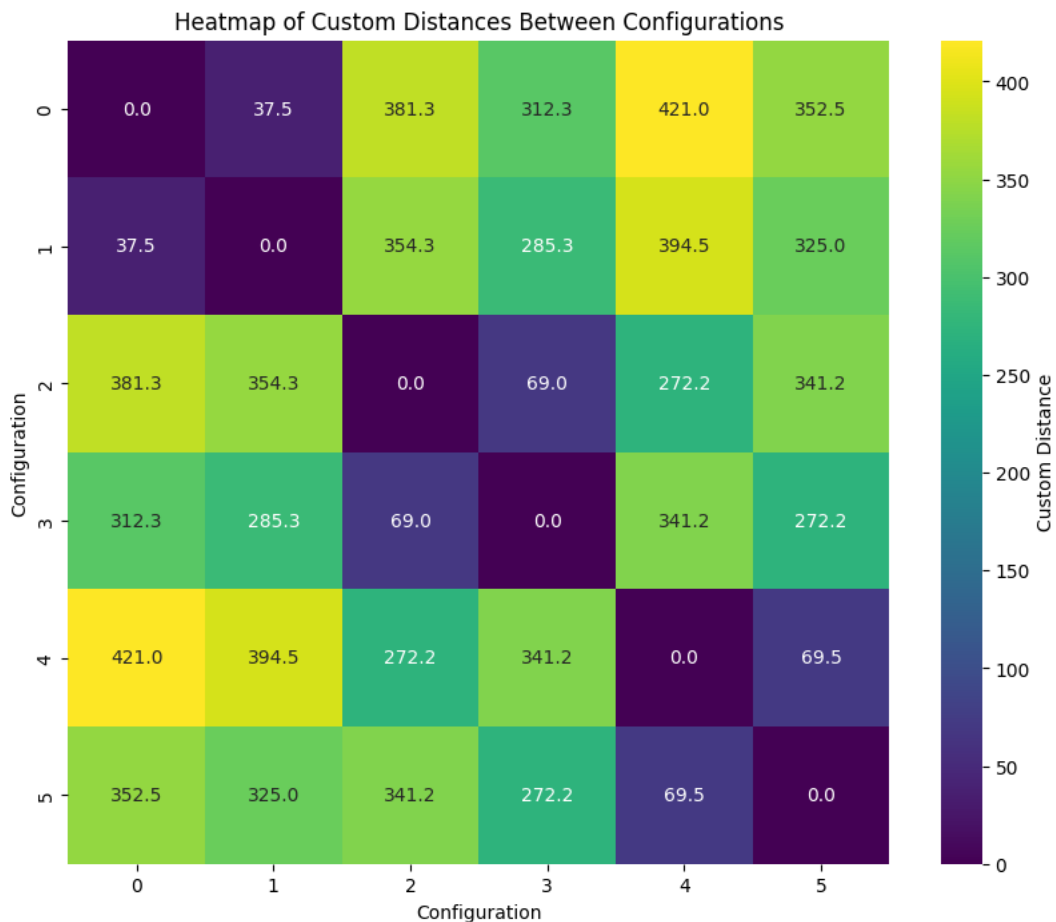


Figure 21: Heatmap of Distances between Configuration of Models

### 3.5.4.4 Anomaly Detection

The detection of anomalies within new parameter files necessitates the establishment of a "normal" distance baseline derived from historical data. Two principal approaches are proposed:

1. **Rule-Based:** Setting a threshold distance, whereby an anomaly is flagged if the new configuration's distance substantially deviates from this threshold.
2. **Time Series Based:** Employing statistical methods to detect outliers within a series of distance measurements over time.

In our implementation of a rule-based approach, the mean and standard deviation of historical distances formed the basis for anomaly identification.

It should be noted that for the current POC implementation focusing on detecting breaking/malicious configurations, the base of historical distances, over which the above statistics are calculated, remains immutable. However, to address configuration drift due to model changes during operation in production, the current component is envisioned to implement further endpoints and mechanisms for the updating of the configuration history to continuously reflect what configurations are defined as “expected”.

#### 3.5.4.5 Generation of Synthetic Data

Given the similar structural pattern of existing configurations, synthetic data generation was simplified to produce values uniformly sampled from the provided ranges. However, in the case that major structural changes appear in the underlying networks, it would become necessary to extend our approach. This could be realized through a synthetic configurations dataset resulting from Neural Architecture Search (NAS) through the following steps:

1. Definition of Parameter Space
2. Selection of Sampling Methods
3. Decisions on Topology
4. Application of Constraints and Heuristics
5. Execution of a Generation Loop
6. Validation of each generated architecture
7. Output of configurations in an appropriate format

The code currently provided will serve as the basis for generating a specified number of synthetic neural network configurations.

#### 3.5.4.6 Siamese GCNN

Additionally, to our anomaly detection approach, we provide methods for the training of a Graph Neural Network (GNN) to ascertain distances between graph embeddings, which implement the following steps:

1. Graph Embeddings
2. Dataset Preparation
3. GNN Architecture Definition
4. Loss Function Definition
5. Training
6. Evaluation
7. Prediction

To facilitate the application of the configurations as inputs to a GNN, a transformation of the configurations into a graph representation is requisite. This entails designating each parameter as a node and defining edges based on parameter relationships. A dataset comprising real and synthetic configurations, coupled with the distances computed by a custom distance function, was utilized to train a GNN model. The model's performance should be assessed through evaluation on a validation set, and subsequent predictions will be generated for new configurations.

#### 3.5.4.7 Component Availability

The implementation of the component can be found in the STAR Repository:

<https://github.com/star-eu/config-validator>

A docker image can be built and deployed to a lab environment using the instructions in the repository's README.md.

### 3.5.4.8 Component Usage

The component encapsulates the aforementioned functionality in a Flask application that can be exposed through a Docker container.

Currently it exposes a single endpoint. To use it send a POST request to the /validateConfigs endpoint. The request body should contain a raw JSON with a list of parameters as follows:

```
{
  "parameters": [
    {
      "name": "Dropout",
      "type": "float",
      "range": [
        0,
        1
      ],
      "value": 0.75
    },
    {
      "name": "Loss",
      "type": "string",
      "range": null,
      "value": "binary_crossentropy"
    }
  ]
}
```

More examples can be found in the payload\_examples subfolder in the project directory.

The JSON config provided through the body of the POST request to the API is expected to contain a list of parameters for a certain STAR AI algorithm. Each parameter in the outer list of parameters is represented by a JSON object that wraps the fields: "name" (the name of the parameter), "type" (its datatype: int, float, string, list etc.), "range" (the range of possible value enforced during config creation) and "value" (the parameter's actual value).

If the input is invalid an error message will be returned:

```
[
  {
    "error": "Invalid arguments!"
  },
  500
]
```

Otherwise, the response will be as follows:

```
[
  {
    "distance": 1115.2666666666667,
    "is_suspicious": "True",
    "mean_distance": 281.94666666666666,
    "threshold": 519.9035043961637
  },
  200
]
```

## 4 Data Reliability Framework PoC Deployment Environment

### 4.1 Cloud Hosting

The software components that comprise the Proof-of-Concept Data Reliability Framework developed in the context of STAR have been deployed on virtual hosts, which are, in essence, cloud servers instantiated on a public cloud provider, named Hetzner Cloud<sup>17</sup>. Effective firewall rules have been configured on each virtual server to securely handle inbound and outbound network traffic, ensuring that only authorized users and applications are permitted to attain access. Furthermore, all the storage units (either ephemeral or persistent volumes) are encrypted in order to mitigate the risks of data exposure.

Figure 22, below, depicts the project space created to accommodate the needs of STAR on Hetzner Cloud.

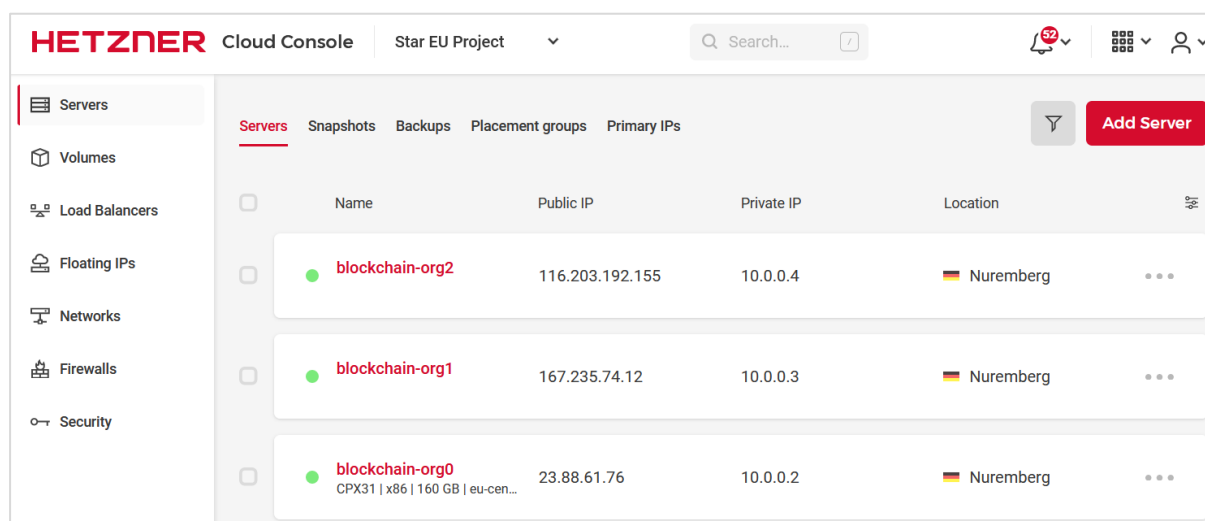


Figure 22: STAR Infrastructure Administrator Dashboard on Hetzner Cloud

All three machines employ Ubuntu 22.04.1 LTS as the operation system.

The recommended configuration for a Manager workstation (blockchain-org0) 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 (blockchain-org1 and blockchain-org2) workstation:

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

<sup>17</sup> Cloud provider's official website: <https://www.hetzner.com/cloud> (accessed October 2023)

*Note: the concepts of Manager and Worker workstation are pertaining to Orchestration and are being analysed in Section 4.6 a little further ahead.*

The features characterizing the STAR Data Reliability Framework hosting environment are the following:

- **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 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.

## 4.2 Certificate Management Protocols

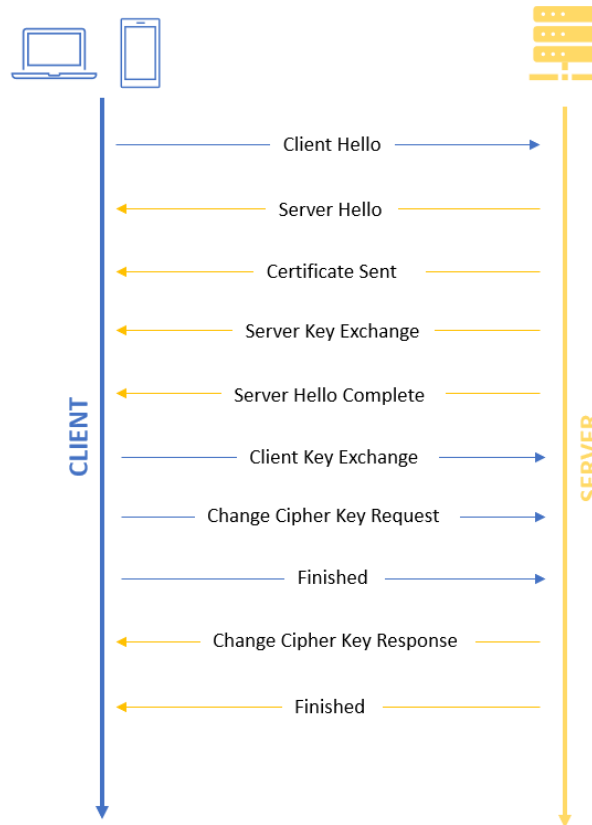
**SSL** (Secure Sockets Layer) and its successor, **TLS** (Transport Layer Security), are protocols for establishing authenticated and encrypted links between networked computers. SSL/TLS works by binding the identities of entities such as websites and companies to cryptographic key pairs via digital documents known as X.509 certificates. Each key pair consists of a private key and a public key. The private key is kept secure, and the public key can be widely distributed via a certificate. If the SSL/TLS certificate itself is signed by a publicly trusted **Certificate Authority** (CA) the certificate will be implicitly trusted by client software such as web browsers and operating systems.

The most common and well-known use of SSL/TLS is secure web browsing via the **Hypertext Transfer Protocol Secure (HTTPS)** protocol. A properly configured public HTTPS website includes an SSL/TLS certificate that is signed by a publicly trusted CA. Users visiting an HTTPS website can be assured of:

- **Authenticity.** The server presenting the certificate is in possession of the private key that matches the public key in the certificate.
- **Integrity.** Documents signed by the certificate (e.g., web pages) have not been altered in transit by a man-in-the-middle (MitM) attack.
- **Encryption.** Communications between the client and server are encrypted.

Because of these properties, SSL/TLS and HTTPS allow users to securely transmit confidential information such as login credentials over the Internet and be sure that the website they are sending them to is authentic [SSL19].

Data sent using HTTPS is secured via the **Transport Layer Security** protocol (TLS), whose handshake is illustrated in Figure 23 below:



*Figure 23: How TLS 1.3 handshake works*

TLS provides three key layers of protection:

- (i) **Encryption** - encrypting the exchanged data to keep it secure from eavesdroppers. That means that while users are using an HTTPS secured service, nobody can "listen" to their conversations, track their activities across multiple pages, or steal their information.
- (ii) **Data integrity** - data cannot be modified or corrupted during transfer, intentionally or otherwise, without being detected.
- (iii) **Authentication** - proves that users communicate and send data to the intended service. It protects against man-in-the-middle (MitM) attacks and builds user trust.

To secure data transmission to the Data Reliability Framework and achieve identification of the different provided services, STAR employs SSL communication. This has been achieved by leveraging the Certificate Authorities provided by Hyperledger Fabric to generate certificates that are signed by the trusted CA, ensuring that the certificate holder is really who they claim to be. An extensive description of the CAs and their particular roles is included in Section 5.2 of the present document.

### 4.3 Hard Disk Encryption of Cloud Machines

To enhance the security of the STAR Data Reliability Framework deployed on Hetzner Cloud's virtual machines, all hard disks attached to these servers have been encrypted. This encryption ensures that data files are persistently stored in an encrypted state, safeguarding against potential threats like bare-metal attacks on the underlying hard drives. In the event that someone gains physical access to the disk, they would face a formidable challenge in

attempting to decipher the encryption key through brute force methods, thereby substantially obstructing their access to the stored data. Furthermore, the encrypted files only become accessible to the operating system and authorized applications in a readable format while the system is actively running and unlocked by a trusted user. This proactive security measure ensures that an unauthorized individual attempting to inspect the disk contents directly would encounter only unintelligible, scrambled data rather than the original files. This robust encryption mechanism serves to protect sensitive data, particularly in scenarios where the hard disk or server may be lost, stolen, or disposed of at the end of its lifecycle, fortifying the overall data security posture.

#### 4.4 SSH key-based authentication

**Secure Shell (SSH)** access, to administer and manage the servers, has been configured, to use only key-based authentication. This is a more secure alternative in comparison to the most commonly used password authentication. Although passwords are sent to the server in a secure manner, they are sometimes not complex or long enough to be resistant to repeated, persistent attackers. Modern processing power combined with automated scripts make brute forcing a password-protected account very possible. Thus, SSH public key authentication in which we generate and store a pair of cryptographic keys and then configure the servers to recognize and accept the generated keys is preferable security-wise. The high-level logic of the SSH handshake between server and clients is being illustrated in Figure 24.

To solidify this approach, as far as the various virtual machines hosting the blockchain infrastructure are concerned, all being equipped with Linux operating systems, the possibility to connect remotely with a password has been deactivated.

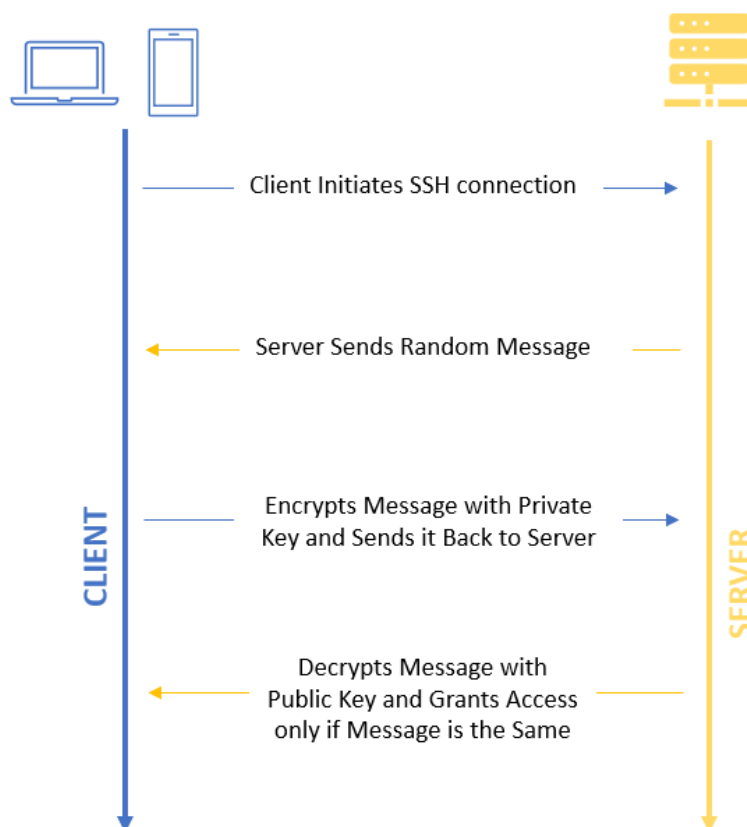


Figure 24: Basic principle of SSH authentication handshake

## 4.5 Containerization of Microservices

### 4.5.1 Overview

**Containerization** is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable - called a container - that runs consistently on any infrastructure. More portable and resource-efficient than virtual machines (VMs), containers have become the de facto compute units of modern cloud-native applications.

Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from a desktop computer to a virtual machine (VM) or from a Linux to a Windows operating system. Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run. This single package of software or "container" is abstracted away from the host operating system, and hence, it stands alone and becomes portable - able to run across any platform or cloud, free of issues.

The concept of containerization and process isolation is actually decades old, but the emergence in 2013 of the open-source **Docker Engine**<sup>18</sup> - an industry standard for containers

<sup>18</sup> Docker Engine official website: <https://www.docker.com/products/container-runtime> (acc. October 2023)

with simple developer tools and a universal packaging approach - accelerated the adoption of this technology [IBM21].

#### 4.5.2 Docker

**Docker** is a set of Platform-as-a-Service (PaaS) products that use OS-level virtualization to deliver software in the aforementioned lightweight packages called containers. It creates simple tooling and a universal packaging approach that bundles up all application dependencies inside a container. The latter is then run on Docker Engine. Docker Engine enables containerized applications to seamlessly run anywhere, consistently on any Linux, Windows, or macOS infrastructure, solving “dependency hell” for developers and operations teams.



Docker **images** contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When one runs the Docker image, it becomes one instance (or multiple instances) of the container. Docker **containers** are the live, running instances of Docker images. While Docker images are read-only files, containers are live, ephemeral, executable content.

Images can be published in a shared repository, such as the Docker Registry<sup>19</sup> or DockerHub<sup>20</sup> and through the docker pull command or through the docker-compose pull functionality these images can be retrieved from the Docker registry and deployed together via a single configuration file. Containerization thus provides OS level virtualization. This means that multiple applications running in containers on a single host, access the same OS kernel. Hence, it is faster and more lightweight than isolating applications using VMs.

Containers have an initial configuration which does not affect the configuration of other containers, even though they share the same host OS. This eliminates errors due to unexpected conflicts or missing dependencies, which are common when applications are installed on a single host without isolation. In addition, in more demanding installations due to increased load of the system, Docker is perfectly suitable to be configured with load balancing mechanisms that can scale up the performance of the system.

#### 4.5.3 Docker as the Cornerstone of STAR Fabric Network

Within the STAR blockchain infrastructure Docker plays a pivotal role in ensuring the efficiency and scalability of the system. Docker containers are employed to encapsulate and deploy various components, including the Hyperledger Fabric nodes, smart contracts (chaincode), façade services, monitoring tools and other associated microservices. By using Docker containers, STAR can achieve a high degree of portability and consistency in its software environment. This means that regardless of the underlying infrastructure or platform, the project can consistently run and manage its blockchain network and related services in a containerized environment. Docker containers simplify the process of packaging and distributing software, ensuring that all dependencies and configurations are consistent across development, testing, and production environments. This not only streamlines the deployment process but also facilitates rapid scaling of the system when needed.

<sup>19</sup> Docker Registry official website: <https://docs.docker.com/registry> (accessed October 2023)

<sup>20</sup> DockerHub official website: <https://hub.docker.com/> (accessed October 2023)

## 4.6 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 Swarm<sup>21</sup> for creating and orchestrating<sup>22</sup> 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 25 below summarises the concepts described above.

---

<sup>21</sup> Official documentation of Swarm within Docker's official website: <https://docs.docker.com/engine/swarm/> (accessed October 2023).

<sup>22</sup> Orchestration as a concept explained in a CISCO article: <https://www.cisco.com/c/en/us/solutions/cloud/what-is-container-orchestration.html> (accessed October 2023).

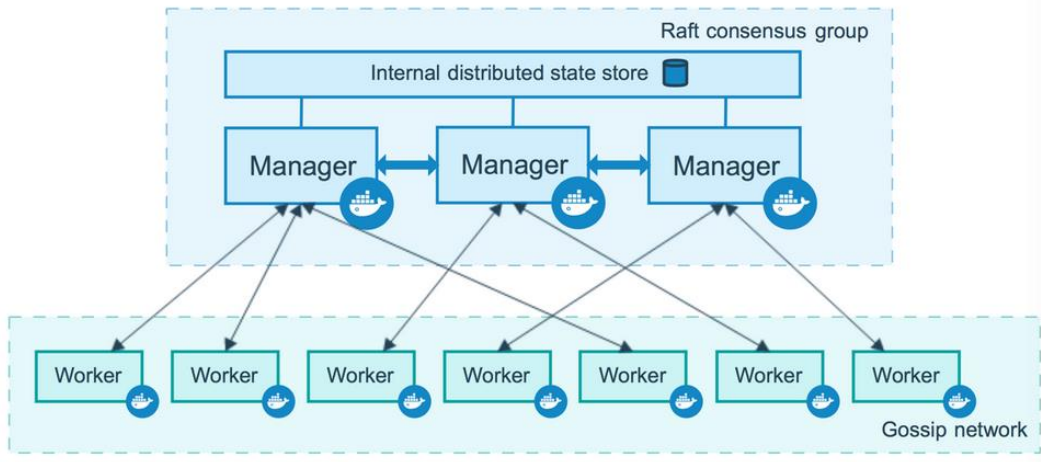


Figure 25: A Docker Swarm Cluster Example [SwarmDocs]

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 26 below summarises the concepts just described.

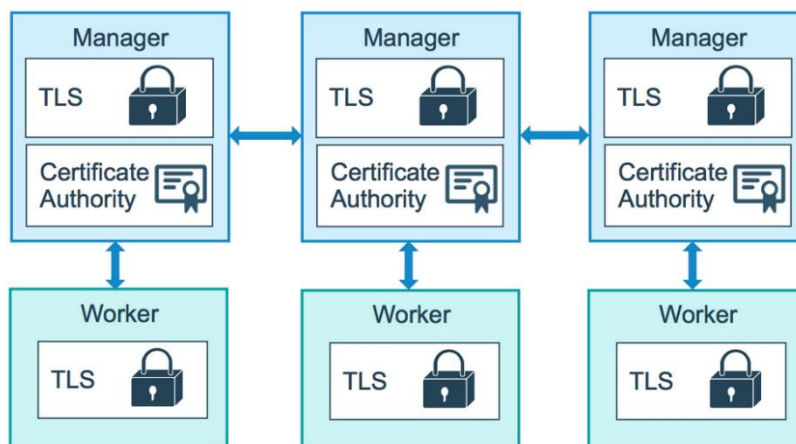


Figure 26: Swarm Security Concept with PKI [SwarmDocs]

The following section discusses the topology devised for the cluster of the Docker containers synthesizing the blockchain services of STAR’s PoC. 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.

### 4.7 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. 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 27.

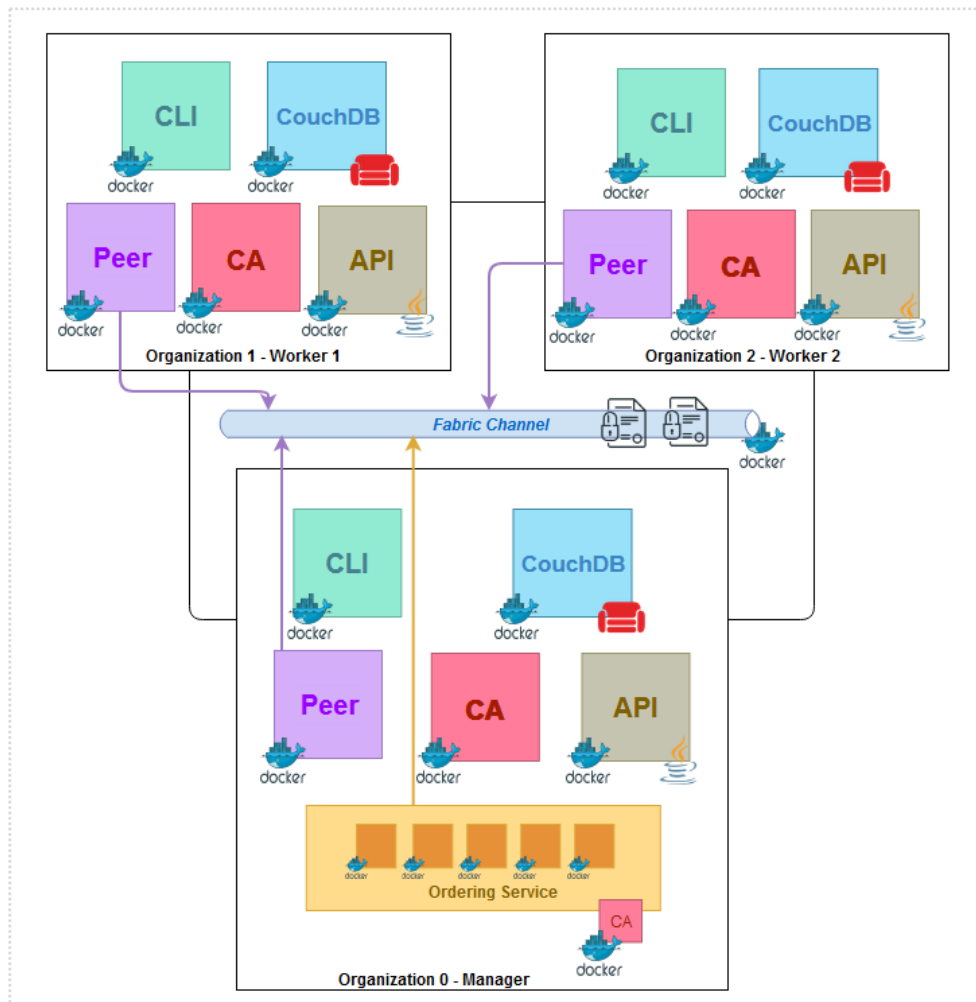


Figure 27: Docker Swarm Overlay Topology for STAR MVP Fabric Network

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.
- 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.
- 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.

As evidenced by Figure 27, 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 purposes of the PoC 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 Spring<sup>23</sup> Boot<sup>24</sup> 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.

Throughout the following chapter we will provide detailed insights into the intricacies of the Hyperledger Fabric’s architecture and elucidate its deployment procedures in the context of STAR.

<sup>23</sup> Spring official webpage: <https://spring.io/> (accessed October 2023)

<sup>24</sup> Spring Boot official webpage: <https://spring.io/projects/spring-boot> (accessed October 2023)

## 5 Data Reliability Framework PoC Architecture Dissection and Deployment Procedure

### 5.1 Introduction

Having previously described the deployment environment that is hosting the Data Reliability Framework Proof-of-Concept (PoC), throughout the present chapter we will provide detailed insights into the intricacies of the actual component's architecture, elucidate the deployment procedures, and offer a comprehensive examination of how Hyperledger Fabric's capabilities empower the wider platform's cybersecurity technological solutions to catalyze transformative change within the domains of security, trust, and traceability. This examination serves, consequently, as a testament to the pioneering role played by blockchain technology in driving innovation in the domains of transparency and data traceability within the broader platform.

In the official GitHub repository of the project, one may access a kit that contains all necessary materials to reproduce the installation process of the Distributed Ledger Services and Data Reliability component. The following kit must be cloned using git in all three virtual machines under the same directory. Subsequently, one may follow the instructions included in the Sections that follow.

<https://github.com/star-eu/dlsdr-deployment-kit>

### 5.2 Authorizers of Decentralized Network Participants

#### 5.2.1 Certificate authorities in the context of Hyperledger Fabric

In Hyperledger Fabric, **Certificate Authorities (CAs)** play a critical role in establishing and managing the security infrastructure of the network. CAs are responsible for issuing, revoking, and managing digital certificates that are used to authenticate and secure network participants, including organizations, peers, orderers, and clients. These digital certificates ensure the confidentiality, integrity, and authenticity of communications within the blockchain network. Each organization within a Fabric network typically has its own CA that manages certificates for its members. CAs generate cryptographic key pairs, sign certificates, and handle certificate revocation. Hyperledger Fabric's CA model allows for flexible identity management, enabling each participant to have unique cryptographic identities while maintaining a high level of security. By enforcing proper authentication and encryption, CAs contribute to maintaining the overall trust and security of the Hyperledger Fabric network.

A Certificate Authority (CA) provides a number of certificate services to users of the Fabric blockchain. Its main function is to produce certificates used for user enrolment, sourcing of transactions invoked on the blockchain, and TLS<sup>25</sup>-secured connections between users or components of the blockchain. Fabric CAs use a client/server architecture; here, the server can be implemented as a cluster to support high availability (HA). As with any CA, the Fabric CA can generate self-signed X.509 certificates<sup>26</sup>. Self-signed X.509 certificates are digital

---

<sup>25</sup> Transport Layer Security (TLS) by the Internet Engineering Task Force: <https://datatracker.ietf.org/wg/tls/documents/> (accessed October 2023).

<sup>26</sup> X.509 standard by the International Telecommunication Union: <https://www.itu.int/rec/T-REC-X.509> (accessed October 2023).

certificates used in Public Key Infrastructure (PKI)<sup>27</sup> that are signed by their own private key, rather than by a trusted certificate authority. They are typically used for testing or internal purposes, as they lack the third-party validation that makes certificates from trusted authorities secure for widespread public use.

In a real-world scenario, the Certificate Authority (CA) administrator plays a pivotal role in the process of registering users within a Hyperledger Fabric network. The CA administrator begins by gathering the necessary information and identity details from the users who are intended to participate in the network. Once this information is collected and verified, the CA administrator generates unique enrolment IDs for each user and associates them with the user's identity. These enrolment IDs serve as the initial point of contact between the user and the CA.

Subsequently, the CA administrator hands over these enrolment IDs to their organization's administrator, who is typically responsible for managing and operating the nodes and participants within their specific organization. Armed with these enrolment IDs, the organization's administrator takes on the task of enrolling the users into the network. This enrolment process involves the following steps:

1. **User Enrolment Request:** The organization's administrator initiates the enrolment process by submitting an enrolment request to the CA. This request includes the user's enrolment ID and any additional authentication or authorization information required by the CA.
2. **Identity Verification:** The CA verifies the authenticity of the enrolment request and cross-references it with the provided enrolment ID. This step ensures that only legitimate users from the organization can request enrolment.
3. **Certificate Issuance:** Upon successful verification, the CA issues an enrolment certificate to the user. This enrolment certificate contains a public-private key pair unique to the user. The private key remains securely with the user while the public key is part of the certificate.
4. **Secure Storage:** The user, under the guidance of the organization's administrator, securely stores the enrolment certificate and private key. This storage is crucial to protect the confidentiality and integrity of the private key, as it must not fall into unauthorized hands.
5. **Subsequent Authentication:** With the enrolment certificate in hand, the user can now authenticate themselves to the Hyperledger Fabric network. The private key is used for cryptographic operations, including signing transactions and endorsing proposals.

Importantly, the entire process ensures that the private certificates, which are essential for user authentication and secure transactions, are self-managed and confined solely within the boundaries of the organization. This approach enhances security and control, as the organization retains exclusive management of its users' credentials and cryptographic keys, reducing the risk of unauthorized access or tampering from external entities. It also aligns with the core principles of decentralization and trust that underpin Hyperledger Fabric networks, where each organization maintains autonomy over its assets and participants<sup>28</sup>.

<sup>27</sup> Public Key Infrastructure definition by ENISA: <https://www.enisa.europa.eu/topics/incident-response/glossary/public-key-infrastructure-pki> (accessed October 2023).

<sup>28</sup> Hyperledger Fabric Certificate Authority official documentation available at: <https://hyperledger-fabric-ca.readthedocs.io/en/latest/index.html> (accessed October 2023).

### 5.2.2 Certificate Authorities in the Context of STAR PoC Architecture

In our “proof-of-concept” architecture for the purposes of STAR we have deployed one Certificate Authority per Organization in their respective virtual machines. In addition, the “Manager” is hosting one more CA to serve the Orderers. To be precise, the “manager” node will be hosting the CA of the “organization-zero” and the CA of the orderers, while the workers will be hosting the CAs of “organization-one” and “organization-two” respectively. Each organization (representing a STAR service developed and maintained by a different shareholder) maintains control over its own CA and the digital identities of its members, ensuring that sensitive cryptographic keys and certificates are securely managed within their own virtual machines. This distributed approach aligns with the principles of decentralization and trust inherent in blockchain networks, while also facilitating secure and efficient communication and coordination among the participating organizations in the STAR project.

Since we have produced a proof-of-concept and deployed all the infrastructure components ourselves, the sequence of actions described in the previous section has only been adhered to, without the confidentiality aspect. There was obviously not need for a distinct CA administrator and organization administrator, nor the need for multiple users per organization. However, should the need have arisen no scalability problems would have appeared on this front, as the inherent properties of Fabric supporting multiple users with different access rights could have been leveraged.

The figure below (Figure 28) helps illustrate the placement of the four CAs within the three virtual machines. The respective Docker containers are represented with the colour red.

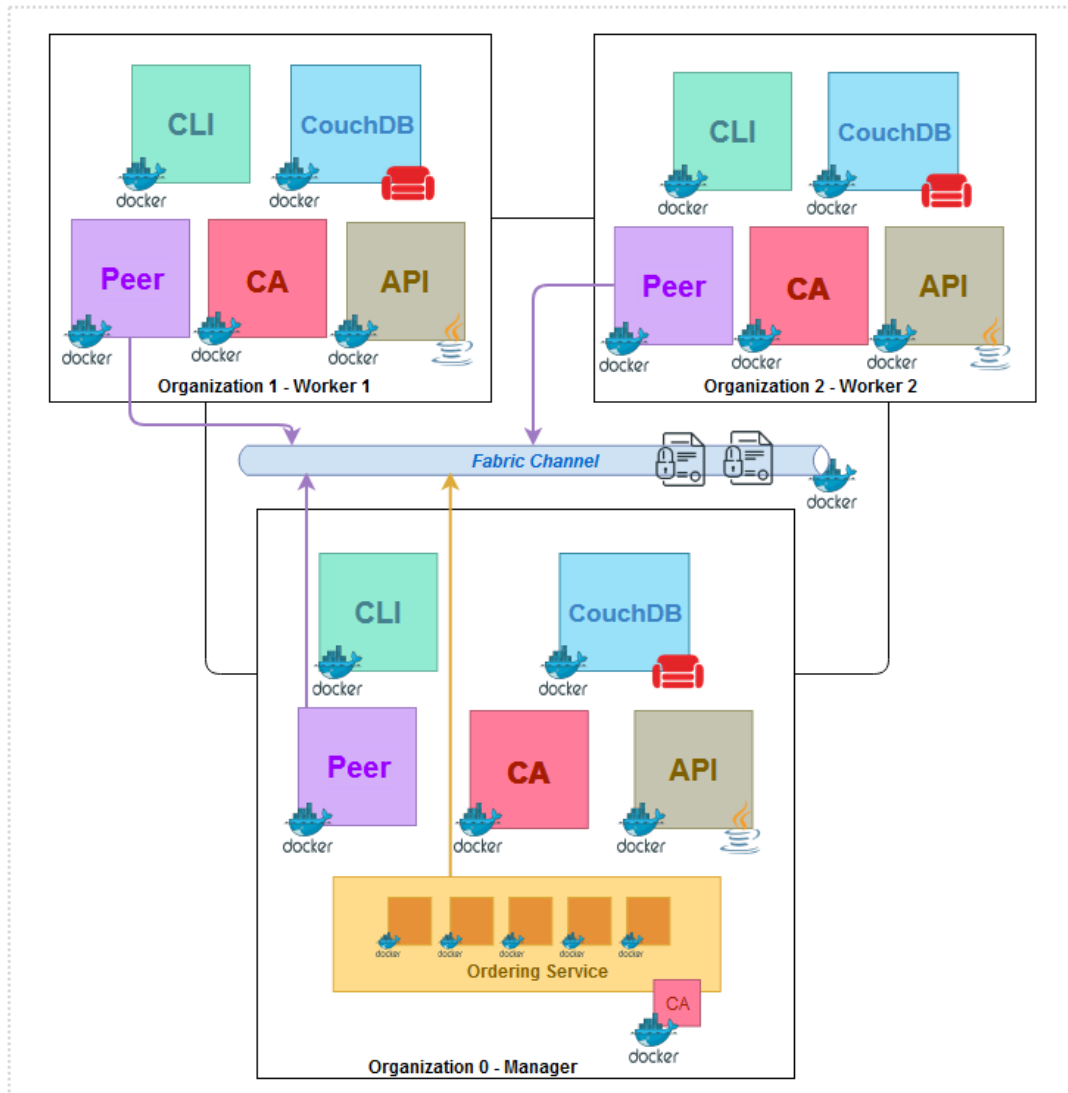


Figure 28: Certificate Authorities in the context of the STAR PoC Fabric Network (as red Docker containers)

### 5.2.3 Configuration and Deployment of the Certificate Authorities

While most of the script executions will place principally on the command line of the virtual machine hosting the “manager” Swarm node, it is essential to create the deployment kit’s directories structure in **all three machines** (using git as explained earlier in the introductory section).

Various configuration files are used to configure and customize the behavior of the Fabric CA server. These files specify parameters like the CA’s listening address, database settings, and other operational details. They are located in the following directory of our deployment kit:

*fabric-deployment/network-setup/organizations/fabric-ca*

The configuration file enacting the actual deployment of the CAs as Docker containers is located here:

*fabric-deployment/network-setup/docker/docker-compose-ca.yaml*

Those configuration files are project and domain agnostic and permit the deployer to customize several parameters such as the project's domain (in our case *star-ai.eu*) and the certification authorities' geographical location. We assume that each Organization (corresponding to a Cybersecurity software component of the STAR platform) is hosted in a different country. Through the console of each virtual machine, we proceed to substitute on the aforementioned configuration files the environment variables with precise values.

All commands of the installation, in **all three virtual machines**, must be executed after having navigated to the following directory:

*fabric-deployment/network-setup/*

After that, on the "blockchain-org0" virtual machine (a.k.a. "manager" node) one may execute the following to prepare the *docker-compose-ca.yaml* file:

```
sed -e 's/${PROJECT_DOMAIN}/star-ai.eu/g' -e
's/${SHORT_DOMAIN}/staraieu/g' docker/docker-compose-ca-matrix.yaml >
docker/docker-compose-ca.yaml
```

Next, on the same machine, one may also execute the following to prepare the *fabric-ca-server-config.yaml* files defining the CAs for orderers and org0 that we assume are physically located in Athens:

```
sed -e 's/${PROJECT_DOMAIN}/star-ai.eu/g' -e
's/${CA_STATE}/Attica/g' -e 's/${CA_CITY}/Athens/g' -e
's/${CA_COUNTRY_ABBR}/GR/g' organizations/fabric-ca/org0/fabric-ca-server-
config-matrix.yaml > organizations/fabric-ca/org0/fabric-ca-server-
config.yaml
```

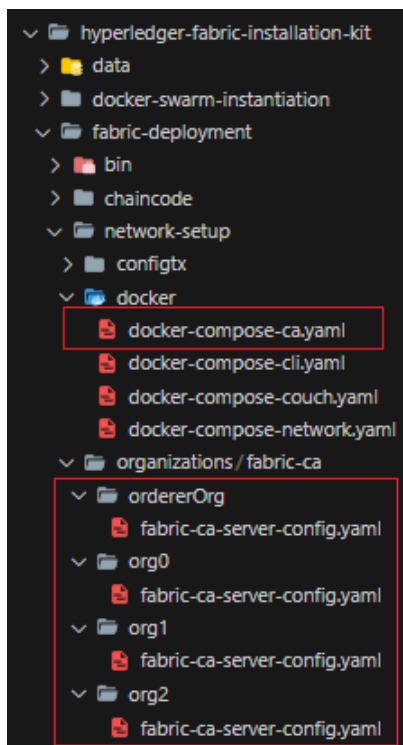
Then, with the same target in mind, one may execute on the machine prepared for "worker1" the following command that places org1 in Bologna:

```
sed -e 's/${PROJECT_DOMAIN}/star-ai.eu/g' -e 's/${CA_STATE}/Emilia-
Romagna/g' -e 's/${CA_CITY}/Bologna/g' -e 's/${CA_COUNTRY_ABBR}/IT/g'
organizations/fabric-ca/org1/fabric-ca-server-config-matrix.yaml >
organizations/fabric-ca/org1/fabric-ca-server-config.yaml
```

Then, with the same target in mind, one may execute on the machine prepared for "worker2" the following command that places org2 in Hildesheim:

```
sed -e 's/${PROJECT_DOMAIN}/star-ai.eu/g' -e 's/${CA_STATE}/Lower
Saxony/g' -e 's/${CA_CITY}/Hildesheim/g' -e 's/${CA_COUNTRY_ABBR}/DE/g'
organizations/fabric-ca/org2/fabric-ca-server-config-matrix.yaml >
organizations/fabric-ca/org2/fabric-ca-server-config.yaml
```

The hierarchy between the directories and the newly created configuration files is depicted in the following figure (Figure 29):



*Figure 29: Material for CA deployment in the installation kit*

After that, to deploy the certification authorities (CAs) as Docker containers, we have used a technique, that deploys the containers with a single command as services to the entirety of the Swarm cluster. The command to deploy the stack was executed in the command line of the “manager” Swarm node only, from within the “network-setup” folder. By leveraging the labels we had already associated to the various Swarm nodes, we explicitly indicated which container will be deployed in which virtual machine / network node. The *docker-compose-ca.yaml* file contains configurations for four services entitled “ca\_org0”, “ca\_org1”, “ca\_org2” and “ca\_orderer”. Let us examine an example (Figure 30):

```

16  ca_org0:
17    deploy:
18      placement:
19        constraints:
20          - node.labels.name == manager
21      image: hyperledger/fabric-ca:1.5.2
22      environment:
23        - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
24        - FABRIC_CA_SERVER_CA_NAME=ca-org0
25        - FABRIC_CA_SERVER_TLS_ENABLED=true
26        - FABRIC_CA_SERVER_PORT=7054
27      ports:
28        - "7054:7054"
29      command: sh -c 'fabric-ca-server start -b admin:adminpw -d'
30      volumes:
31        - ../organizations/fabric-ca/org0:/etc/hyperledger/fabric-ca-server
32      container_name: ca_org0
33      networks:
34        fabric-network:
35          aliases:
36            - ca.org0.${PROJECT_DOMAIN}

```

Figure 30: Definition of a CA for organization-zero

The service is named “ca\_org0” and, obviously, represents the certificate authority of “blockchain-org0”. It is clearly stated that the containers formulating the service are expected to be deployed only on the Swarm network nodes marked where the label “name” has the value “manager”. Then we state the directory of certificate authority within the Docker container, its name i.e., “ca-org0”, that TLS has been enabled and that the port of the service is 7054. We follow up by exposing said port and declaring the command that instantiates the service within the container. Then a specific directory of the hierarchy we presented earlier to be used as a volume for the container (note: this must be existent in all nodes bearing the “manager” role. Then a name for the container (this can be deprecated in the latest versions). Finally, the alias with which our container will be known within the Docker network we created earlier (i.e., ca.org0.star-ai.eu). The other services follow suit.

A summary of the Certificate Authorities of our proof-of-concept are the following (Table 10):

Table 10: CA information per Virtual Machine

Name	Swarm Node	Network Alias
ca_org0	manager	ca.org0.star-ai.eu
ca_org1	worker1	ca.org1.star-ai.eu
ca_org2	worker2	ca.org2.star-ai.eu
ca_orderer	manager	ca.orderer.star-ai.eu

The commands to deploy the stack via the “manager” node is the following:

```

cd fabric-deployment/network-setup/

docker stack deploy -c docker/docker-compose-ca.yaml hyperledger_fabric

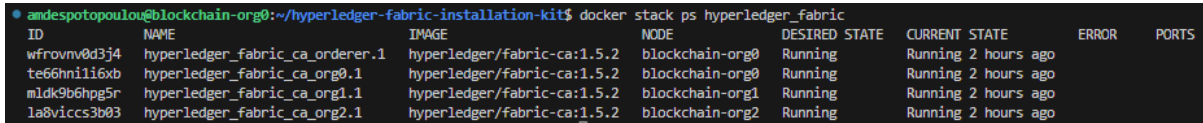
```

### 5.2.4 Verification of the Certificate Authorities installation

To make sure that everything is in place, we can type at the console of our “manager”:

```
docker stack ps hyperledger_fabric
```

The result is illustrated below (Figure 31):



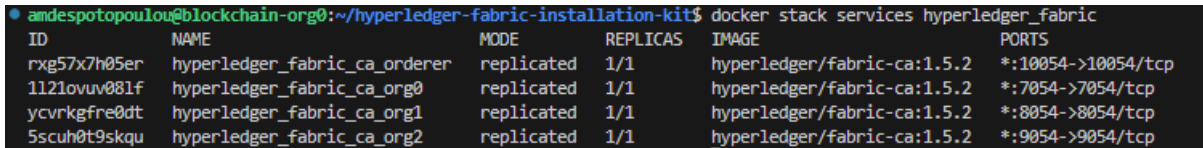
ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
wfrovnv0d3j4	hyperledger_fabric_ca_orderer.1	hyperledger/fabric-ca:1.5.2	blockchain-org0	Running	Running 2 hours ago		
te66hni1i6xb	hyperledger_fabric_ca_org0.1	hyperledger/fabric-ca:1.5.2	blockchain-org0	Running	Running 2 hours ago		
m1dk9b6hpg5r	hyperledger_fabric_ca_org1.1	hyperledger/fabric-ca:1.5.2	blockchain-org1	Running	Running 2 hours ago		
1a8viccs3b03	hyperledger_fabric_ca_org2.1	hyperledger/fabric-ca:1.5.2	blockchain-org2	Running	Running 2 hours ago		

Figure 31: The result of "docker stack ps hyperledger\_fabric" command

Alternatively, to verify the services that are part of the deployed stack one may type:

```
docker stack services hyperledger_fabric
```

The result is illustrated below (Figure 32):



ID	NAME	MODE	REPLICAS	IMAGE	PORTS
rxg57x7h05er	hyperledger_fabric_ca_orderer	replicated	1/1	hyperledger/fabric-ca:1.5.2	*:10054->10054/tcp
11210vuv081f	hyperledger_fabric_ca_org0	replicated	1/1	hyperledger/fabric-ca:1.5.2	*:7054->7054/tcp
ycvrkgfre0dt	hyperledger_fabric_ca_org1	replicated	1/1	hyperledger/fabric-ca:1.5.2	*:8054->8054/tcp
5scuh0t9skqu	hyperledger_fabric_ca_org2	replicated	1/1	hyperledger/fabric-ca:1.5.2	*:9054->9054/tcp

Figure 32: The result of "docker stack services hyperledger\_fabric" command

Pay attention to the number of replicas. If zeroes appear in the first column (e.g., 0/1) means that the service is not up yet or encountered an error.

Note that the CA-server containers can be accessed using the IP of one of the nodes and the ports 7054, 8054, 9054 and 10054 respectively. This happens thanks to Docker Swarm.

The certificates generated after the deployment of the Certificate Authorities must be visible in the following folders (an example depicted in Figure 33):

**blockchain-org0 machine:**

- fabric-deployment/network-setup/organizations/fabric-ca/org0/*
- fabric-deployment/network-setup/organizations/fabric-ca/ordererOrg/*

**blockchain-org1 machine:**

- fabric-deployment/network-setup/organizations/fabric-ca/org1/*

**blockchain-org2 machine:**

- fabric-deployment/network-setup/organizations/fabric-ca/org2/*

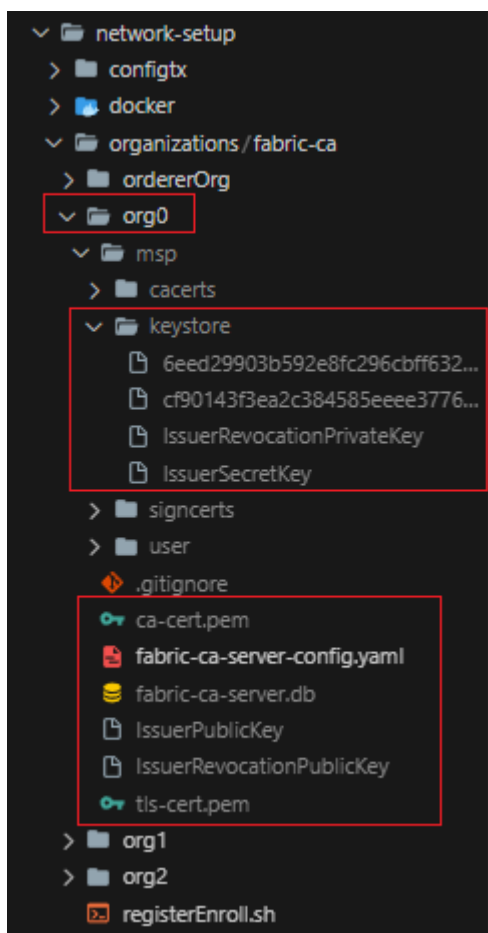


Figure 33: Generated certificates after CA installation for Organization Zero

Indicatively:

- The `ca-cert.pem` is the CA's public certificate. It contains the CA's public key and is used for verifying signatures made by the CA. Other network components and participants can use this certificate to ensure the authenticity of certificates issued by the CA.
- The `tls-cert.pem` file is a certificate file used for securing network communication over the Transport Layer Security (TLS) or its predecessor, Secure Sockets Layer (SSL).
- The `issuerPublicKey` file typically contains the public key of the CA that issued a particular certificate. This information is used by peers and clients to verify the authenticity of certificates in the network. It allows them to validate that a certificate was indeed issued by a trusted CA.
- Similar to `issuerPublicKey`, `issuerRevocationPublicKey` contains the public key used by the CA to sign certificate revocation lists (CRLs). CRLs are important for revoking certificates that have been compromised or are no longer valid, and the public key is needed to verify the CRL's signature.

## 5.3 Secure Digital Communication among Nodes Representing Services

### 5.3.1 Cryptomaterial Generation in the Context of Hyperledger Fabric

In Hyperledger Fabric v2.x, generating **TLS (Transport Layer Security) certificates** for peers and orderers is essential to establish a robust and secure communication framework within the blockchain network. These certificates serve multiple vital purposes. First, TLS certificates ensure data encryption, safeguarding sensitive information by encoding it during transmission and preventing unauthorized access. Second, they play a crucial role in authentication, verifying the identities of network participants and allowing only trusted entities to engage in transactions and interactions. Third, TLS certificates guarantee data integrity, utilizing cryptographic mechanisms to detect any unauthorized alterations or tampering with transmitted data. By securing communication endpoints, TLS certificates thwart potential man-in-the-middle attacks, safeguarding against unauthorized interception and manipulation of data. Industries subject to regulatory compliance benefit from the recognized security standards TLS certificates offer. Moreover, these certificates foster trust and confidence among peers and orderers, as they enable nodes to validate each other's authenticity, establishing a foundation for a reliable and resilient network environment. To sum up, TLS certificates are indispensable tools for fortifying communication channels, ensuring confidentiality, authenticity, and data integrity in Hyperledger Fabric networks.

In this section we will be using client applications against the CA servers instantiated to generate TLS certificates for administrators, users, orderers and peer nodes. Those are necessary for all three Organizations, as well as the five Orderers that also ought to securely communicate among them.

Before doing so, let us examine a few key terms:

- A **Peer Organization**<sup>29</sup> represents a distinct member or entity within a consortium or blockchain network. It comprises a collection of peer nodes that participate in the network's consensus, endorsement, and validation processes. Each Peer Organization maintains its own copy of the distributed ledger and smart contracts (chaincode), enabling decentralized and secure data storage and processing. Peer Organizations collaborate to endorse and validate transactions, contributing to the network's consensus mechanism. They establish trust through cryptographic identity and access management, utilizing certificates and private keys.
- An **Orderer**<sup>30</sup> serves as a critical component responsible for maintaining the chronological order and consistency of transactions across the distributed network. The Orderer plays a pivotal role in achieving consensus among Peer Organizations by collecting endorsed transactions, packaging them into blocks, and then disseminating those blocks to the peers for validation and eventual inclusion in the shared ledger. It ensures that transactions are agreed upon and executed in the same sequence across all network participants, safeguarding against double-spending and maintaining the integrity of the blockchain.
- A **Certificate Authority (CA)** is responsible for issuing and managing cryptographic

<sup>29</sup> Concept of Peers explained in official Hyperledger Fabric documentation: <https://hyperledger-fabric.readthedocs.io/en/latest/peers/peers.html> (accessed October 2023).

<sup>30</sup> Concept of Ordering explained in official Hyperledger Fabric documentation: [https://hyperledger-fabric.readthedocs.io/en/latest/orderer/ordering\\_service.html](https://hyperledger-fabric.readthedocs.io/en/latest/orderer/ordering_service.html) (accessed October 2023).

identities (certificates) for participants within the network. It verifies the authenticity of participants and generates the necessary cryptographic materials, including public-private key pairs and enrollment certificates. CAs ensure secure communication and interactions by providing participants with the means to prove their identity and sign transactions. CAs are integral to establishing the authenticity and security of network participants. There can be multiple CAs in a Fabric network, each responsible for specific organizations or participant groups.

- A **Membership Service Provider (MSP)**<sup>31</sup>, on the other hand, defines the policies and rules that govern the membership of organizations and participants in the network. It specifies which CAs are considered valid within the network and manages the identities and roles of participants associated with a particular organization. The MSP defines access control policies, cryptographic configurations, and other parameters that determine who can access the network and perform specific actions. It's responsible for maintaining trust and access control within the network by enforcing the rules and policies established for each participating organization.
- The **CA server** is the core component responsible for issuing, renewing, and managing cryptographic certificates for participants within the network. It generates public-private key pairs, issues enrollment certificates, and ensures the security of cryptographic materials. The CA server performs identity verification and maintains the integrity of the identity management process. It acts as the central authority that validates and binds the identities of network participants to their cryptographic keys. There can be multiple CA servers in a network, each associated with different organizations or roles.
- A **CA client** is a component that interacts with the CA server to request and manage certificates. It serves as an interface between the participants and the CA server. Participants, such as peers, orderers, and clients, use CA clients to enroll, renew, or revoke their certificates. The CA client is responsible for submitting enrollment requests, providing the necessary information for identity verification, and receiving the resulting enrollment certificates and cryptographic keys. CA clients ensure that participants have the required credentials to interact securely with the network.

In the context of Hyperledger Fabric, **registering and enrolling an identity**<sup>32</sup> are two distinct steps in the process of establishing a participant's identity and access credentials within the blockchain network<sup>33</sup>.

- **Registering an Identity** involves providing initial information about a participant, such as their name, role, and affiliation, to a Certificate Authority (CA). The CA then creates a record for the participant and assigns a unique identifier. This step essentially reserves a spot for the participant within the network, allowing them to proceed with the enrollment process.
- **Enrolling an Identity** is the subsequent step after registration. It involves the actual issuance of cryptographic credentials to the participant. During enrollment, the participant interacts with the CA to authenticate themselves and obtain the necessary cryptographic materials, which include a private key and an enrollment certificate. The private key is used for cryptographic operations like signing transactions, while the

<sup>31</sup> Membership Service Provider concept as analysed in the official Fabric documentation: <https://hyperledger-fabric.readthedocs.io/en/latest/membership/membership.html> (accessed October 2023).

<sup>32</sup> Registering and Enrolling identities with a CA in the official Fabric documentation: [https://hyperledger-fabric-ca.readthedocs.io/en/latest/deployguide/use\\_CA.html](https://hyperledger-fabric-ca.readthedocs.io/en/latest/deployguide/use_CA.html) (accessed October 2023).

<sup>33</sup> Identity concept as discussed in the official Fabric documentation: <https://hyperledger-fabric.readthedocs.io/en/latest/identity/identity.html> (accessed October 2023).

enrollment certificate serves as proof of the participant's identity and is used for authentication when interacting with the network.

### 5.3.2 Generation of Cryptomaterial for the STAR PoC Architecture

In order to generate TLS certificates for administrators, users, orderers and peer nodes we will be using some scripts provided in the installation kit. Those can be found at:

*fabric-deployment/network-setup/organizations/fabric-ca/registerEnroll.sh*

The CA client has the form of a binary located at:

*fabric-deployment/bin/fabric-ca-client*

Once more we ought to navigate within the "network-setup" folder of the "manager" virtual machine. From there we must make sure that the components we need have the proper rights to be accessed and executed by our user:

```
sudo chmod 777 organizations -R
sudo chmod 777 ../bin/fabric-ca-client -R
```

As our script file contains some functions, we may make them executable via the command line thus:

```
source ./organizations/fabric-ca/registerEnroll.sh
```

Also, we must refresh the following environment variable:

```
export PROJECT_DOMAIN="star-ai.eu"
```

Then the following commands will create everything necessary for "Organization 0" and the five Orderers respectively:

```
createOrg0
createOrderer
```

Those two functions **must conclude without errors**.

We then navigate within the "network-setup" folder of the "worker1" virtual machine and respectively type:

```
sudo chmod 777 organizations -R
sudo chmod 777 ../bin/fabric-ca-client -R
export PROJECT_DOMAIN="star-ai.eu"
source ./organizations/fabric-ca/registerEnroll.sh
createOrg1
```

We then navigate within the "network-setup" folder of the "worker2" virtual machine and respectively type:

```
sudo chmod 777 organizations -R
sudo chmod 777 ../bin/fabric-ca-client -R
export PROJECT_DOMAIN="star-ai.eu"
source ./organizations/fabric-ca/registerEnroll.sh
createOrg2
```

### 5.3.3 Verification of Cryptomaterial Generation

The aforementioned script, for each Organization, performs the following:

1. It registers peer 0.
2. It registers a user.
3. It registers the organization admin.
4. It generates the MSP of peer 0.
5. It generates the TLS certificates of peer 0.
6. It generates the MSP of the user.
7. It generates the MSP of the admin.

For the five Orderers, it performs the following:

1. It registers each Orderer.
2. It registers the admin of the Orderers.
3. It generates the MSP of each Orderer.
4. It generates the TLS certificates of each Orderer.
5. It generates the MSP of the admin.

A successful execution of the commands is depicted in Figure 34 below:

```

• amdespotopoulou@blockchain-org0:~/hyperledger-fabric-installation-kit/fabric-deployment/network-setup$ createOrg0
Enroll the CA admin
+ fabric-ca-client enroll -u https://admin:adminpw@localhost:7054 --caname ca-org0 --tls.certfiles /home/amdespotopoulou/hyperledger-fabric-inst
2023/08/14 19:50:17 [INFO] Created a default configuration file at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/n
2023/08/14 19:50:17 [INFO] TLS Enabled
2023/08/14 19:50:17 [INFO] generating key: &{A:ecdsa S:256}
2023/08/14 19:50:17 [INFO] encoded CSR
2023/08/14 19:50:17 [INFO] Stored client certificate at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-setu
2023/08/14 19:50:17 [INFO] Stored root CA certificate at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-set
2023/08/14 19:50:17 [INFO] Stored Issuer public key at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-setup
2023/08/14 19:50:17 [INFO] Stored Issuer revocation public key at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/ne
Register peer0
+ fabric-ca-client register --caname ca-org0 --id.name peer0 --id.secret peer0pw --id.type peer --tls.certfiles /home/amdespotopoulou/hyperledge
2023/08/14 19:50:17 [INFO] Configuration file location: /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-setu
2023/08/14 19:50:17 [INFO] TLS Enabled
2023/08/14 19:50:17 [INFO] TLS Enabled
Password: peer0pw
Register user
+ fabric-ca-client register --caname ca-org0 --id.name user1 --id.secret user1pw --id.type client --tls.certfiles /home/amdespotopoulou/hyperled
2023/08/14 19:50:17 [INFO] Configuration file location: /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-setu
2023/08/14 19:50:17 [INFO] TLS Enabled
2023/08/14 19:50:17 [INFO] TLS Enabled
Password: user1pw
Register the org admin
+ fabric-ca-client register --caname ca-org0 --id.name org0admin --id.secret org0adminpw --id.type admin --tls.certfiles /home/amdespotopoulou/h
2023/08/14 19:50:17 [INFO] Configuration file location: /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-setu
2023/08/14 19:50:17 [INFO] TLS Enabled
2023/08/14 19:50:17 [INFO] TLS Enabled
Password: org0adminpw
Generate the peer0 msp
+ fabric-ca-client enroll -u https://peer0:peer0pw@localhost:7054 --caname ca-org0 -M /home/amdespotopoulou/hyperledger-fabric-installation-kit/
sts peer0.org0.star-ai.com --tls.certfiles /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-setup/organizatio
2023/08/14 19:50:18 [INFO] TLS Enabled
2023/08/14 19:50:18 [INFO] generating key: &{A:ecdsa S:256}
2023/08/14 19:50:18 [INFO] encoded CSR
2023/08/14 19:50:18 [INFO] Stored client certificate at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-setu
2023/08/14 19:50:18 [INFO] Stored root CA certificate at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-set
2023/08/14 19:50:18 [INFO] Stored Issuer public key at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-setup
2023/08/14 19:50:18 [INFO] Stored Issuer revocation public key at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/ne
Generate the peer0-tls certificates
+ fabric-ca-client enroll -u https://peer0:peer0pw@localhost:7054 --caname ca-org0 -M /home/amdespotopoulou/hyperledger-fabric-installation-kit/
ment.profile tls --csr.hosts peer0.org0.star-ai.com --csr.hosts localhost --tls.certfiles /home/amdespotopoulou/hyperledger-fabric-installation-
2023/08/14 19:50:18 [INFO] TLS Enabled
2023/08/14 19:50:18 [INFO] generating key: &{A:ecdsa S:256}
2023/08/14 19:50:18 [INFO] encoded CSR
2023/08/14 19:50:18 [INFO] Stored client certificate at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-setu
2023/08/14 19:50:18 [INFO] Stored TLS root CA certificate at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network
org0.pem
2023/08/14 19:50:18 [INFO] Stored Issuer public key at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-setup
2023/08/14 19:50:18 [INFO] Stored Issuer revocation public key at /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/ne

```

Figure 34: A successful execution of the functions enclosed in the registerEnroll.sh script

Provided everything finishes without error, the “manager” node must have now under “network-setup/organizations” two new folders (see *Figure 35*):

- \* ordererOrganizations
- \* peerOrganizations/org0.star-ai.eu

Similarly, worker1 now has:

- \* ordererOrganizations
- \* peerOrganizations/org1.star-ai.eu

Similarly, worker2 now has:

- \* ordererOrganizations
- \* peerOrganizations/org2.star-ai.eu

Those contain in sub-folders all the material needed for secure interactions between the components of the Hyperledger Fabric network.

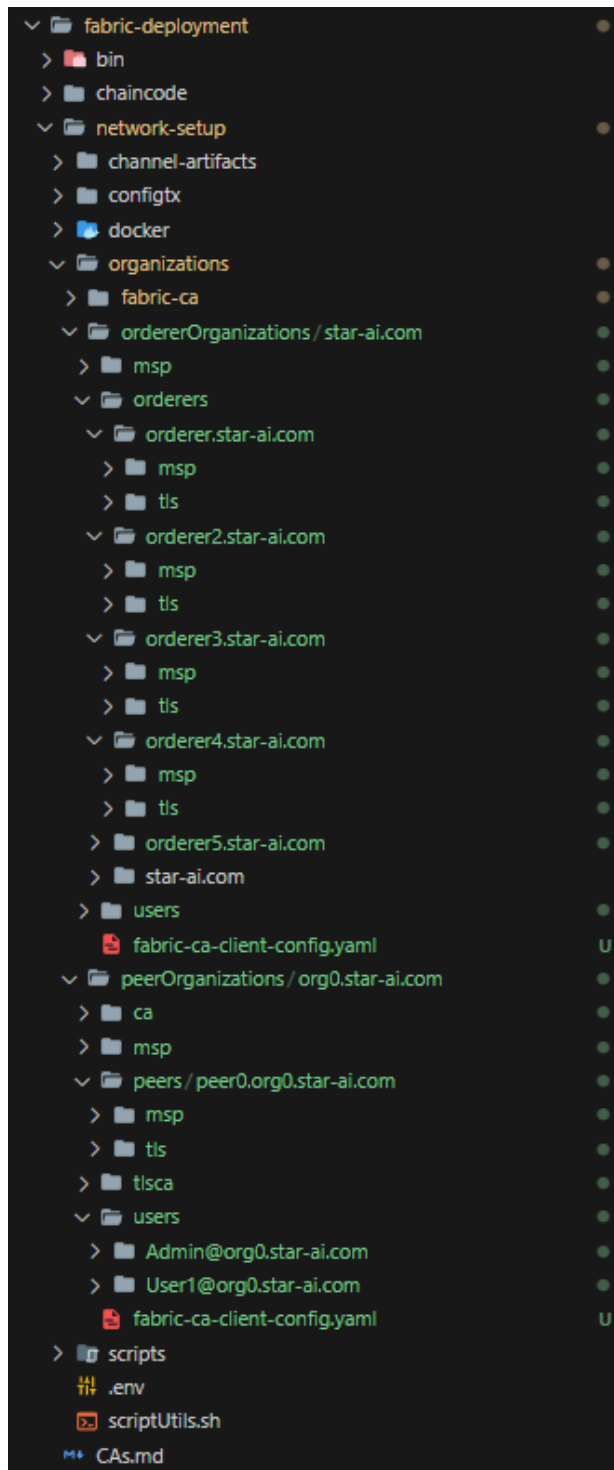


Figure 35: Result of certificates creation script on "manager" node

### 5.3.4 Dissemination of Identities among Nodes of the STAR PoC Network

As a final step we have to copy some of the certificates among the machines.

First, from the "worker1" node we copy the folder:

*/fabric-deployment/network-setup/organizations/peerOrganizations/org1.star-ai.eu/*

and place it on the “manager” node again as:

```
/fabric-deployment/network-setup/organizations/peerOrganizations/org1.star-ai.eu
```

Second, from the “worker2” node we copy the folder:

```
/fabric-deployment/network-setup/organizations/peerOrganizations/org2.star-ai.eu
```

and place it on the “manager” node again as:

```
/fabric-deployment/network-setup/organizations/peerOrganizations/org2.star-ai.eu
```

Then, from the “manager” node we copy the folder:

```
/fabric-deployment/network-setup/organizations/ordererOrganizations/
```

and place it on both “worker1” and “worker2” nodes as:

```
/fabric-deployment/network-setup/organizations/ordererOrganizations/
```

## 5.4 Bootstrapping the Blockchain

### 5.4.1 The Concept of the Orderer Genesis Block in Hyperledger Fabric

In the intricate landscape of Hyperledger Fabric, an **Orderer Genesis Block** assumes a pivotal role by serving as the cornerstone of the entire network infrastructure. This block encapsulates a wealth of critical information, encompassing not only the identity and cryptographic particulars of the Orderer organization(s) but also a compendium of network-wide settings that orchestrate the precise functioning of the Hyperledger Fabric network.

At its core, the Orderer Genesis Block lays down the fundamental framework upon which the Orderer service operates, addressing essential aspects such as consensus algorithms. The block harbors the secrets of how consensus is to be achieved among the distributed Orderer nodes, thus establishing the core agreement mechanism that ensures the authenticity and consistency of transactions within the network.

Crafting the Orderer Genesis Block is an indispensable step during the setup and initiation of the Hyperledger Fabric network. It is the genesis of trust, as it forms the basis for all future transactions and ledger updates. Once created, this foundational block becomes the keystone for bootstrapping the network, effectively enabling the Orderer nodes to kickstart their essential roles in transaction processing and dissemination. Moreover, the Orderer Genesis Block assumes an ongoing role in the network's operation. It plays an instrumental role in preserving the sanctity and order of transactions flowing through the Hyperledger Fabric network. By serving as a beacon of integrity, it ensures that each transaction is meticulously recorded and arranged in a tamper-proof and chronologically accurate manner, upholding the principles of immutability and transparency that define blockchain technology.

In this section we will be documenting such a **generation of an Orderer Genesis Block**.

### 5.4.2 Creating the Genesis Block for the STAR PoC Ordering Service

Returning to our architectural diagram (Figure 28) we may trace the placement of the five components acting as Orderers within the three virtual machines. Out of choice they are all hosted on “manager” node as is the entire service along with its corresponding CA. The respective Docker containers are represented with the colour gold.

In Hyperledger Fabric, it's generally recommended to distribute Orderer nodes across multiple physical or virtual machines (VMs) and, if possible, across different organizations for the sake of decentralization, fault tolerance, and security. Ours is just a PoC, but in a production-level system, one with multiple organisations each hosted on its proper virtual machine, we could have chosen a subset of them to host an Orderer component. In the context of Hyperledger Fabric, the number of Orderer nodes does not necessarily have to be odd, nor is there a strict requirement for a specific number of such nodes. Unlike some consensus algorithms where odd numbers are preferred to avoid tie-breaker scenarios, Hyperledger Fabric employs a pluggable consensus mechanism, allowing you to choose the consensus algorithm that best suits your network's requirements.

Towards the end of generating our genesis block we will be using some scripts provided in the installation kit. Those can be found at:

*fabric-deployment/network-setup/scripts/createGenesis.sh*

The script employs a binary located at:

*fabric-deployment/bin/configtxgen*

The genesis block will be created using the console of the "manager" virtual machine and, in particular, from the "network-setup" folder. From there we must make sure that the components we need have the proper rights to be accessed and executed by our user:

```
sudo chmod 777 scripts -R
sudo chmod 777 ../bin/configtxgen -R
```

There is an additional configuration file that we have to generate using the command:

```
sed 's/${PROJECT_DOMAIN}/star-ai.eu/g' configtx/configtx_matrix.yaml >
configtx/configtx.yaml
```

This command produces the following file:

*fabric-deployment/network-setup/configtx/configtx.yaml*

The information on the network architecture to be included in the genesis block are included in this file as depicted in Figure 36 below:

```
Profiles:

TwoOrgsOrdererGenesis:
  <<: *ChannelDefaults
  Orderer:
    <<: *OrdererDefaults
    Organizations:
      - *OrdererOrg
    Capabilities:
      <<: *OrdererCapabilities
    Consortiums:
      SampleConsortium:
        Organizations:
          - *Org0
          - *Org1
          - *Org2
```

Figure 36: Configuration for the Genesis Block from configtx.yaml

Continuing in the "network-setup" folder of the "manager", we execute the script:

```
./scripts/createGenesis.sh
```

A successful result looks like this (Figure 37):

```
amdespotopoulou@blockchain-org0:~/hyperledger-fabric-installation-kit/fabric-deployment/network-setup$ ./scripts/createGenesis.sh
/home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-setup/./bin/configtxgen
Generating Orderer Genesis block
++ configtxgen -profile TwoOrgsOrdererGenesis -channelID system-channel -outputBlock ./system-genesis-block/gensis.block
2023-08-15 13:10:46.211 UTC 0001 INFO [common.tools.configtxgen] main -> Loading configuration
2023-08-15 13:10:46.226 UTC 0002 INFO [common.tools.configtxgen.localconfig] completeInitialization -> orderer type: etcdraft
2023-08-15 13:10:46.226 UTC 0003 INFO [common.tools.configtxgen.localconfig] load -> Loaded configuration: /home/amdespotopoulou/hyperledger-fabric-installation-kit/fabric-deployment/network-setup/configtx/configtx.yaml
2023-08-15 13:10:46.228 UTC 0005 INFO [common.tools.configtxgen] doOutputBlock -> Generating genesis block
2023-08-15 13:10:46.228 UTC 0006 INFO [common.tools.configtxgen] doOutputBlock -> Creating system channel genesis block
2023-08-15 13:10:46.229 UTC 0007 INFO [common.tools.configtxgen] doOutputBlock -> Writing genesis block
++ res=0
```

Figure 37: Successful creation of the Genesis Block in the console

The genesis block is visible in the "manager" node under the newly created "system-genesis-block" folder (Figure 38):

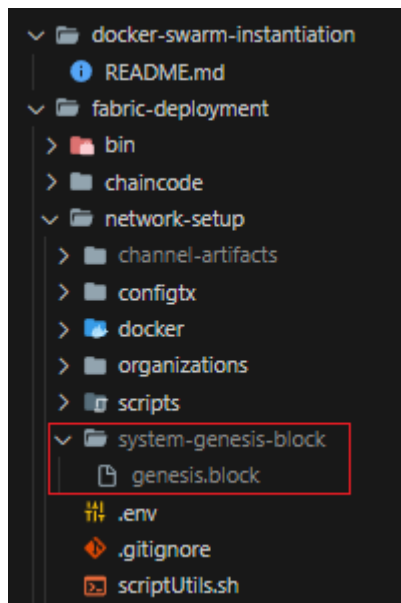


Figure 38: Genesis block within the Manager node folder structure

## 5.5 Configuration of Private Data Channels between Network Participants

### 5.5.1 The Concept of Channels and Anchor Peers in Hyperledger Fabric

Hyperledger Fabric, as a permissioned blockchain protocol, enables the creation of private data channels between network participants to maintain data confidentiality and access control. This feature is especially valuable in business and consortium blockchain networks where different entities want to transact while preserving the privacy of their sensitive information. In Hyperledger Fabric, each network participant is identified and authenticated, and the permission to access and transact with data is controlled through an endorsement policy. Private data channels are a mechanism to further restrict data access to a subset of network participants, in our case a selection of services from those formulating the STAR platform.

In Fabric, a **Channel Genesis Block** is the initial and foundational block that marks the starting point of transactions within a specific channel. A channel represents a private communication pathway within a Hyperledger Fabric network, allowing a subset of network participants to transact and share data privately. The Channel Genesis Block encapsulates essential configuration details specific to the channel, such as the identities of participating organizations, their cryptographic materials, access control policies, and any initial data that needs to be included in the ledger. This block ensures that all participants within the channel share a common starting point and configuration, facilitating consensus and verification of transactions within that isolated context. As new transactions are endorsed and committed within the channel, they extend from the Channel Genesis Block, forming a sequential and tamper-evident ledger unique to that channel's participants<sup>34</sup>.

On the other hand, **Anchor Peers** refer to the specific peers within an organization that are designated to facilitate cross-organization communication and discovery within a particular channel. Each participating organization within a channel typically selects one or more of its peers to serve as anchor peers. These anchor peers act as stable communication endpoints and provide a point of contact for peers in other organizations to establish connections and share important information. Anchor peers play a vital role in maintaining the connectivity and coordination between organizations, ensuring that transactions and data can be efficiently propagated and exchanged across the network. They are used during endorsement, transaction propagation, and query processes, enabling efficient communication and enhancing the overall resilience and responsiveness of the Hyperledger Fabric network<sup>35</sup>.

In the following section we will be **creating three channels**, one per use case, using a specified configuration transaction and communicating with the orderer service to establish the channel. To be precise we will be generating channel creation transaction artifacts using a specified channel profile (TwoOrgsChannel) as part of the process to create a new channel in Fabric. Then, we will be generating a genesis block configuration artifact for the system channel as part of the process to create the initial configuration for the orderer nodes in the network. Finally, we will be updating Anchor Peer nodes (peer0.org1 and peer0.org2).

---

<sup>34</sup> Hyperledger Fabric official documentation regarding channel creation available at: [https://hyperledger-fabric.readthedocs.io/en/latest/create\\_channel/create\\_channel\\_overview.html](https://hyperledger-fabric.readthedocs.io/en/latest/create_channel/create_channel_overview.html) (accessed October 2023).

<sup>35</sup> Anchor Peers explained at Hyperledger Fabric official documentation: <https://hyperledger-fabric.readthedocs.io/en/latest/gossip.html#anchor-peers> (accessed October 2023).

## 5.5.2 Instantiation of Channels for STAR Industrial Metadata Use Cases

Towards the end of generating our genesis block we will be using a script provided in the installation kit. This can be found at:

*fabric-deployment/network-setup/scripts/createChannelTx.sh*

The script employs a binary located at:

*fabric-deployment/bin/configtxgen*

The genesis block will be created using the console of the “manager” virtual machine and in particular from the “network-setup” folder. From there we must make sure that the components we need have the proper rights to be accessed and executed by our user (might already be ready from a previous step):

```
sudo chmod 777 scripts -R
sudo chmod 777 ../bin/configtxgen -R
```

The information on the channel configuration to be included in the genesis block is included in this file:

*fabric-deployment/network-setup/configtx/configtx.yaml*

For our PoC all three organizations are joining our channels (even though this is not necessary; it is a business decision) as depicted in Figure 39:

```
TwoOrgsChannel:
  Consortium: SampleConsortium
  <<: *ChannelDefaults
  Application:
    <<: *ApplicationDefaults
    Organizations:
      - *Org0
      - *Org1
      - *Org2
    Capabilities:
      <<: *ApplicationCapabilities
```

*Figure 39: Configuration for the Channel from configtx.yaml*

To create this file one has to use a matrix file and configure the projects domain name, something already done in a previous step.

Continuing in the “network-setup” folder of the “manager”, we execute three times, one per use case, the script:

*./scripts/createChannelTx.sh*

What ought to be changed manually between executions is a small change in line 10 (Figure 40). There instead of the sample “mychannel”, we use instead consecutively:

datasourceschannel / processorschannel / observationschannel

```

6 CHANNEL_NAME="$1"
7 DELAY="$2"
8 MAX_RETRY="$3"
9 VERBOSE="$4"
10 : ${CHANNEL_NAME:="mychannel"}
11 : ${DELAY:="3"}
12 : ${MAX_RETRY:="5"}
13 : ${VERBOSE:="true"}

```

Figure 40: "mychannel" on line 10 must be changed with actual channel name

A successful result, with "mychannel" as a demonstrator is depicted below (Figure 41):

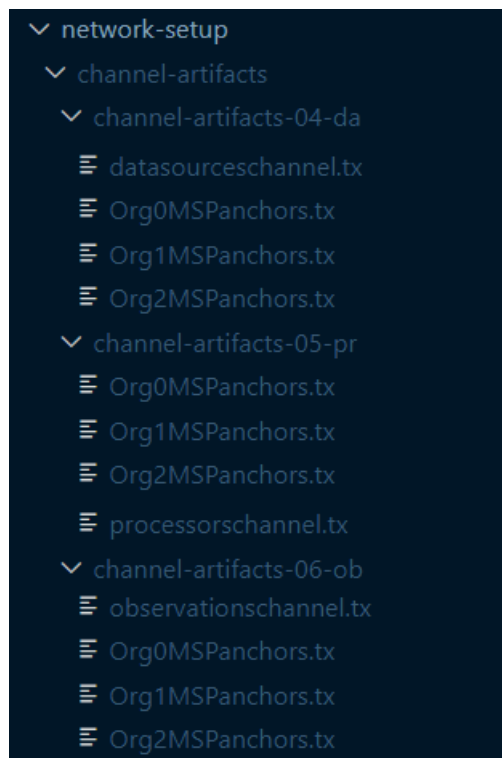
```

amdespotopoulou@blockchain-org0:~/hyperledger-fabric-installation-kit/fabric-deployment/network-setup$ ./scripts/createChannelTx.sh
Generating channel create transaction 'mychannel.tx'
+ configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/mychannel.tx -channelID mychannel
2023-08-16 11:32:31.911 UTC 0001 INFO [common.tools.configtxgen] main -> Loading configuration
2023-08-16 11:32:31.930 UTC 0002 INFO [common.tools.configtxgen.localconfig] Load -> Loaded configuration: /home/amdespotopoulou/hyperled
2023-08-16 11:32:31.930 UTC 0003 INFO [common.tools.configtxgen] doOutputChannelCreateTx -> Generating new channel configtx
2023-08-16 11:32:31.933 UTC 0004 INFO [common.tools.configtxgen] doOutputChannelCreateTx -> Writing new channel tx
+ res=0
Generating anchor peer update transactions
Generating anchor peer update transaction for Org0MSP
+ configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org0MSPanchors.tx -channelID mychannel -asOrg Org0MSP
2023-08-16 11:32:31.968 UTC 0001 INFO [common.tools.configtxgen] main -> Loading configuration
2023-08-16 11:32:31.983 UTC 0002 INFO [common.tools.configtxgen.localconfig] Load -> Loaded configuration: /home/amdespotopoulou/hyperled
2023-08-16 11:32:31.983 UTC 0003 INFO [common.tools.configtxgen] doOutputAnchorPeersUpdate -> Generating anchor peer update
2023-08-16 11:32:31.985 UTC 0004 INFO [common.tools.configtxgen] doOutputAnchorPeersUpdate -> Writing anchor peer update
+ res=0
Generating anchor peer update transaction for Org1MSP
+ configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID mychannel -asOrg Org1MSP
2023-08-16 11:32:32.030 UTC 0001 INFO [common.tools.configtxgen] main -> Loading configuration
2023-08-16 11:32:32.045 UTC 0002 INFO [common.tools.configtxgen.localconfig] Load -> Loaded configuration: /home/amdespotopoulou/hyperled
2023-08-16 11:32:32.045 UTC 0003 INFO [common.tools.configtxgen] doOutputAnchorPeersUpdate -> Generating anchor peer update
2023-08-16 11:32:32.047 UTC 0004 INFO [common.tools.configtxgen] doOutputAnchorPeersUpdate -> Writing anchor peer update
+ res=0
Generating anchor peer update transaction for Org2MSP
+ configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID mychannel -asOrg Org2MSP
2023-08-16 11:32:32.089 UTC 0001 INFO [common.tools.configtxgen] main -> Loading configuration
2023-08-16 11:32:32.118 UTC 0002 INFO [common.tools.configtxgen.localconfig] Load -> Loaded configuration: /home/amdespotopoulou/hyperled
2023-08-16 11:32:32.118 UTC 0003 INFO [common.tools.configtxgen] doOutputAnchorPeersUpdate -> Generating anchor peer update
2023-08-16 11:32:32.121 UTC 0004 INFO [common.tools.configtxgen] doOutputAnchorPeersUpdate -> Writing anchor peer update
+ res=0

```

Figure 41: Successful creation of a Channel in the console

The genesis block and three transaction files are visible in the "manager node" under the newly created "channel-artifacts" folder. After each execution, one per channel, the four created artifacts must be manually placed in a subfolder in order not to be overwritten by the next. Therefore, we end up with a file structure for the "manager" that resembles like this (Figure 42):



*Figure 42: Channel artifacts within the Manager node folder structure*

Last but not least, we then proceed **to copy** the folder “channel-artifacts” and its contents to the other two machines, in the exact same position under “network-setup”. Thus, we are keeping our anchor peers up to date.

**Note:** Up to now we have just created material for our Channels. Those will be instantiated later.

## 5.6 Joining Nodes to the Network and Global State Maintenance

### 5.6.1 Key Participants formulating a Hyperledger Fabric Network

In Hyperledger Fabric, **Organizations, Peers, and Orderers** play distinct yet interconnected roles within the network architecture. **Organizations** represent the distinct entities that participate in the network, often corresponding to real-world entities like businesses or institutions. Each organization possesses its own set of peers, policies, and identities. **Peers**, on the other hand, are the nodes within an organization that maintain and endorse the blockchain ledger. They can be of two types: **endorsing peers** and **committing peers**. Endorsing peers execute chaincode (smart contracts) and provide endorsements for transactions. When a transaction is proposed to the network, it is sent to endorsing peers. These peers simulate the transaction, verify its legitimacy, and endorse it by signing the results. These endorsements essentially serve as a vote of confidence in the proposed transaction's validity. Committing peers, from their part, receive endorsed transactions from endorsing peers and are responsible for ordering them into blocks. They validate the endorsements and ensure that the transaction complies with the channel's policies. Once a block is created, committing peers engage in a consensus process to determine the order of transactions and then commit them to the ledger. This ensures the integrity of the ledger and makes the transactions irreversible.

**Orderers**, often referred to as the **ordering service**, are responsible for managing the sequence of transactions and ensuring consensus across the network. They create blocks of ordered transactions, which are then disseminated to the peers for validation and commitment. By separating the endorsement and ordering processes, Hyperledger Fabric achieves a high degree of modularity and scalability. Organizations can maintain their autonomy while participating in a shared network, and consensus is achieved through a pluggable consensus mechanism, which can be adapted to fit the specific requirements of the use case.

### 5.6.2 The Concept of Ordering

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 43 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, 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].

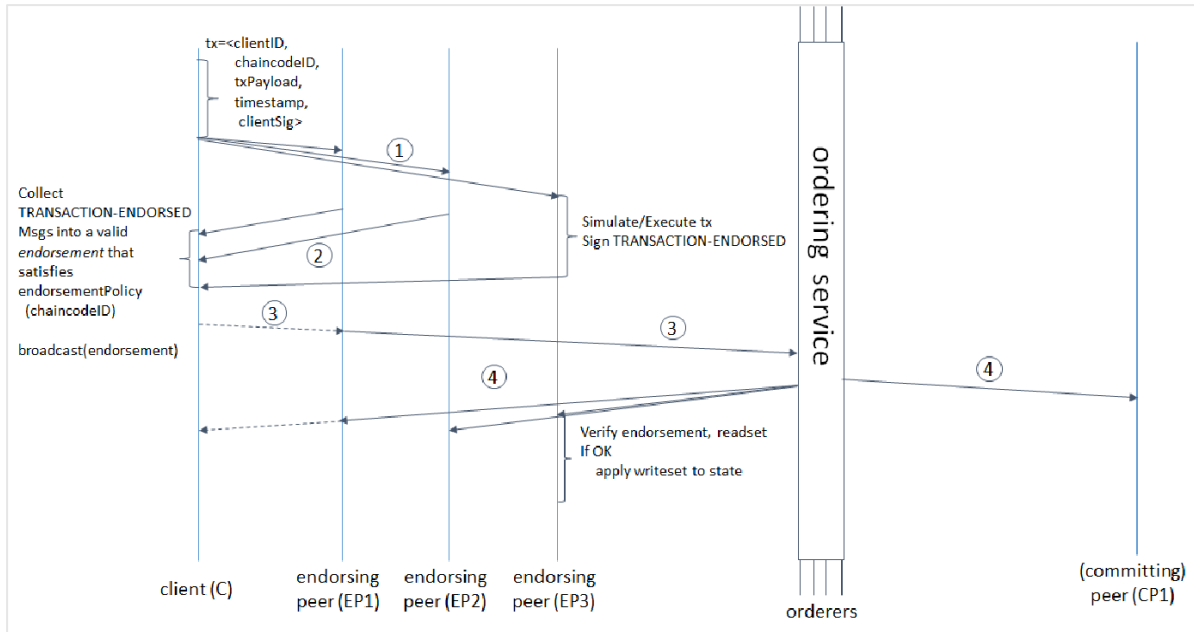


Figure 43: Interfacing to the HLF Ordering Service [FabricDocs]

### 5.6.3 Maintenance of a Common Global State in Fabric

Hyperledger Fabric achieves the maintenance of a common global state through a combination of technical mechanisms. This common global state is a fundamental feature in distributed ledger technologies, ensuring that all peers within a network have the same view of the ledger.

In Hyperledger Fabric, the ledger is divided into two main components: the world state and the transaction log. The world state represents the current state of the ledger, while the transaction log records all historical transactions.

The key to maintaining a common global state in Hyperledger Fabric is the use of a world state database, often implemented using CouchDB<sup>36</sup>, an open-source document-oriented NoSQL<sup>37</sup> database. CouchDB serves as a versatile and decentralized storage layer that holds the world state of the blockchain network, complementing the transactional ledger maintained by the peers. Each peer in the network maintains its local copy of the world state, which is essentially a snapshot of the ledger's current state. The latter stores the current values of assets and entities in a structured manner, allowing for efficient querying, indexing, and rich data modeling. By employing a distributed and decentralized architecture, CouchDB ensures high availability, fault tolerance, and data immutability.

When transactions are processed and committed to the ledger, the changes to the world state are recorded in CouchDB. Each peer updates its local copy of the world state accordingly. This synchronization ensures that all peers have the same view of the ledger's current state. Furthermore, one of the unique features of using CouchDB for the world state is its ability to support rich queries. This allows for efficient data retrieval and querying, which is crucial for

<sup>36</sup> Official website of Apache CouchDB: <https://couchdb.apache.org/> (accessed October 2023).

<sup>37</sup> Explanation of the concept of NoSQL databases by MongoDB developers: <https://www.mongodb.com/nosql-explained> (accessed October 2023).

enterprise applications. When a transaction proposal is made, endorsing peers use CouchDB to validate and endorse transactions based on their current view of the world state<sup>38</sup>.

CouchDB, when used as the world state database in the context of Hyperledger Fabric, **is not inherently immutable**. While Fabric's ledger itself is designed to be immutable, as it records all historical transactions and ensures their integrity, CouchDB, being a NoSQL database, does not offer the same level of immutability by default. It is capable of overwriting or updating data if allowed by the access control policies and application logic. To ensure immutability of the data stored in CouchDB, Fabric relies on cryptographic hashing and digital signatures to prevent unauthorized changes to the ledger. The historical record of transactions in the Fabric ledger provides the immutable history, while CouchDB primarily serves as a means for efficiently querying and accessing the current world state. So, while CouchDB may not be inherently immutable, it plays a crucial role in ensuring the integrity of the ledger by facilitating efficient access to the current state of the blockchain.

An alternative approach to maintaining a common global state is to use a state-based replication model. In this model, the entire state is replicated across all peers in the network, ensuring that each peer has an identical copy of the entire ledger state. This approach is straightforward but can be resource-intensive, especially in large networks, as it requires replicating and synchronizing the entire state across all peers.

#### 5.6.4 Launching the Network for the STAR PoC Architecture

In the present section we will be discussing the **deployment** of three Peer nodes, one per Organization (virtual machine)<sup>39</sup>. A CouchDB instance will be maintaining global state in each. In addition, five Orderers will be deployed in the machine hosting Organization Zero.

**Note:** All instantiations shall take place via the command line of the "manager" node. Swarm and the .yaml configuration file will handle the distribution of Docker containers within the cluster.

The desired result, given that the CAs have already been deployed, is to achieve the following distribution of containers (Figure 44):

---

<sup>38</sup> The official Hyperledger Fabric documentation on the usage of CouchDB: [https://hyperledger-fabric.readthedocs.io/en/latest/couchdb\\_tutorial.html](https://hyperledger-fabric.readthedocs.io/en/latest/couchdb_tutorial.html) (accessed October 2023).

<sup>39</sup> Deployment of Peers and Ordering Nodes as described in Hyperledger Fabric's official documentation: [https://hyperledger-fabric.readthedocs.io/en/latest/deployment\\_guide\\_overview.html#step-five-deploy-peers-and-ordering-nodes](https://hyperledger-fabric.readthedocs.io/en/latest/deployment_guide_overview.html#step-five-deploy-peers-and-ordering-nodes) (accessed October 2023).

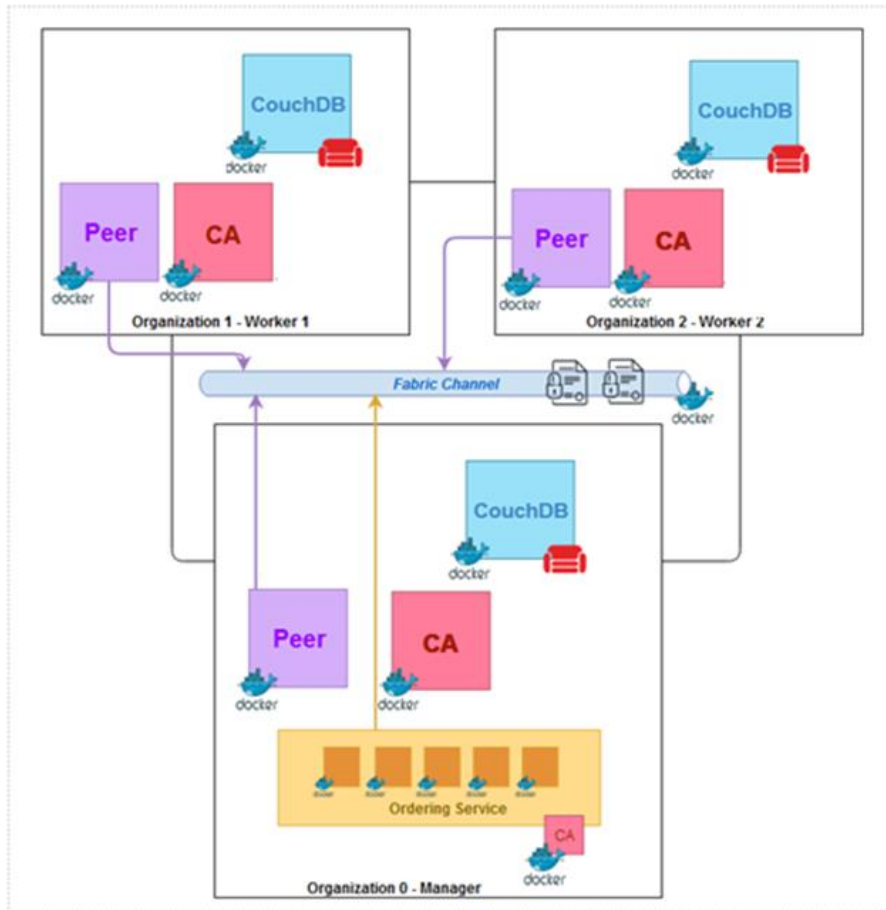


Figure 44: Distribution of Docker Containers in VMs

Using the command line of the “manager” node, we navigate as usual to the directory:

*fabric-deployment/network-setup/*

Once there, we proceed by preparing two of our configuration (docker-compose) files by updating two matrices with the project’s domain name:

```
sed -e 's/${PROJECT_DOMAIN}/star-ai.eu/g' -e 's/${SHORT_DOMAIN}/staraieu/g'
docker/docker-compose-couch-matrix.yaml > docker/docker-compose-couch.yaml
```

```
sed -e 's/${PROJECT_DOMAIN}/star-ai.eu/g' -e 's/${SHORT_DOMAIN}/staraieu/g'
docker/docker-compose-network-matrix.yaml > docker/docker-compose-network.yaml
```

Finally, we proceed to actually deploy the Docker containers using the command:

```
docker stack deploy --with-registry-auth -c docker/docker-compose-network.yaml -c docker/docker-compose-couch.yaml hyperledger_fabric
```

The visible result is depicted below (Figure 45):

```

● amdespotopoulou@blockchain-org0:~/hyperledger-fabric-installation-kit/fabric-deployment/net
work-setup$ docker stack deploy --with-registry-auth -c docker/docker-compose-network.yaml
-c docker/docker-compose-couch.yaml hyperledger_fabric
Ignoring deprecated options:

container_name: Setting the container name is not supported.

Creating service hyperledger_fabric_couchdb0
Creating service hyperledger_fabric_couchdb1
Creating service hyperledger_fabric_orderer2staraicom
Creating service hyperledger_fabric_ordererstaraicom
Creating service hyperledger_fabric_peer0org0staraicom
Creating service hyperledger_fabric_couchdb2
Creating service hyperledger_fabric_orderer3staraicom
Creating service hyperledger_fabric_orderer4staraicom
Creating service hyperledger_fabric_orderer5staraicom
Creating service hyperledger_fabric_peer0org1staraicom
Creating service hyperledger_fabric_peer0org2staraicom
    
```

Figure 45: The result of stack deployment through the console of the "manager"

To make sure that everything is in place, we can type at the console of our "manager":

```
docker stack ps hyperledger_fabric
```

The result is illustrated in Figure 46 below:

```

● amdespotopoulou@blockchain-org0:~/hyperledger-fabric-installation-kit$ docker stack ps hyperledger_fabric
ID                NAME                                IMAGE                                NODE                DESIRED STATE
wfrovnv0d3j4     hyperledger_fabric_ca_orderer.1    hyperledger/fabric-ca:1.5.2        blockchain-org0     Running
te66hni1i6xb     hyperledger_fabric_ca_org0.1       hyperledger/fabric-ca:1.5.2        blockchain-org0     Running
mldk9b6hpg5r     hyperledger_fabric_ca_org1.1       hyperledger/fabric-ca:1.5.2        blockchain-org1     Running
la8viccs3b03     hyperledger_fabric_ca_org2.1       hyperledger/fabric-ca:1.5.2        blockchain-org2     Running
ga4sq6yykwr6     hyperledger_fabric_couchdb0.1      couchdb:3.2.2                      blockchain-org0     Running
dkvotf1suu2e     hyperledger_fabric_couchdb1.1      couchdb:3.2.2                      blockchain-org1     Running
uh10r91ct5hy     hyperledger_fabric_couchdb2.1      couchdb:3.2.2                      blockchain-org2     Running
fj5aaaj28bke7    hyperledger_fabric_orderer2staraicom.1 hyperledger/fabric-orderer:2.3      blockchain-org0     Running
x8fx30vkdkej     hyperledger_fabric_orderer3staraicom.1 hyperledger/fabric-orderer:2.3      blockchain-org0     Running
lmgaaacz60xuf    hyperledger_fabric_orderer4staraicom.1 hyperledger/fabric-orderer:2.3      blockchain-org0     Running
4kb8mcb5fsq1     hyperledger_fabric_orderer5staraicom.1 hyperledger/fabric-orderer:2.3      blockchain-org0     Running
xjypsnszve7w3    hyperledger_fabric_ordererstaraicom.1 hyperledger/fabric-orderer:2.3      blockchain-org0     Running
r2pegar5jeou     hyperledger_fabric_peer0org0staraicom.1 hyperledger/fabric-peer:2.3         blockchain-org0     Running
xdzrrx3tulrd     hyperledger_fabric_peer0org1staraicom.1 hyperledger/fabric-peer:2.3         blockchain-org1     Running
z28lil38mk95     hyperledger_fabric_peer0org2staraicom.1 hyperledger/fabric-peer:2.3         blockchain-org2     Running
    
```

Figure 46: The result of "docker stack ps hyperledger\_fabric" command

Alternatively, to verify the services that are part of the deployed stack one may type:

```
docker stack services hyperledger_fabric
```

The result is illustrated in Figure 47 below:

```

amdespotopoulou@blockchain-org0:~/hyperledger-fabric-installation-kit$ docker stack services hyperledger_fabric
ID                NAME                                MODE          REPLICAS  IMAGE                                PORTS
rxg57x7h05er     hyperledger_fabric_ca_orderer      replicated    1/1       hyperledger/fabric-ca:1.5.2        *:10054->10054/tcp
11210uvu081f     hyperledger_fabric_ca_org0         replicated    1/1       hyperledger/fabric-ca:1.5.2        *:7054->7054/tcp
ycvrkgrfre0dt    hyperledger_fabric_ca_org1         replicated    1/1       hyperledger/fabric-ca:1.5.2        *:8054->8054/tcp
5scuh0t9skqu     hyperledger_fabric_ca_org2         replicated    1/1       hyperledger/fabric-ca:1.5.2        *:9054->9054/tcp
c5j6e1508qdx     hyperledger_fabric_couchdb0        replicated    1/1       couchdb:3.2.2                       *:5984->5984/tcp
y0fsh3n5n12s     hyperledger_fabric_couchdb1        replicated    1/1       couchdb:3.2.2                       *:7984->5984/tcp
m7ybrgysvcuv     hyperledger_fabric_couchdb2        replicated    1/1       couchdb:3.2.2                       *:9984->5984/tcp
qoqop0qhpbuu     hyperledger_fabric_orderer2staraicom replicated    1/1       hyperledger/fabric-orderer:2.3      *:8050->7050/tcp
vxbyofqz3qzq     hyperledger_fabric_orderer3staraicom replicated    1/1       hyperledger/fabric-orderer:2.3      *:9050->7050/tcp
ic27uu3a1ddu     hyperledger_fabric_orderer4staraicom replicated    1/1       hyperledger/fabric-orderer:2.3      *:10050->7050/tcp
jdk06nzvcdyw     hyperledger_fabric_orderer5staraicom replicated    1/1       hyperledger/fabric-orderer:2.3      *:11050->7050/tcp
nqxehbwi3wvh     hyperledger_fabric_ordererstaraicom replicated    1/1       hyperledger/fabric-orderer:2.3      *:7050->7050/tcp
t9n2naaxuru3     hyperledger_fabric_peer0org0staraicom replicated    1/1       hyperledger/fabric-peer:2.3        *:7051->9051/tcp
p4ryv2ro6lrx     hyperledger_fabric_peer0org1staraicom replicated    1/1       hyperledger/fabric-peer:2.3        *:9051->9051/tcp
lb8t2ppw15e8     hyperledger_fabric_peer0org2staraicom replicated    1/1       hyperledger/fabric-peer:2.3        *:11051->11051/tcp
  
```

Figure 47: The result of "docker stack services hyperledger\_fabric" command

One must pay attention to the number of replicas. If zeroes appear in the respective column (i.e., 0/1) means that the service is not up yet, and a few seconds or minutes must elapse before moving further with the deployment procedure.

The deployed services are also depicted in the following screenshot from the Portainer dashboard we have deployed for monitoring the cluster (Figure 48 and Figure 49). Naturally, this can be consulted anytime for resolving issues and not exclusively during deployment.

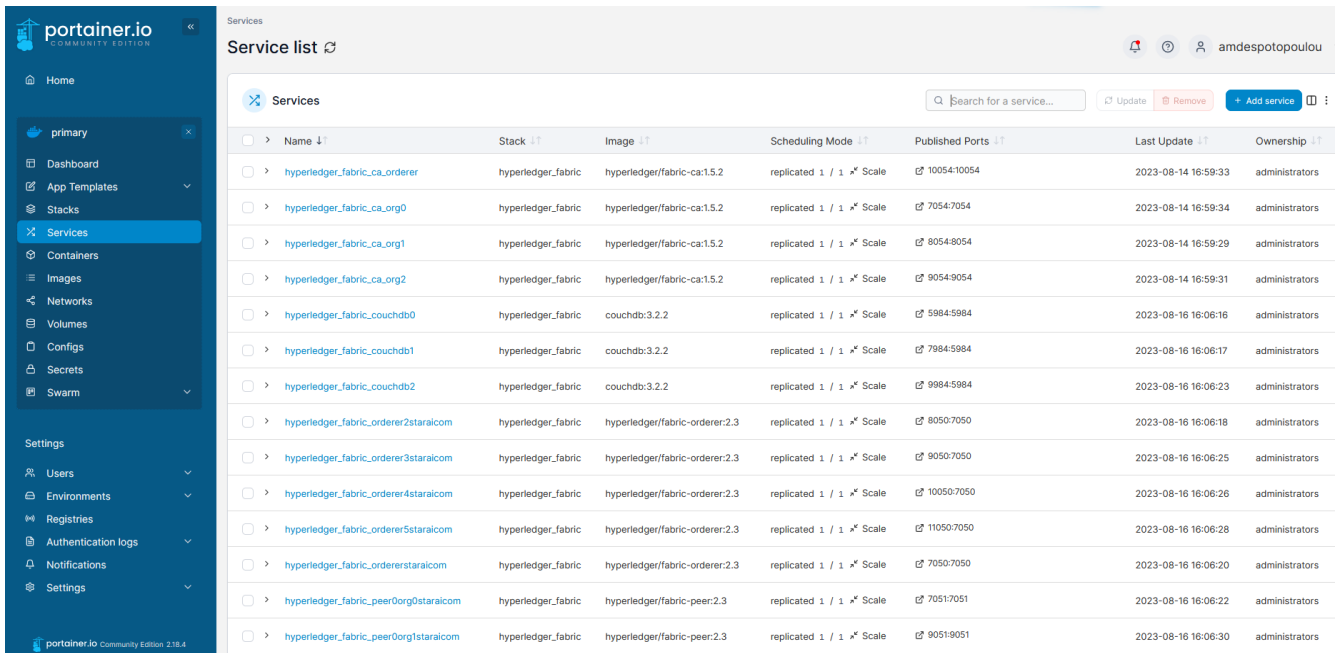


Figure 48: The deployed services of our stack as monitored via Portainer



Figure 49: The distribution of running services among nodes as visualised in Portainer

### 5.6.5 Monitoring current Global State using Fauxton

**Fauxton**<sup>40</sup> is a web-based graphical user interface (GUI) included with Apache CouchDB, the state database used in Hyperledger Fabric. It provides a user-friendly way to interact with and visualize the data stored in CouchDB. In the context of Hyperledger Fabric, Fauxton can be utilized to visualize the global state of the blockchain on a node by connecting to the CouchDB instance associated with that node. This allows users to explore the current values of assets, identities, and other data stored in the state database, along with historical changes. Fauxton's querying and filtering capabilities enable users to search for specific data elements, perform custom queries, and retrieve insights into the state of the network. This visualization tool proves useful for monitoring and analyzing the current status of assets and transactions in a Hyperledger Fabric network, aiding developers, administrators, and other stakeholders in making informed decisions and gaining a deeper understanding of the network's overall state.

The three dashboards (one per Organization) can be accessed in all nodes by evoking the responding ports.

```

http://<machine_ip>:5984/_utils/           [organization zero]
http://<machine_ip>:7984/_utils/         [organization one]
http://<machine_ip>:9984/_utils/         [organization two]

```

The administrator credentials (Figure 50) can be accessed (and changed before deployment via the .yaml configuration file).

<sup>40</sup> Official Fauxton visual guide is available here: <https://couchdb.apache.org/fauxton-visual-guide/index.html> (accessed October 2023).

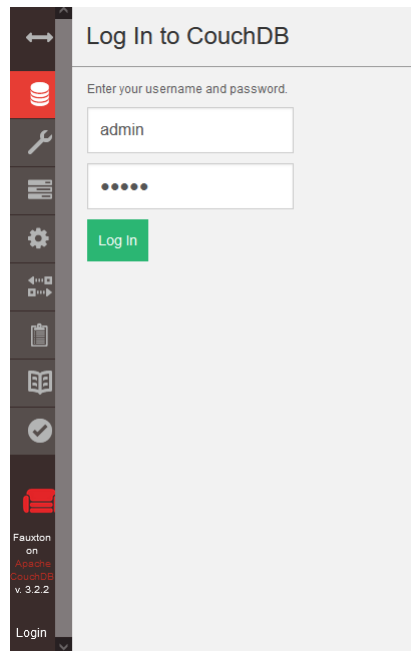


Figure 50: Request for administrator credentials upon accessing the Fauxton dashboard

Fauxton can be then used to directly "read" the databases in a human-friendly manner (Figure 51). This function is reserved to **administrators only**. End-users ought to use the Dapps built on top of the Fabric blockchain.

Name	Size	# of Docs	Partitioned	Actions
<i>_replicator</i>	2.3 KB	1	No	[Refresh] [Lock] [Delete]
<i>_users</i>	2.3 KB	1	No	[Refresh] [Lock] [Delete]
<i>fabric__internal</i>	291 bytes	1	No	[Refresh] [Lock] [Delete]

Showing 1-3 of 3 databases. Databases per page 20

Figure 51: The databases created by default as seen through Fauxton

## 5.7 Establishing the Communication between STAR Platform Services (Nodes)

### 5.7.1 Instantiation of Channels between Hyperledger Fabric Nodes

In Hyperledger Fabric, **CLI docker containers** serve as a critical interface for interacting with the network through the Command Line Interface (CLI). These containers encapsulate the necessary tools and utilities for managing and administering various aspects of the blockchain network, such as creating and joining channels, installing chaincode, endorsing transactions, and querying the ledger. CLI Docker containers provide a consistent environment for users to execute commands and carry out administrative tasks across different nodes and systems in the network<sup>41</sup>.

Channels in Hyperledger Fabric are private sub-networks that allow for the segregation and isolation of transactions between specific participants. While the genesis block establishes the foundational configuration of the channel, instantiation is the process of creating the necessary infrastructure to begin transactions within that channel. This includes defining the ordering service, setting up endorsement policies, and configuring the peers that will participate in the channel. **Instantiating a channel** ensures that the necessary components are in place for secure and isolated transactions among the chosen parties.

Peers are the nodes that maintain the distributed ledger and endorse transactions in Hyperledger Fabric. When peers are joined to a channel, they become part of that channel's network and can participate in validating and endorsing transactions that pertain to that channel. **Joining a peer to a channel** allows it to receive blocks of transactions and updates related to that channel, ensuring synchronization and consistency among peers. Anchor peers play a crucial role in maintaining network stability by serving as a reference point for other peers in the same organization to establish connections. **Updating anchor peers'** information in the channel configuration ensures that peers across different organizations can effectively communicate and maintain a reliable network topology, enhancing the overall resilience and performance of the Hyperledger Fabric network.

In the present section we will be instantiating the Channels for whom we have already produced artifacts. Towards this end we need to create one more container per organization, a CLI tool to perform our operations from. Then we will be joining all three of our peers to said channel and then updating all of them on the fact.

### 5.7.2 Working with the CLI Containers representing Nodes

Our first step is to once more to prepare a .yaml configuration file.

Using the command line of the "manager" node, we navigate as usual to the

*/fabric-deployment/network-setup/*

directory and execute the following command after updating with the domain name:

```
sed -e 's/${PROJECT_DOMAIN}/star-ai.eu/g' -e 's/${SHORT_DOMAIN}/staraieu/g'
docker/docker-compose-cli-matrix.yaml > docker/docker-compose-cli.yaml
```

<sup>41</sup> Documentation of Hyperledger Fabric CLI commands available here: <https://fabric-docs-test.readthedocs.io/en/latest/API/CLI/> (accessed October 2023).

Then, using the command line of the “manager” node, we execute the following command:

```
docker stack deploy --with-registry-auth -c docker/docker-compose-cli.yaml hyperledger_fabric
```

The visible result is depicted below (Figure 52):

```

• amdespotopoulou@blockchain-org0:~/hyperledger-fabric-installation-kit/fabric-deployment/net
work-setup$ docker stack deploy --with-registry-auth -c docker/docker-compose-cli.yaml hype
rledger_fabric
Ignoring deprecated options:

container_name: Setting the container name is not supported.

Creating service hyperledger_fabric_cliOrg0
Creating service hyperledger_fabric_cliOrg1
Creating service hyperledger_fabric_cliOrg2
    
```

Figure 52: The result of stack deployment through the console of the “manager”

To verify and troubleshoot the deployment of the three Swarm services, one may use the console commands and dashboards examined in previous sections. One thing to note is that the CLI containers are **not exposed** to the greater environment via any port.

The desired result is to achieve the following distribution of containers (Figure 53):

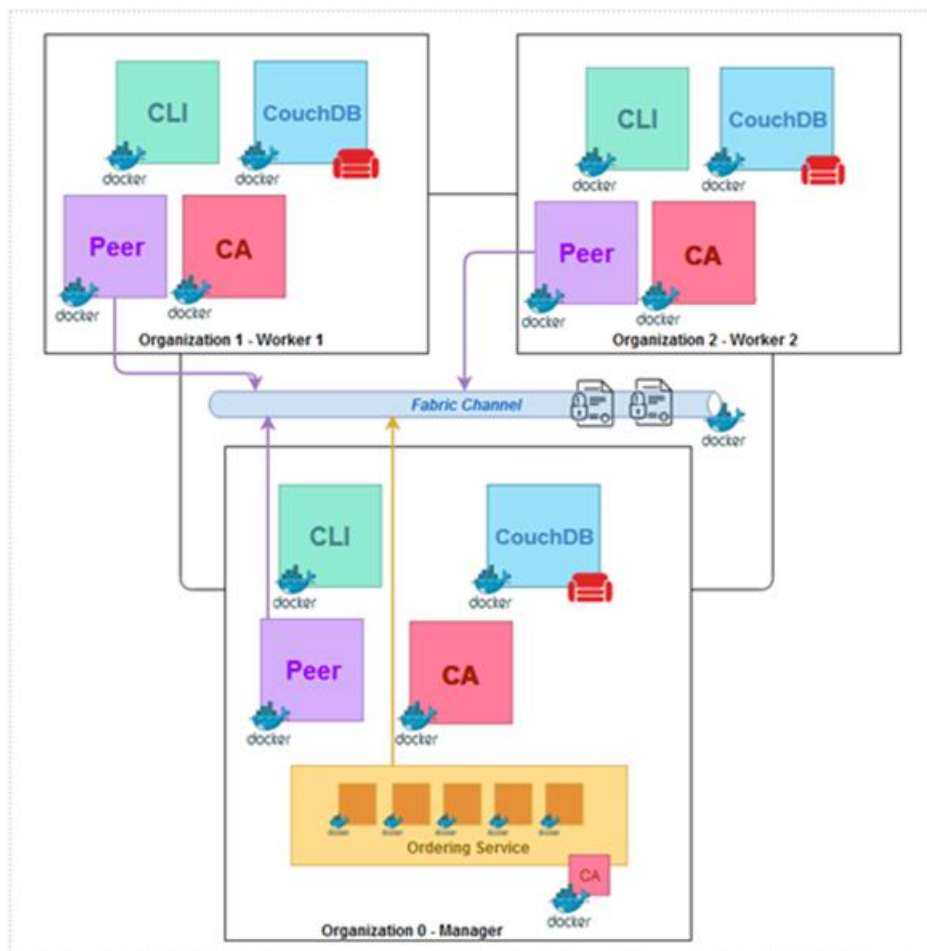


Figure 53: The distribution of Docker containers after the deployment of CLIs (depicted in light green)

To use the tools incorporated into the CLI, one has to access the container's own console. This is feasible in two ways (let us start with the one of Org0 in the "manager" node as an example).

The first method is using the command line of the virtual machine. First, we obtain the identifier of the CLI container:

```
docker ps | grep cli
```

Then we insert the container's own command line by typing:

```
docker exec -it <container_ID> /bin/bash
```

The second method is to use Portainer. From the "Containers" menu we locate the container entitled

```
hyperledger_fabric_cliOrg0...
```

and then, at the "quick actions" column we click the "exec console" one (Figure 54) and the "Connect" blue button (Figure 55).

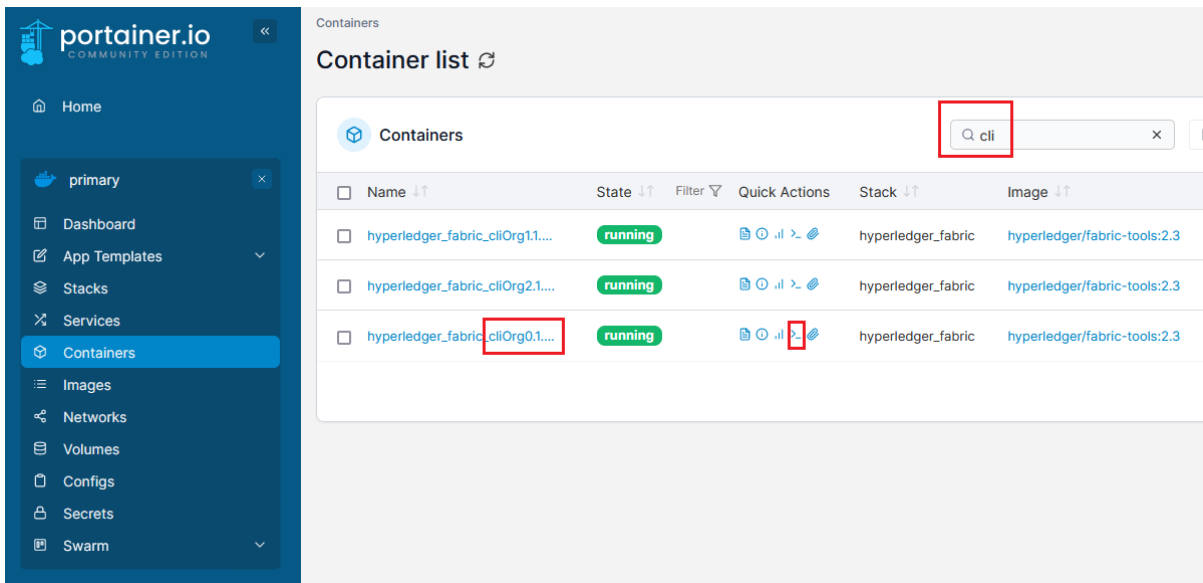


Figure 54: Accessing the console of the CLI corresponding to Organization Zero via Portainer

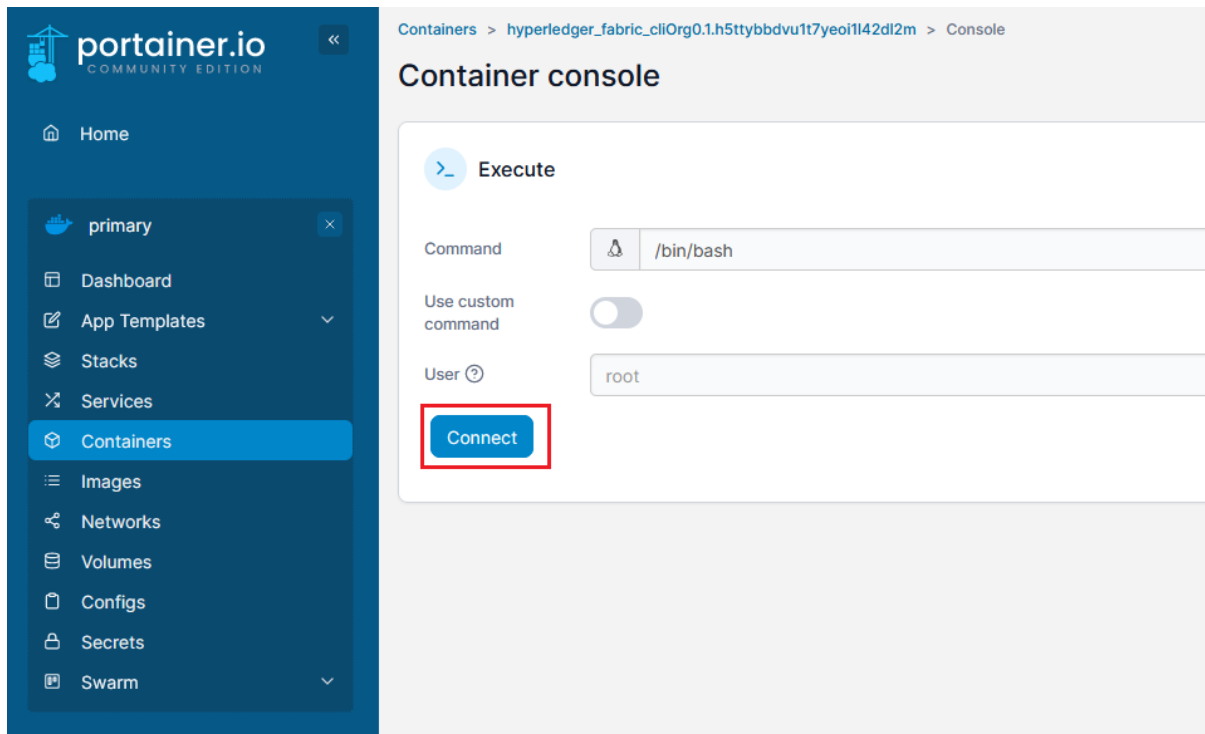


Figure 55: The button to press so as to enable the console

### 5.7.3 Instantiation of Channels for STAR PoC Architecture

Once inside the CLI container of Org0, we will be instantiating the three new channels. The channels will be communicating with one of the orderers, the one accessed via port 7050 (this is a random decision for our PoC, since all Orderers are hosted on different ports of the same virtual machine). The material to be taken into account are the files we created earlier and placed in subfolders into the following directory (Figure 42):

*fabric-deployment/network-setup/channel-artifacts*

Then we proceed to execute the following six commands in order:

```
export CHANNEL_NAME=datasourceschannel
```

```
peer channel create -o orderer.${PROJECT_DOMAIN}:7050 -c ${CHANNEL_NAME} -f
./channel-artifacts/channel-artifacts-04-da/${CHANNEL_NAME}.tx --
outputBlock ./channel-artifacts/channel-artifacts-04-
da/${CHANNEL_NAME}.block --tls --cafile $ORDERER_CA
```

```
export CHANNEL_NAME=processorschannel
```

```
peer channel create -o orderer.${PROJECT_DOMAIN}:7050 -c ${CHANNEL_NAME} -f
./channel-artifacts/channel-artifacts-05-pr/${CHANNEL_NAME}.tx --
outputBlock ./channel-artifacts/channel-artifacts-05-
pr/${CHANNEL_NAME}.block --tls --cafile $ORDERER_CA
```

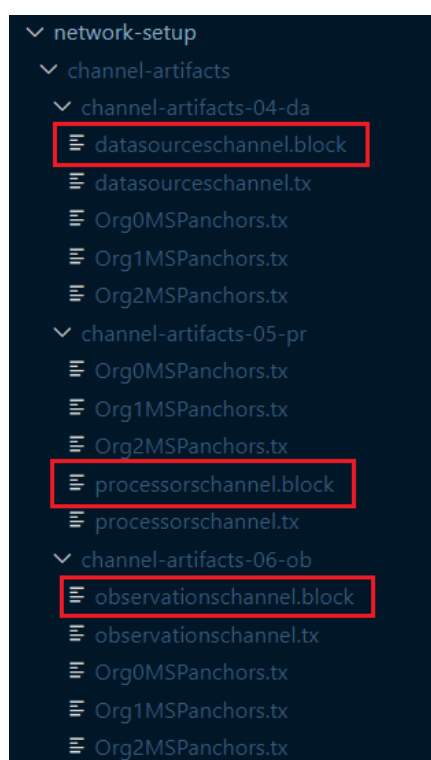
```
export CHANNEL_NAME=observationschannel
```

```
peer channel create -o orderer.${PROJECT_DOMAIN}:7050 -c ${CHANNEL_NAME} -f
./channel-artifacts/channel-artifacts-06-ob/${CHANNEL_NAME}.tx --
outputBlock ./channel-artifacts/channel-artifacts-06-
ob/${CHANNEL_NAME}.block --tls --cafile $ORDERER_CA
```

The execution must conclude without errors, with the indication:

```
INFO 0b0 Received block: 0
```

In addition, new .block files must be visible within the folders hosting the channel artifacts (Figure 56):



*Figure 56: The .block files created after channels instantiation*

The .block file created per channel is called the Genesis Block or the **Channel Genesis Block**. The Genesis Block serves as the initial block of transactions that defines the foundational configuration and parameters of the channel. This block is then used when other peers join the channel, as it contains the initial configuration that all peers need to synchronize and operate within the newly created channel.

This .block artifact **must then be copied** to the other two machines.

## 5.7.4 Join of Peer Nodes to Channel and Update of Anchor Peers

The next step is to join the peers to the Channel. After the copy of the .block file to the other two VMs, the following command must be executed **inside the console of each** of the three CLI containers.

As a reminder to enter the CLI container console one must type:

```
docker ps | grep cli  
docker exec -it <container_ID> /bin/bash
```

Once inside, to join the Peer to the Channel, one must type:

```
peer channel join -b ./channel-artifacts/channel-artifacts-04-  
da/datasourceschannel.block  
  
peer channel join -b ./channel-artifacts/channel-artifacts-05-  
pr/processorschannel.block  
  
peer channel join -b ./channel-artifacts/channel-artifacts-06-  
ob/observationschannel.block
```

Success is indicated by the final resulting line:

```
Successfully submitted proposal to join channel
```

To further verify success, one may also type (always inside the container):

```
peer channel list
```

The result concludes with the following lines:

```
Channels peers has joined:  
  
datasourceschannel  
  
processorschannel  
  
observationschannel
```

For our peers to be able to communicate among them using the Service Discovery, we have as a final step to update the Anchor Peers. In each machine, while inside the console of the CLI container, one must execute respectively:

In CLI hosted at "manager":

```
export PROJECT_DOMAIN="digiprime.eu"  
  
export CORE_PEER_LOCALMSPID=Org0MSP
```

```
peer channel update -o orderer.${PROJECT_DOMAIN}:7050 \
  --ordererTLSHostnameOverride orderer.${PROJECT_DOMAIN} \
  -c datasourceschannel -f ./channel-artifacts/channel-artifacts-04-
da/${CORE_PEER_LOCALMSPID}anchors.tx --tls \
  --cafile $ORDERER_CA

peer channel update -o orderer.${PROJECT_DOMAIN}:7050 \
  --ordererTLSHostnameOverride orderer.${PROJECT_DOMAIN} \
  -c processorschannel -f ./channel-artifacts/channel-artifacts-05-
pr/${CORE_PEER_LOCALMSPID}anchors.tx --tls \
  --cafile $ORDERER_CA

peer channel update -o orderer.${PROJECT_DOMAIN}:7050 \
  --ordererTLSHostnameOverride orderer.${PROJECT_DOMAIN} \
  -c observationschannel -f ./channel-artifacts/channel-artifacts-06-
ob/${CORE_PEER_LOCALMSPID}anchors.tx --tls \
  --cafile $ORDERER_CA
```

In CLI hosted at “worker1” and “worker2” one must type the same five commands with the only exception that the `CORE_PEER_LOCALMSPID` parameter must be set to `Org1MSP` and `Org2MSP` respectively instead.

The action can be considered successful if the last of the resulting messages reads:

```
Successfully submitted channel update
```

## 5.8 Running Business Logic through Smart Contracts

### 5.8.1 Smart Contracts in the Context of Hyperledger Fabric

Earlier in the present document, and most precisely in Section 2.1.4, we have discussed in length the concept of smart contracts both in the pre-blockchain academic literature, as well as the interpretation most researchers agree on today. Szabo's vision was inspired by the idea of applying computer code to enforce, verify, or facilitate agreements without the need for human intervention, offering a decentralized, secure, and tamper-resistant method for executing transactions and ensuring the integrity of agreements. In the contemporary context, blockchain smart contracts are automated, trustless agreements that run on blockchain technology, executing actions based on predefined conditions without the need for intermediaries, providing a secure and transparent method for various applications in industries like finance, real estate, and more.

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].

In the context of Industry 4.0 use cases empowered by AI, Hyperledger Fabric's chaincode may govern, among other applications, interactions between IoT devices and AI algorithms, ensuring data sources' authenticity, verifying the integrity of data processor configurations, and validating the accuracy of data processor results. Those are the business problems that have been the objects of interest for STAR and have been examined from multiple facets in the present document.

Chaincode in Hyperledger Fabric is executed within Docker containers, which provide a secure and isolated environment for smart contract execution. When chaincode is deployed, it is packaged inside a Docker container image along with all its dependencies and runtime environment. This containerization ensures that the chaincode runs consistently across different nodes and eliminates any issues related to the host's configuration. Additionally, it enhances security by isolating the chaincode's execution from the underlying infrastructure, which is vital in a permissioned blockchain network like Fabric. Naturally, containers corresponding to chaincode are monitorable using all the usual user interfaces and observability tools (including logs) available to operations-oriented software engineers (consult Chapter 6 for suggestions).

### 5.8.2 Producing Chaincode for the Three Services of The PoC

As far as STAR is concerned, smart contracts and chaincode will support the three services described in detail in sections 3.4.2, 3.4.3 and 3.4.4: (i) Data Sources Verifiability (ii) Data Processor Configurations Verifiability and (iii) Data Processor Results Verifiability.

The most popular programming languages for writing chaincode in Hyperledger Fabric are Go, JavaScript and Java. Go<sup>42</sup>, one of the most popular chaincode programming languages in Fabric, is known for its efficiency and suitability for complex, high-performance applications. JavaScript<sup>43</sup> is favored for its ease of development and widespread adoption among web developers, offering seamless integration with Fabric for smart contract coding. Java<sup>44</sup>, another widely used option, brings its portability and versatility, allowing developers to leverage existing Java knowledge to create chaincode in Fabric. For STAR the elected language has been Java.

Three different Java projects have been created, one per use case. To automate code lifecycle, including testing, the Apache Maven<sup>45</sup> automation tool has been employed. Apache Maven is open-source and widely used for project management and build automation in Java projects, providing a structured way to manage dependencies, build processes, and project lifecycles.

<sup>42</sup> Official website for Go programming language: <https://go.dev> (accessed October 2023).

<sup>43</sup> Official website for the ECMAScript® 2023 language specification: <https://ecma-international.org/publications-and-standards/standards/ecma-262/> (accessed October 2023).

<sup>44</sup> Official website for Java: <https://www.java.com/en/> (accessed October 2023).

<sup>45</sup> Official website for Apache Maven: <https://maven.apache.org/> (accessed October 2023).

For automated chaincode testing the library that has been leveraged and incorporated into Maven’s lifecycle steps is Mockito<sup>46</sup>.

The chaincode has been designed so as to accommodate the functions and specifications described in Chapter 3. A code snippet of the smart contract produced in the context of the “Data Sources Traceability” use case is visible right below (Figure 57):

```

1  package eu.starai.blockchainservices.chaincode.model;
2
3  import org.hyperledger.fabric.contract.Context;
4  import org.hyperledger.fabric.contract.ContractInterface;
5  import org.hyperledger.fabric.contract.annotation.*;
6  import org.hyperledger.fabric.shim.ledger.KeyValue;
7  import org.hyperledger.fabric.shim.ledger.QueryResultsIterator;
8
9  import java.time.LocalDateTime;
10 import java.util.*;
11
12 import static java.nio.charset.StandardCharsets.UTF_8;
13
14 no usages
15 @Contract(name = "DataSourcesContract",
16           info = @Info(title = "DataSourcesContract contract",
17                       description = "Chaincode for Data Sources Business Operations",
18                       version = "1.0",
19                       license =
20                           @License(name = "SPDX-License-Identifier: Apache-2.0",
21                                   url = ""),
22                       contact = @Contact(email = "angelamaria.despotopoulou@netcompany.com",
23                                           name = "DataSourcesContract",
24                                           url = "https://www.star-ai.eu/"))))
25 @Default
26 public class DataSourcesContract implements ContractInterface {
27
28     no usages
29     public DataSourcesContract() {
30
31     }
32
33     no usages
34     @Transaction()
35     public boolean datasourceExists(Context ctx, String sourceID) {
36         byte[] buffer = ctx.getStub().getState(sourceID);
37         return (buffer != null && buffer.length > 0);
38     }
39
40 }

```

Figure 57: Code snippet of chaincode (smart contract) corresponding to “Data Sources Traceability” Use Case

<sup>46</sup> Official website for the Mockito framework: <https://site.mockito.org> (accessed October 2023).

## 6 Continuous Monitoring of the Cluster hosting the DLSDR Framework

### 6.1 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.

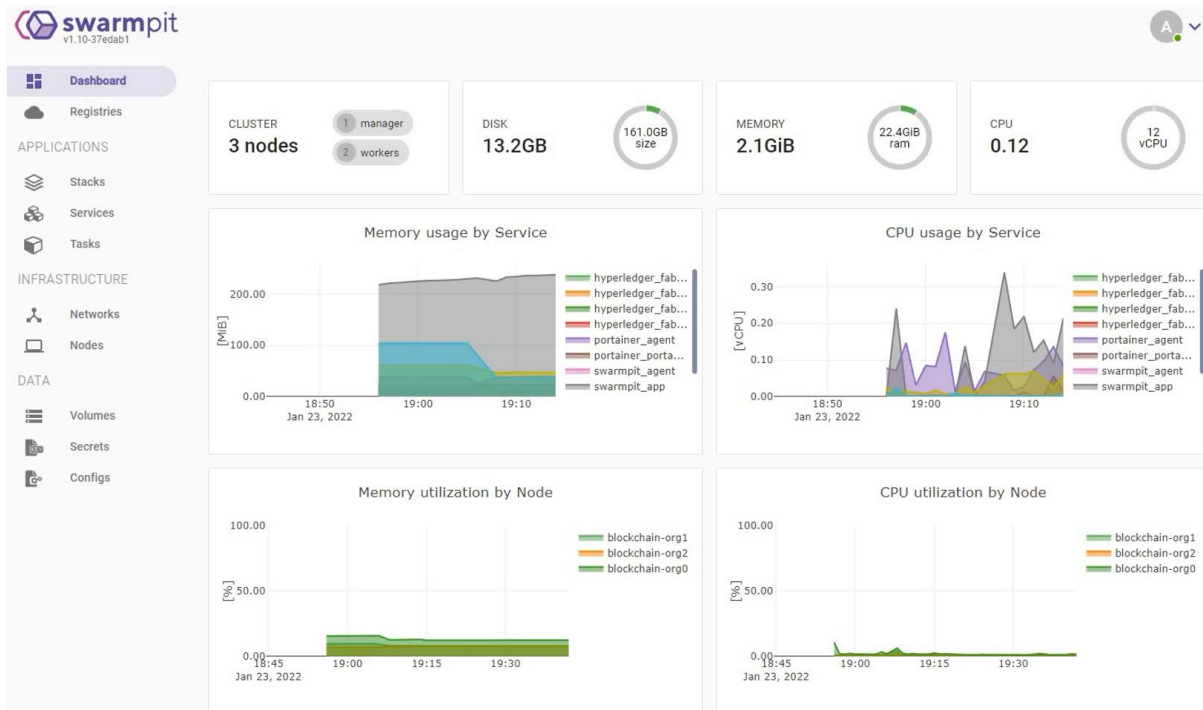


Figure 58: Swarmpit Master Dashboard for Resources Monitoring

#### 6.1.1 Swarmpit

Swarmpit<sup>47</sup> 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.

<sup>47</sup> Swarmpit official webpage: <https://swarmpit.io/> (accessed April 2022)

- **Service auto-redeploy:** checks whether new service image has been published and updates service accordingly.

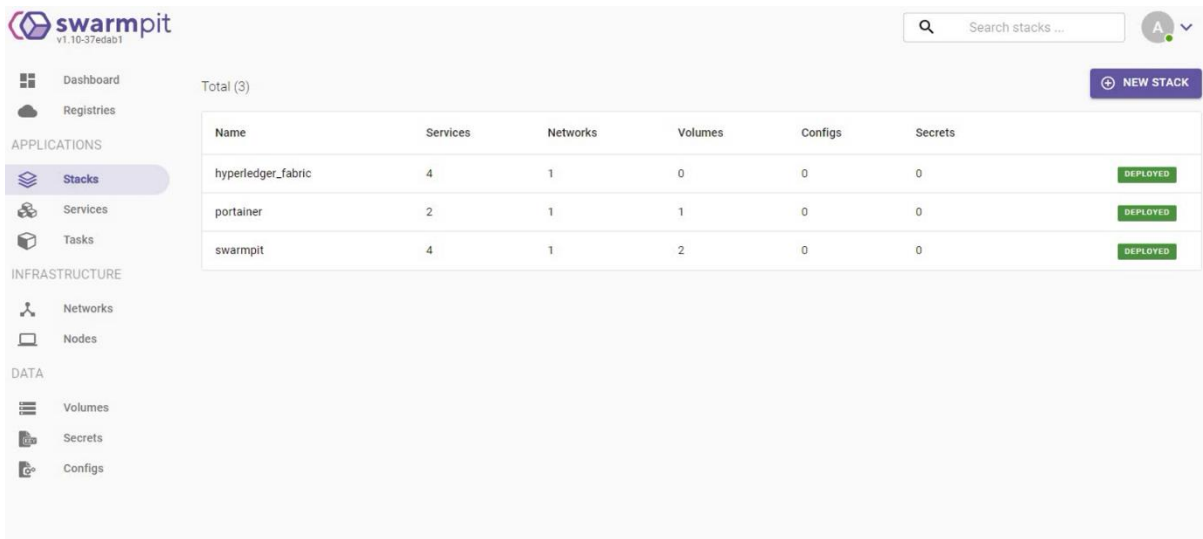


Figure 59: Swarmpit Depicting Deployed Docker Swarm Stacks

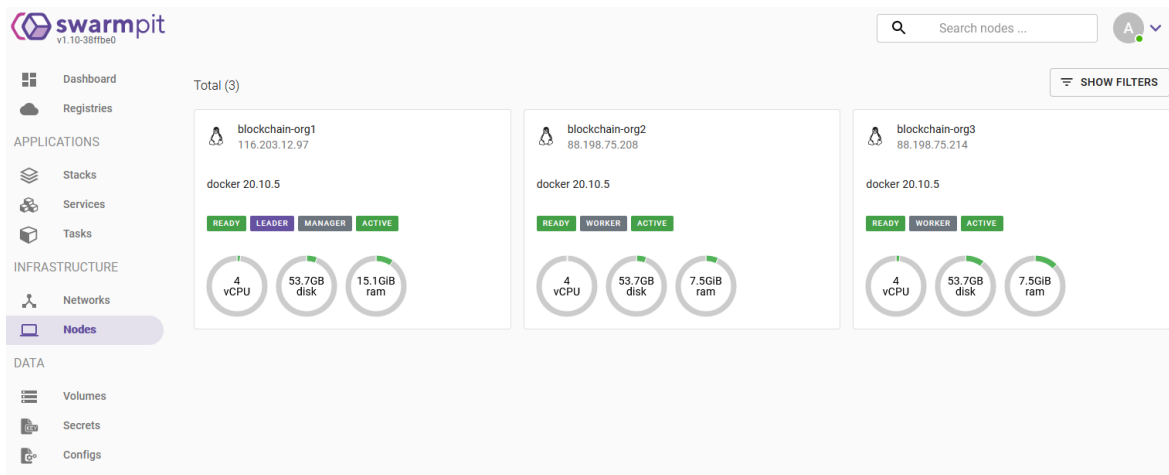


Figure 60: Swarmpit Depicting the Virtual Machines Formulating a Network

The dashboard of Docker Swarm is illustrated in Figure 58, Figure 59 and Figure 60.

### 6.1.2 Portainer

Portainer<sup>48</sup> 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 61, Figure 62 and Figure 63.

<sup>48</sup> Portainer official website: <https://www.portainer.io/> (accessed April 2022)

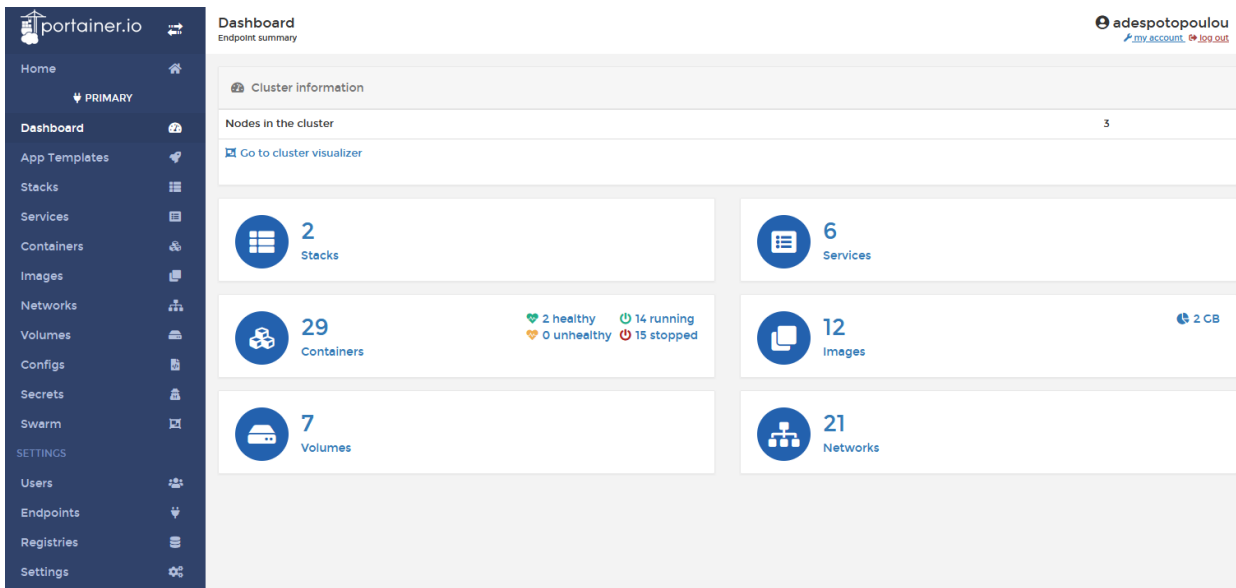


Figure 61: Portainer Landing Page for Containers Administration

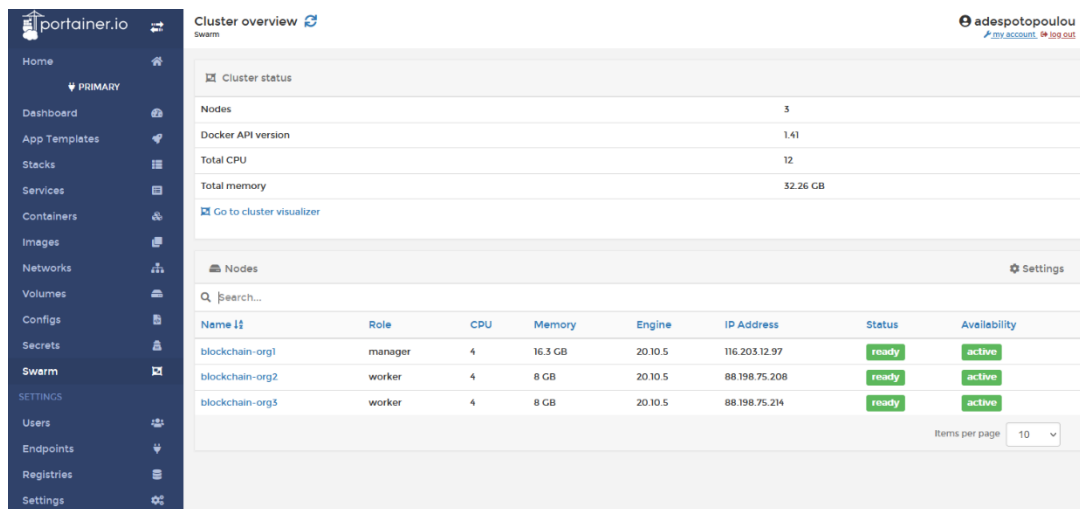


Figure 62: Portainer Swarm Cluster Overview

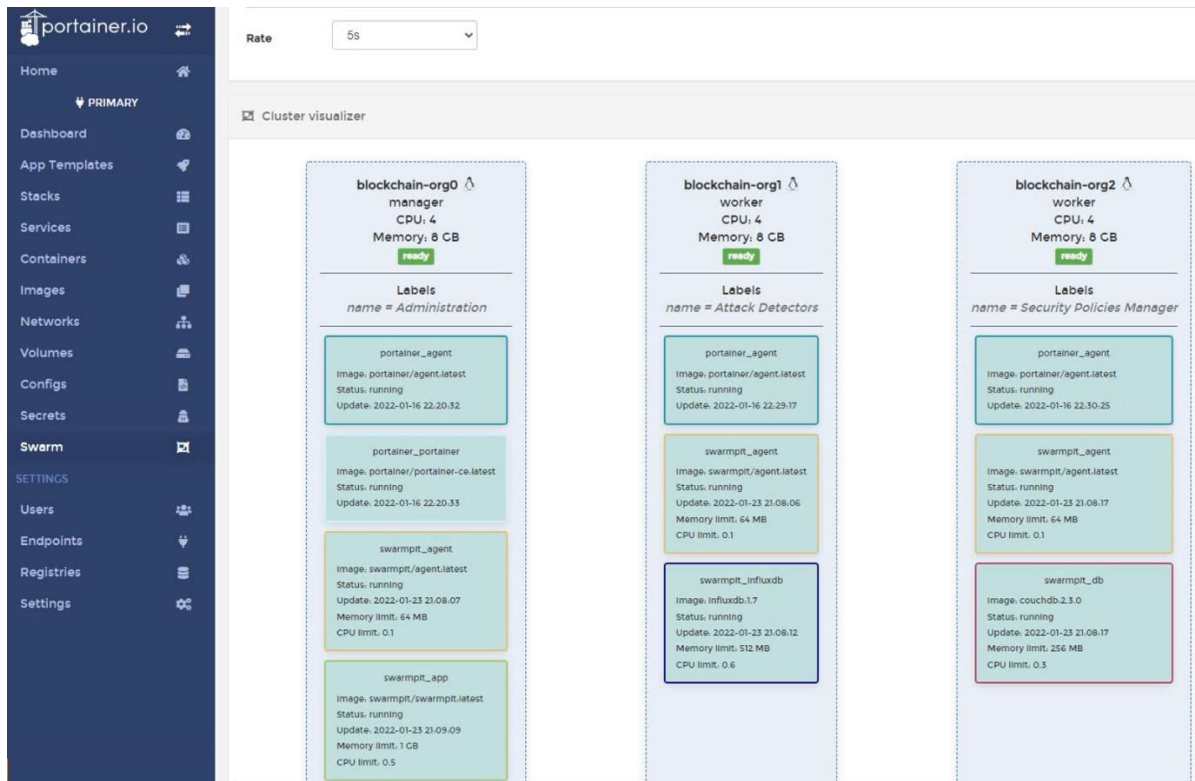


Figure 63: Portainer Swarm Visualizer with Nodes Assigned to STAR Services

## 6.2 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.

### 6.2.1.1 Hyperledger Explorer

Hyperledger Explorer<sup>49</sup> 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 64 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.

<sup>49</sup> Hyperledger Explorer official website: <https://www.hyperledger.org/use/explorer> (accessed April 2022)

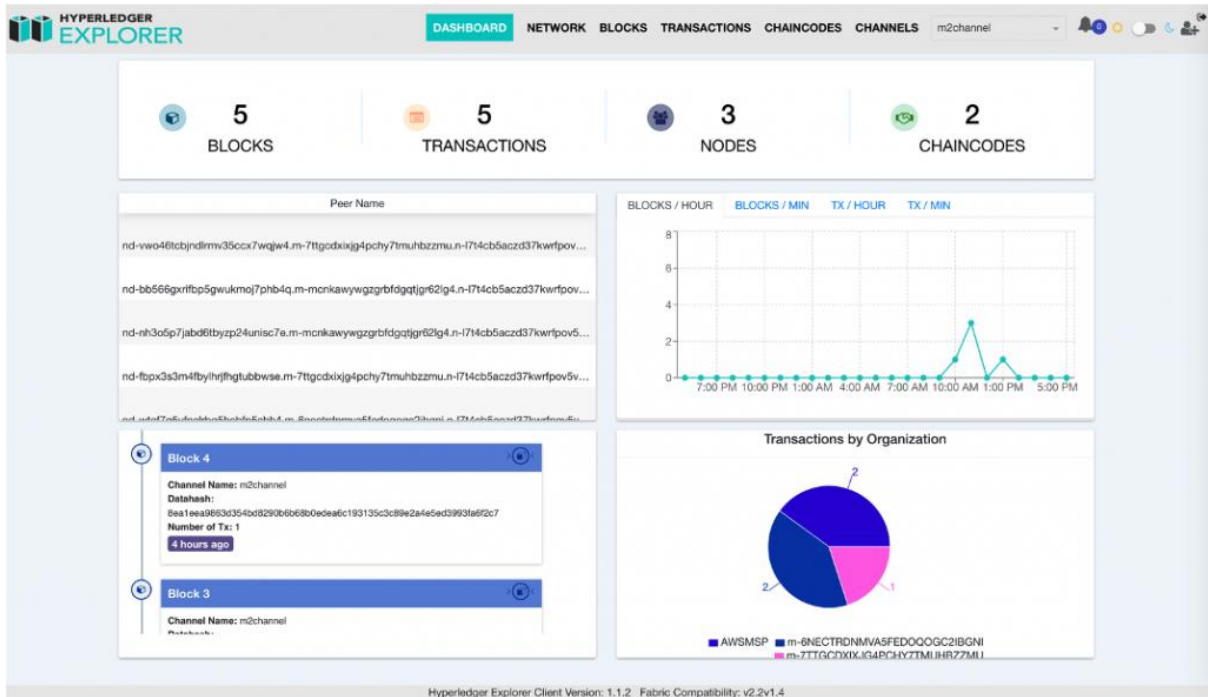


Figure 64: Hyperledger Explorer Dashboard

## 7 Conclusion and Future Research Directions

In this concluding chapter, we reflect on the journey that began with the inception of D3.1 and culminated in this advanced deliverable. Our work has revolved around harnessing the potential of distributed ledger technologies to enhance the traceability of industrial data within AI systems. Building upon the foundational principles and strategies outlined in D3.1, we have made significant strides in this endeavour, and as we look to the future, there are several key points to highlight:

### 7.1 Achievements and Key Findings

In this document, we delved deeper into the technical intricacies of our approach, providing a more detailed roadmap for implementation and deployment. We emphasized the synergy between blockchain and AI, highlighting the robust security and transparency that this combination brings to the table. Our dual-pronged approach to ensuring the integrity of configuration files in AI-powered components has shown promising results, and the diagrams illustrating these approaches provide a clear understanding of our methods.

We placed strong emphasis on the practical aspects of deploying a cluster of blockchain nodes. The steps involved in bringing our vision to life were elucidated, and specific milestones were outlined. The integration of our advancements into the STAR cybersecurity toolset in the context of WP3 and then into the pilot sites in the context of WP6 is pivotal for enhancing the resilience of the industrial systems under our focus against cyber threats. Strategies for seamless integration, scalability, and leveraging feedback loops through monitoring tools have been discussed.

This deliverable serves as a valuable resource for stakeholders, offering a comprehensive blueprint for the successful implementation of future distributed ledger-based solutions to bolster industrial data traceability within the dynamic landscape of AI applications for Industry 4.0 and beyond.

### 7.2 Future Research Directions

As we conclude this phase of our work, we propose several avenues for future research and development:

#### Advanced Dissimilarity Measurement

Building on our work in measuring dissimilarity between configuration files, we encourage further exploration into advanced techniques for dissimilarity measurement, especially in semi-structured, hierarchical formats like JSON. This research could lead to more efficient methods for detecting unauthorized changes in configuration files.

Furthermore, the reconstruction of convolutional neural networks as graphs has shown promise in improving the accuracy of AI industrial data representation and analysis. Future research should focus on refining this approach, exploring its applicability to different types of neural networks, and optimizing the process for broader use cases.

#### Blockchain Integration in Diverse Industrial Systems

The current deliverable primarily focuses on the use of blockchain for enhancing industrial data traceability in AI systems. Future research should expand the scope to encompass a wider range of industrial systems, including (meta)data generated by sensors, a bigger variety of industrial automation devices, robots, cyber-physical systems and business information

systems. Investigating the adaptability and effectiveness of our approach in these diverse contexts is essential.

### Performance Optimization

Optimizing the performance and scalability of the infrastructure developed within Task 3.1 is crucial for industrial deployments. Future work should aim to tailor the solution to meet the unique demands of various industrial contexts, ensuring maximum efficiency and scalability as we move from PoC to a higher-TRL output.

In conclusion, our journey to harness distributed ledger technologies for enhancing industrial data traceability within AI systems has yielded significant progress. The foundations laid in this deliverable serve as a solid starting point for further exploration and development. By addressing the outlined research directions, we can continue to push the boundaries of what is achievable in the dynamic landscape of AI and industrial data traceability.

## 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: <a href="https://pharmaphorum.com/views-and-analysis/">https://pharmaphorum.com/views-and-analysis/</a>
[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. <a href="https://jsoncompare.com/">https://jsoncompare.com/</a> .
[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</i> (2018)
[ExplorerDocs]	Official Hyperledger Explorer Documentation, accessed October 2023. Documents available online: <a href="https://blockchain-explorer.readthedocs.io/en/main/index.html">https://blockchain-explorer.readthedocs.io/en/main/index.html</a>
[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: <a href="http://creativecommons.org/licenses/by/4.0/">http://creativecommons.org/licenses/by/4.0/</a>
[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 &amp; 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. <a href="http://www.jsondiff.com">http://www.jsondiff.com</a> .
[Grover16]	A. Grover, J. Leskovec, node2vec: scalable feature learning for networks, <i>Proceedings of KDD, ACM</i> (2016), pp. 855-864
[Grover19]	A. Grover, A. Zweig, S. Ermon, Graphite: iterative generative modeling of graphs, <i>Proceedings of ICML</i> (2019), 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. Technol.</i> , 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.
[IBM21]	IBM Cloud Education, "Containerization," 23 June 2021. [Online]. Available: <a href="https://www.ibm.com/cloud/learn/containerization">https://www.ibm.com/cloud/learn/containerization</a> . [Accessed October 2023].
[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: <a href="https://www.kaleido.io/blockchain-blog/enterprise-blockchain-protocols-a-technical-analysis-of-ethereum-vs-fabric-vs-corda">https://www.kaleido.io/blockchain-blog/enterprise-blockchain-protocols-a-technical-analysis-of-ethereum-vs-fabric-vs-corda</a> [Accessed October 2023].
[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, Learning Deep Generative Models of Graphs (2018), 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.
[Liu18]	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," <i>2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)</i> , 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 Proc. Resilience Week (RWS), 2017, pp. 18–23.
[Nakamoto08]	S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: <a href="https://bitcoin.org/bitcoin.pdf">https://bitcoin.org/bitcoin.pdf</a> . [Accessed October 2023].
[Neuhaus09]	M. Neuhaus, K. Riesen, and H. Bunke, "Novel kernels for errortolerant graph classification", Spatial Vis., vol. 22, no. 5, pp. 425–441, 2009.
[Noizat15]	P. Noizat et al., "Blockchain electronic vote" in Handbook of Digital Currency, San Diego, CA, USA:Academic, pp. 453-461, 2015.
[Pawlik15]	Pawlik, M., Augsten, N.: Efficient computation of the tree edit distance. ACM Trans. Database Syst. (TODS) 40(1) (2015). Article No. 3
[Pawlik16]	Mateusz Pawlik and Nikolaus Augsten. 2016. Tree edit distance: Robust and memory-efficient. Information Systems 56 (2016), 157–173
[Perozzi14]	B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: online learning of social representations, Proceedings of KDD, ACM (2014), pp. 701-710
[PostgreSQL22]	PostgreSQL Documentation. 2023. Additional Supplied Modules. <a href="https://www.postgresql.org/docs/current/contrib.html">https://www.postgresql.org/docs/current/contrib.html</a> [Accessed October 2023].
[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", Int. J. Inf. Manage., vol. 46, pp. 70-82, Jun. 2019.
[Radanović18]	I. Radanović and R. Likić, "Opportunities for use of blockchain technology in medicine", Appl. Health Econ. Health Policy, vol. 16, no. 5, pp. 583-590, Oct. 2018.
[Ramachandran18]	Aravind Ramachandran and Murat Kantarcioglu. 2018. SmartProvenance: A Distributed, Blockchain Based DataProvenance System. Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY '18. Association for Computing Machinery, New York, NY, USA, 35–42. DOI: <a href="https://doi.org/10.1145/3176258.3176333">https://doi.org/10.1145/3176258.3176333</a>
[Riesen14]	K. Riesen, A. Fischer, and H. Bunke, "Combining bipartite graph matching and beam search for graph edit distance approximation", in ANNPR 2014, 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", Proc. Int. Smart Cities Conf. (ISC), 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 Proc. IEEE Int. Conf. Internet Things (iThings), IEEE Green Comput. Commun. (GreenCom), IEEE Cyber, Phys. Social Computing, 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) Similarity Search and Applications. SISAP 2017. Lecture Notes

	in Computer Science, vol 10609. Springer, Cham. <a href="https://doi.org/10.1007/978-3-319-68474-1_11">https://doi.org/10.1007/978-3-319-68474-1_11</a>
[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, <i>Proceedings of ICML (2018)</i> , 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, <i>Proceedings of ICLR (2020)</i>
[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", <i>Procedia Manufacturing</i> , Volume 54, 2021, Pages 64-69, ISSN 2351-9789, <a href="https://doi.org/10.1016/j.promfg.2021.07.011">https://doi.org/10.1016/j.promfg.2021.07.011</a> .
[Sovrin20]	Sovrin Foundation, "Data Privacy Regulation and Distributed Ledger Technology", <i>Whitepaper Collection</i> , 2020.
[SSL21]	SSL Support Team, "What is SSL?," 28 September 2021. [Online]. Available: <a href="https://www.ssl.com/faqs/faq-what-is-ssl/">https://www.ssl.com/faqs/faq-what-is-ssl/</a> . [Accessed October 2023].
[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 <i>GbrPR 2017</i> , 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 October 2022. Documents available online: <a href="https://docs.docker.com/engine/swarm/">https://docs.docker.com/engine/swarm/</a>
[Szabo17]	N. Szabo, "Winning Strategies for Smart Contracts," <i>Blockchain Research Institute Whitepapers</i> , 2017.
[Tai79]	Tai, H.-C.: The tree-to-tree correction problem. <i>J. ACM (JACM)</i> 26(3), 422–433 (1979)
[Tian16]	F. Tian, "An agri-food supply chain traceability system for China based on RFID & blockchain technology," in <i>Proc. 13th Int. Conf. Service Syst. Service Manage. (ICSSSM)</i> , 2016, pp. 1–6
[Tian17]	F. Tian, "A supply chain traceability system for food safety based on HACCP, blockchain & Internet of Things," in <i>Proc. Int. Conf. Service Syst. Service Manage. (ICSSSM)</i> , 2017, pp. 1–6.
[Velickovic18]	P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, <i>Proceedings of ICLR (2018)</i>
[Vukolić15]	M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication" in <i>International Workshop on Open Problems in Network Security</i> , 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", <i>Appl. Sci.</i> , 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. SIAM J. Comput. 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", Energy Procedia, 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 11: 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:schema>
```

```
</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>
```