

**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:** Netcompany-Intrasoft



Funded by the Horizon 2020 Framework Programme of the European Union

## DELIVERABLE

### D6.4 – Integrated STAR Platform-Final version

<b>Dissemination level</b>	PU -Public
<b>Type of Document</b>	Report
<b>Contractual date of delivery</b>	30/06/2023
<b>Deliverable Leader</b>	DFKI
<b>Status - version, date</b>	Final v1.0, 16/11/2023
<b>WP / Task responsible</b>	WP6
<b>Keywords:</b>	Service platform, validation

*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

This deliverable as the final version of the Integrated STAR Platform which supersedes the previous versions, provides the activities and the progress that has been done by the task 6.2-Service Platform Integration and Lab Validation (M5-M30).

Task 6.2 "*Service Platform Integration and Lab Validation*", led by DFKI is a collaboration between pilot partners with other participants from WP3, WP4, and WP5 and the technology providers (INTRA-LU, THA, JSI, QLE, UPRC, UBI, SUPSI). The relationship between the STAR reference architecture and the integrated platform is addressed in this document. The document explains the repository's technologies/tools which are utilized in the STAR project to ease the CI/CD and collaboration between different partners. Furthermore, the essential information (e.g., description of the components, relation to the reference architecture, documentation, installation guideline, dependencies, and test cases) about different artifacts/software developed by technology providers who are coupled to this task and link them to the testbeds' scenarios is presented.

In the conclusive version, we delve deeper into the intricate integration of each WPs' components utilized in the pilot scenarios. Additionally, the document provides a detailed overview of the requirements, activities, and shared information that facilitated effective communication among partners, aligning with the task specifications. Furthermore, a comprehensive analysis of the communication processes and integration of WPs-level components, essential for achieving the goals of the STAR ecosystem is presented.

<b>Deliverable Leader:</b>	DFKI (Hooman Tavakoli)
<b>Contributors:</b>	INTRA-LU, THA, JSI, QLE, UPRC, UBI, SUPSI
<b>Reviewers:</b>	JSI, R2M
<b>Approved by:</b>	INTRA

<b>Document History</b>			
<b>Version</b>	<b>Date</b>	<b>Contributor(s)</b>	<b>Description</b>
V0.1	23/06/2023	DFKI	Initiate the Document
V0.2	02/08/2023	DFKI	Finalising the conclusion and supported scenarios
V0.3	03/08/2023	DFKI, PCL	Updating the use cases and scenarios in each section of components.
V0.4	28/08/2023	DFKI	New section for integration between WPs initiated.
V0.6	02/11/2023	DFKI	DFKI final review
V0.7	07/11/2023	DFKI	Updating based on the Reviewer's input from R2M
V0.8	16/11/2023	DFKI	Finalising the document with remaining comments from final review
V1.0	16/11/2023	INTRA	QA and creation of the final submitted version

# Table of Contents

- EXECUTIVE SUMMARY ..... 2**
- TABLE OF FIGURES..... 6**
- LIST OF TABLES..... 7**
- DEFINITIONS, ACRONYMS AND ABBREVIATIONS ..... 8**
- 1 INTRODUCTION..... 9**
  - 1.1 OVERVIEW AND PURPOSE..... 9
  - 1.2 RELATIONSHIP TO OTHER DELIVERABLES..... 9
  - 1.3 DELIVERABLE STRUCTURE .....10
- 2 FROM REFERENCE ARCHITECTURE TO INTEGRATED PLATFORM ..... 11**
  - 2.1 THE STAR REFERENCE ARCHITECTURE .....12
  - 2.2 WP LEVEL DEPLOYMENT/PHYSICAL DIAGRAMS TO BE USED FOR THE LAB VALIDATION .....13
  - 2.3 PHYSICAL VIEW OF THE STAR CYBERSECURITY MODULES .....14
  - 2.4 PHYSICAL VIEW OF THE STAR ACTIVE LEARNING AND XAI MODULES .....15
    - 2.4.1 *Physical View of Active Learning Module.....15*
    - 2.4.2 *Physical View of the Explainable AI (XAI) Module .....16*
    - 2.4.3 *Physical Views of the Reinforcement Learning Modules .....16*
    - 2.4.4 *Physical View of Safety Zones Detection Module.....16*
    - 2.4.5 *Physical View of Simulated Reality Module.....18*
  - 2.5 PHYSICAL VIEW OF THE STAR HUMAN CENTRIC DIGITAL TWIN MODULES .....19
- 3 SOURCE CODE, REPOSITORY & TOOLS ..... 20**
  - 3.1 TECHNOLOGIES AND TOOLS .....20
  - 3.2 VERSION CONTROL SYSTEM AND REPOSITORY: GIT AND GITHUB.....21
  - 3.3 CONTAINERIZATION .....23
    - 3.3.1 *Docker .....24*
    - 3.3.2 *Dockerfile .....24*
    - 3.3.3 *Docker Compose.....25*
    - 3.3.4 *Docker Usage.....25*
  - 3.4 CONTAINER REPOSITORY & REGISTRY MANAGEMENT .....25
  - 3.5 MANAGEMENT/MONITORING WITH PORTAINER .....26
- 4 THE STAR COMPONENTS ..... 28**
  - 4.1 SECURITY AND DATA GOVERNANCE.....28
    - 4.1.1 *Components.....28*
    - 4.1.2 *Use Cases.....43*
    - 4.1.3 *Inter WP3 integration and communication .....44*
  - 4.2 SAFE, TRANSPARENT AND RELIABLE HUMAN-ROBOT COLLABORATION .....47
    - 4.2.1 *Components.....47*
    - 4.2.2 *Use Cases.....60*
    - 4.2.3 *Inter WP4 integration and communication .....60*
  - 4.3 HUMAN CENTRED SIMULATION AND DIGITAL TWINS .....61
    - 4.3.1 *Components.....61*
    - 4.3.2 *Use Cases.....75*
    - 4.3.3 *Inter WP5 Integration and Communication .....76*
- 5 TESTING, VALIDATION, AND INTEGRATION ROADMAP ..... 78**
  - 5.1 LAB REQUIREMENTS, AND ENVIRONMENT .....78

5.1.1 *General Asset List* ..... 78

5.1.2 *Hardware requirement* ..... 79

5.2 SUPPORTED SCENARIOS ..... 79

5.2.1 *Automation Tools*..... 80

5.2.2 *Validation the Components* ..... 80

5.2.3 *Evaluation the accuracy of the Architectures*..... 80

5.2.4 *The Process of Accessing the Data*..... 80

5.3 INTEGRATION OF TECHNICAL COMPONENTS WITH THE STAR SECURE STORAGE. .... 80

**6 CONCLUSION**..... **83**

**REFERENCES** ..... **84**

## Table of Figures

FIGURE 1: HIGH LEVEL REFERENCE MODEL FOR THE FUNCTIONALITIES OF THE STAR PLATFORM. ....	12
FIGURE 2: STAR FUNCTIONAL MODULES AND LOGICAL VIEW OF THE ARCHITECTURE [STAR-D2.7].....	13
FIGURE 3: DEPLOYMENT DIAGRAM FOR THE CYBERSECURITY MODULES OF THE STAR ARCHITECTURE (I.E., MODULES DEVELOPED IN WP3)- CAPTURED FROM D2.7 .....	15
FIGURE 4: THE DEPLOYMENT DIAGRAM FOR AI SERVICE.....	15
FIGURE 5: PHYSICAL VIEW OF THE STAR XAI COMPONENT/MODULES .....	16
FIGURE 6: PHYSICAL VIEW OF THE SAFETY ZONE DETECTION & FLEET OPTIMIZER MODULES.....	17
FIGURE 7: PHYSICAL VIEW OF THE SAFETY ZONE DETECTION. ....	18
FIGURE 8: PHYSICAL VIEW FOR THE SIMULATED REALITY. ....	19
FIGURE 9: HUMAN DIGITAL TWIN CORE INFRASTRUCTURE DEPLOYMENT SHOWCASE .....	19
FIGURE 10: STAR-AI GITHUB PAGE.....	22
FIGURE 11: A COMPLETE GIT BRANCHING MODEL .....	23
FIGURE 12: AI CYBER-DEFENCE TOOL INTERNAL ARCHITECTURE.....	34
FIGURE 13: CONSOLE OUTPUT OF DOCKER PS .....	35
FIGURE 14: SSPM HIGH LEVEL ARCHITECTURE .....	41
FIGURE 15: WP3 ARCHITECTURE AND DESIGNED APIs .....	45
FIGURE 16: ARCHITECTURE OF THE COMPONENTS FOR NATURE LANGUAGE PROCESSING .....	55
FIGURE 17: INTERACTION BETWEEN WP4 COMPONENTS. ....	61
FIGURE 18: SAFETY ZONE DETECTION & AMR FLEET OPTIMIZER PHYSICAL VIEW .....	62
FIGURE 19: DEPLOYMENT VIEW OF HDT COMPONENTS .....	69
FIGURE 20: WP5 MAIN INTEGRATIONS .....	77

## List of Tables

TABLE 1: EDGE/CLOUD DEPLOYMENT CONSIDERATIONS FOR THE MAIN COMPONENTS OF THE STAR ARCHITECTURE FROM D2.7 .....	13
TABLE 2: INDICATIVE OUTPUT OF AI CYBER-DEFENCE TOOL .....	36

## Definitions, Acronyms and Abbreviations

Acronym/ Abbreviation	Title
<b>API</b>	Application Programming Interface
<b>CE</b>	Community Edition
<b>CLI</b>	Command Line Input
<b>CRUD</b>	Create Read Update Delete
<b>DLSDR</b>	Distributed Ledger Services for Data Reliability
<b>DoA</b>	Description of Action
<b>DSL</b>	Domain-Specific Language
<b>EAE</b>	Edge Analytics Engine
<b>GUI</b>	Graphical User Interface
<b>JSON</b>	JavaScript Object Notation
<b>JVM</b>	Java Virtual Machine
<b>MVP</b>	Minimum Viable Product
<b>OEM</b>	Original Equipment Manufacturer
<b>P2P</b>	Peer-to-Peer
<b>RL</b>	Reinforcement Learning
<b>RMS</b>	Runtime Monitoring System
<b>SDK</b>	Software Development Kit
<b>SLA</b>	Service Level Agreement
<b>URI</b>	Universal Resource Identifier
<b>URL</b>	Universal Resource Locator
<b>UUID</b>	Universally Unique Identifier
<b>WP</b>	Work Package
<b>XSD</b>	XML Schema Definition
<b>DB</b>	Database



# 1 Introduction

## 1.1 Overview and Purpose

The AI approaches are becoming more favoured approaches in the research and industrial environment. One vital aspect of employing AI systems is considering the reliability and safety of the AI systems, and it becomes more demanding when the system is planned to be utilized in the industrial environment. The main goal of the STAR project is to research, implement validate and demonstrate the trusted AI technologies related to the production lines and manufactory's scenarios. The STAR project provides a holistic approach to tackle a wide range of trustworthy approaches from data reliability to cybersecurity and AI system explainability. The STAR project designs and implements multiple prototype systems including systems for data provenance and traceability, cyber-defence against some of the most prominent attacks that target AI systems, Explainable AI (XAI) algorithms, human-centric AI-based systems such as human-centric digital twins, systems for the trusted and safe operation of mobile robots in production lines, human-robot collaboration systems, simulated reality systems for effective cobots, to name but a few.

The STAR project's intentions are applied in the industrial environment and need to be proven and tested in different testbeds with different scenarios. WP6 aims to integrate, validate, and evaluate the STAR goals in the various testbeds with different use cases. Task 6.2 "Service Platform Integration and Lab Validation", in this work package which starts from M5 and continues till M30 of the project's lifespan, focuses on the integration of the project technical development and prototyping in the STAR platform for secure and safe AI in the manufacturing area.

In this task, the main purpose is to focus on the integration of the technology providers' components into the STAR platform. Considering that the vast amount of the components in the STAR projects are software or middleware, it is essential to utilize the modern approaches for CI/CD (Continuous Integration / Continuous Deployment), based on the DevOps principles and tools. Furthermore, in this task, we consider the approaches which facilitate the packaging and distribution of the software/ middleware components, like leveraging the containerization approaches (e.g., Docker images).

The integration process in this task is driven by the reference architecture of the STAR project. Moreover, the interface between different components of the platform from different technology providers is one of the key points fulfilled within this task. Finally, for the validation phase, we considered that the integrated platform and its components, and functionalities are validated in different use cases from different pilots to identify and implement improvements to the various part of the integrated platform.

## 1.2 Relationship to Other Deliverables

In this section, other deliverables related to the D6.4 are listed. In addition, we address why various deliverables are related to this document.

D2.7- "STAR Reference Architecture and Blueprints".

In the STAR reference architecture, we model the relationship between different technologies which are categorized into three clusters that build the STAR AI platform. Since in this deliverable, we focus on the test, validation, and integration of different STAR components/technologies into the STAR platform, it is inevitable to have a wide perspective

on how these different technologies bind and communicate together. Moreover, for the Physical diagram for the lab validation, which is addressed in section 2, we leverage this deliverable.

D2.5- "Data Models and Data Collection".

In the deliverable 6.4, we list different components and provide test cases related to the inter-component and inter-WPs as the atomic test. For this reason, there is a crucial binding between the data models and data collection and this deliverable.

D3.1 and D3.2 "Decentralized Reliability for Industrial Data and Distributed Analytics".

The D3.1 and D3.2 are focusing on the project's decentralized approach for provenance and tracking of industrial data utilized in AI systems. The Distributed Ledger Services for Data Reliability (DLSDR) (subsection 4.1.1.2) as a component for Security and Data Governance (Section 4.1) majorly references to this deliverable.

## 1.3 Deliverable Structure

In this deliverable we report on:

- The reference architecture and link that to the integrated platform. Moreover, we provide the WP-level deployment/physical diagram used for the lab validation.
- The repository initiated for the artifacts, software, and source codes from technology providers.
- The components that are utilized in the STAR project to address the Pilots' use cases. These components are categorized into three domains: "Security and Data Governance", "Safe, Transparent and Reliable Human-Robot Collaboration", and "Human Centered Simulation and Digital Twin". We explain the different components and their relation to the use cases. Furthermore, we establish the integration and relationships among the components within the WPs across these three categories. This approach ensures a comprehensive understanding of the interconnections and relationships among various technologies.
- Ultimately, within this task, we have conducted a thorough list of activities encompassing the testing, validation, and integration of diverse artifacts within the integrated platform. This encompassed fulfilling lab requirements, initializing the platform, and addressing supported scenarios pertinent to the task's objectives. Additionally, we focus on integrating the components to encompass the entire STAR ecosystem. This integration aids us in gaining a comprehensive perspective on the interoperability requirements among different artifacts and pilots at the project level.

## 2 From reference architecture to integrated platform

The STAR project is facilitated by a wide range of systems and functionalities. The project takes a comprehensive approach that addresses multiple aspects of AI trustworthiness from data reliability to the cybersecurity and explainability of AI systems. Consequently, the STAR project designs and implements multiple prototype systems, systems for data provenance and traceability, cyber-defence against some of the most prominent attacks that target AI systems, Explainable AI (XAI) algorithms, human-centric AI-based systems such as human-centric digital twins, systems for the trusted and safe operation of mobile robots in production lines, human-robot collaboration systems, simulated reality systems for effective cobots and more.

Although each of the STAR-related prototype systems can be researched, implemented, and tested independently, it is inevitable to have a holistic reference architecture that connects them since most of them are intricately connected. For instance, the XAI systems can be used to support the operation of cyber-defence strategies (e.g., by detecting abnormalities in their operation), as well as the operation of simulated reality systems (e.g., through helping in the production of reliable data to set up the simulated environment). In this context, STAR is not limited to researching each of the above systems individually. It also explores ways and methods that could boost the optimal interplay and integration of the above systems towards a holistic and efficient approach to trusted AI in manufacturing.

For the reason of the optimal integration of the STAR AI systems, the STAR project also researched the structuring principles that provide the optimal integration of the various AI prototyping and documents the software architecture that reflect these structuring principles. In this direction, the project introduced a reference architecture model that can support the development, deployment, and operation of end-to-end integrated systems for trusted AI in industrial environments. The model is characterized as “reference” as it is not limited to supporting the integration and deployment of the STAR platform. It is also destined to serve as a blueprint for a wider class of trusted AI systems i.e., helping integrators of AI solutions to develop and deployed trusted AI in dynamic manufacturing environments.

The STAR reference architecture (STAR-RA) model considers the specifications and functionalities of the various AI building blocks of the project’s solutions and provides a set of fundamentals for their integration into trusted AI solutions. As previously discussed, the STAR architecture model is aimed at being abstract and general to support the development of trusted AI beyond the boundaries of the project. To this end, the model is developed based on principles and concepts of existing reference architecture models for Industry 4.0, the Industrial Internet of Things (IIoT), and Bigdata systems. Especially, existing models are extended and/or customized to address the STAR project’s trusted AI vision. Hence, the STAR-RA model aims to be usable and exploitable by the numerous manufacturers and AI/IIoT solution providers for manufacturing environments, which can already be fitted to existing reference architectures for Industry 4.0.

The development and research activities of the project, the specification of the reference scenarios for trusted AI in manufacturing, the development of the various modules of the STAR platform and solutions, the specification/evolution of Industry 4.0 and AI standards, as well as the collection and management of data for training and developing AI systems have direct influence and therefore have an essential impact on the development and validation of

the STAR architecture. Hence, the agile approach to specify, validate, and document the STAR-RA is chosen. Especially, the STAR-RA is specified in two iterations. The first iteration was driven by the project’s activities during the first semester of STAR’s lifetime, while in the second phase, the STAR-RA incorporated inputs from the final versions of the STAR reference scenarios and technologies specifications. Moreover, between the two iterations, the project used the first version of the architecture to drive the development and integration of technical/technological systems in WP3, WP4, WP5, and WP6. This enabled the project to receive feedback from the actual, implementation, deployment, and use of the first version of the architecture. Accordingly, the project used this feedback to fine-tune the specification of the architecture. Consequently, there has been an inevitable bind between the reference architecture of the STAR project and scenarios from testbeds and technology providers.

## 2.1 The STAR Reference Architecture

The major functionalities of the STAR platform can be clustered into three categories. Figure 1 illustrates the high-level reference model for the functionalities of the STAR platform. The mentioned domains are as follows:

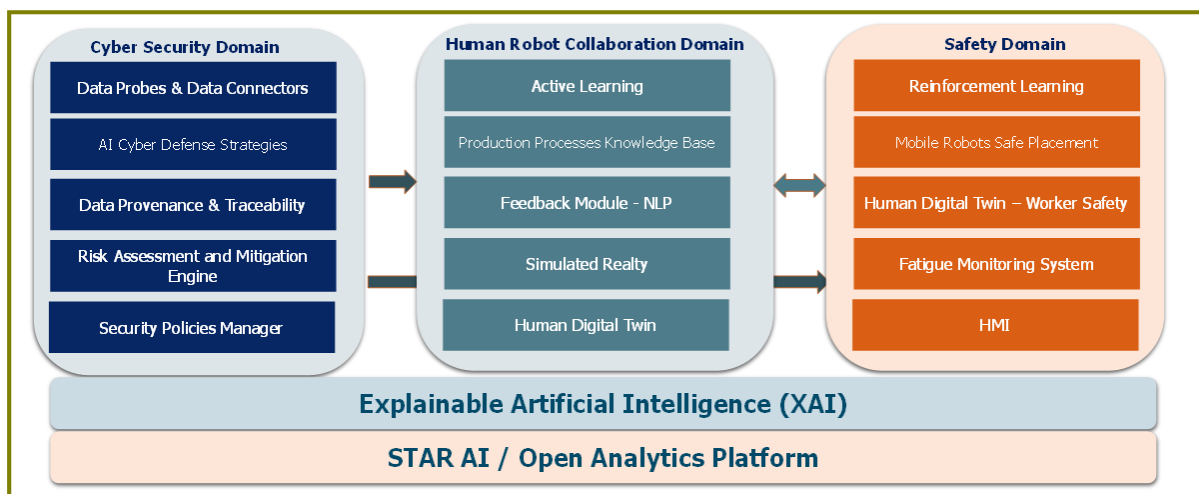


Figure 1: High Level Reference Model for the Functionalities of the STAR Platform.

**Cyber Security Domain.** This block contains functionalities that ensure the reliability and security of industrial data and AI algorithms that are trained and tested based on them. The functionalities of these domains support and reinforce the trustworthiness of the project’s functions in the other two domains.

**Human-Robot Collaboration Domain.** Provides the functionalities to fulfil the trusted collaboration between robots and workers in the industrial environment.

**Safety Domain.** Provides the safety of industrial operations containing the workers and/or autonomous systems.

As depicted in Figure 1, the functionalities of the three mentioned domains depend on the XAI and AI algorithms. The XAI plays a crucial role in the operation of the security platform by supporting the defence strategies in the cybersecurity domain, data generation in a simulated reality, and active learning functionalities in the human-robot collaboration as well as the development of human digital twins in the safety domain.

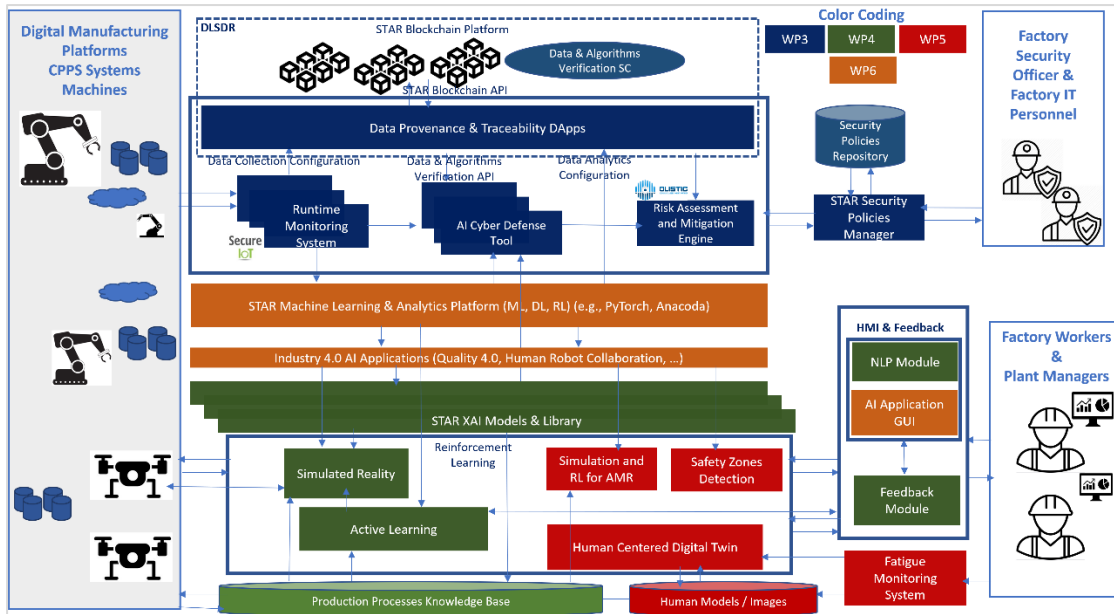


Figure 2: STAR Functional Modules and Logical View of the Architecture [STAR-D2.7]

In Figure 2, the STAR functional modules, as well as the logical perspective of the architecture, are presented. As illustrated in the image above, the STAR platform starts with receiving the data from the factory environment and providing different types of services and functionalities to the cyber-security teams of the factory and also to other factory stakeholders. (e.g., industrial engineers, plant managers, factory workers).

In Figure 2 different modules from each WPs and their relations to others are presented.

- For the WP3, Data and Algorithms Verification SC, and Security Policies Repository are the focused modules.
- For the WP4, STAR XAI Models and Library, Simulated Reality, NLP Module, and Feedback Module, and Active Learning are the target components.
- For the WP5, labeled with red color, we have Fatigue Monitoring System, human-centered Digital Twin, RL systems and AMR Safety, and Human Model Images.
- AI Application GUI, STAR Machine Learning and Analytics Platform, Industry 4.0 AI Applications, and Production Process Knowledge Base are the related modules to the WP6.

## 2.2 WP level Deployment/Physical Diagrams to be used for the Lab Validation

The physical deployment of the STAR platform mainly refers to the cloud/edge deployment model. Based on a different feature of the operations (e.g., Datapoints availability, energy efficiency, low latency-real-time performance, and privacy), they can be deployed in cloud or edge servers.

The main components of the STAR architecture can have different physical deployment choices. In Table 1, the major STAR components which are needed to be deployed as a part of the lab validation and the different options for the physical deployments are illustrated.

Table 1: Edge/Cloud Deployment Considerations for the main components of the STAR architecture from D2.7

Component Name	Physical Deployment Choice
Data Probes / Data Connectors	Cloud/Edge, specifically: Cloud: Monitoring Engine; Edge: Data Collectors (Beats)
STAR Blockchain (DLT)	Cloud
AI Cyber Defence Strategies	Cloud
Risk Assessment and Mitigation Engine (RAME)	Cloud
Security Policies Manager (SPM)	Cloud/Edge, specifically: Cloud: Policy Management Engine, Policy Validation; Edge: Policy enforcement, Policy Validation
XAI Library	Cloud/Edge
Simulated Reality	Cloud/Edge
Active Learning (AL)	Cloud
NLP Module (incl. TTS, STT, Sentiment Analysis)	Cloud
Production Processes Knowledge Base	Cloud
Feedback Module	Cloud

The physical view in this task of test, validation, and integration of the lab environment provides us a wide perspective and clear view of how different components at the WP level require separate cloud application server(s). In the following, we describe different components briefly. The details about this section can be reached through the D2.7.

### 2.3 Physical View of the STAR Cybersecurity Modules

Figure 3 illustrates the complete deployment diagram/physical view of the components from WP3, which can be containerized in Docker images. In this figure, the number of the 7 VMs (Application Server) for different components is presented. Moreover, the minimum requirements for each component to be deployed are mentioned. This STAR security and data governance for AI systems in manufacturing infrastructure consists of DLSDR infrastructure, Runtime Monitoring System, Cyber-Defence Strategies, Policy Manager, and Risk Management and Mitigation Engine components/artifacts.



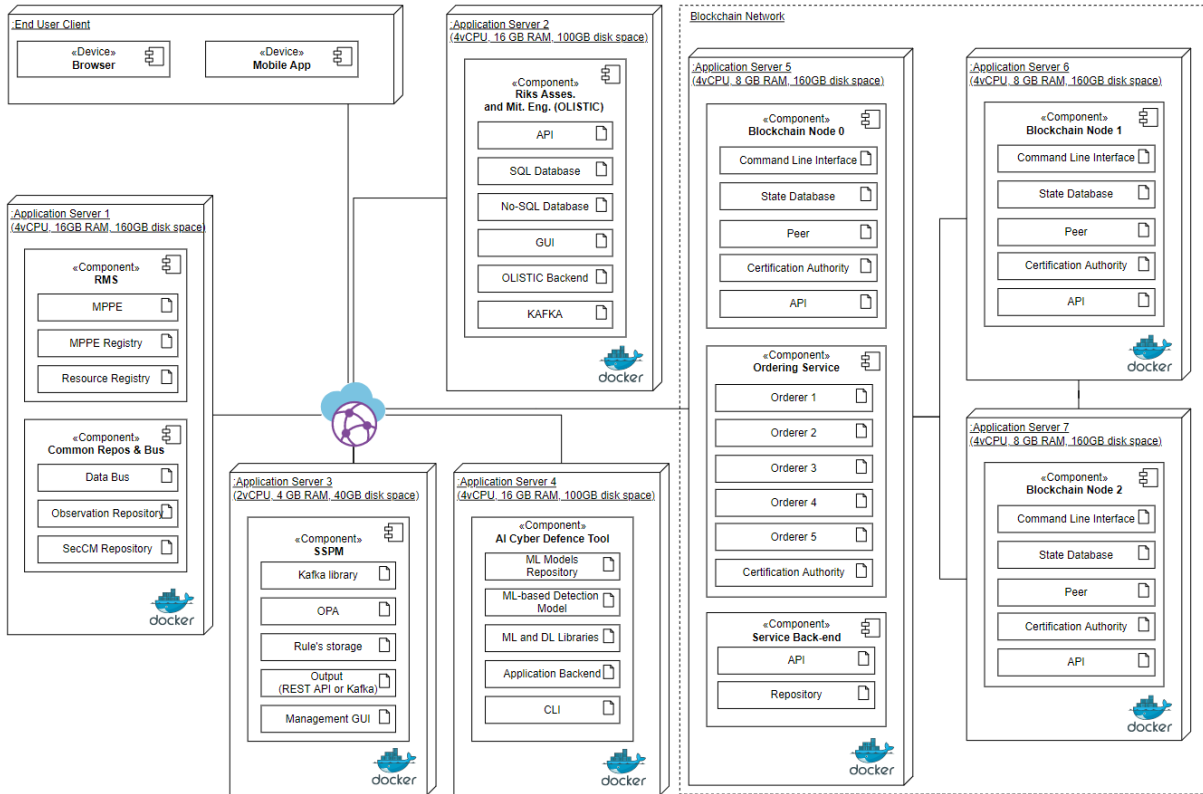


Figure 3: Deployment Diagram for the Cybersecurity Modules of the STAR Architecture (i.e., modules developed in WP3)- Captured from D2.7

## 2.4 Physical View of the STAR Active Learning and XAI Modules

The source code for the Active Learning and XAI modules is managed in private repositories. The final version of the XAI service offers the Machine learning models functionalities through the REST API.

### 2.4.1 Physical View of Active Learning Module

STAR project specifically utilizes the Supervised learning approach of the active learning module. Figure 4 illustrates the integration between gateway services with some machine learning models and their communication with AL services.

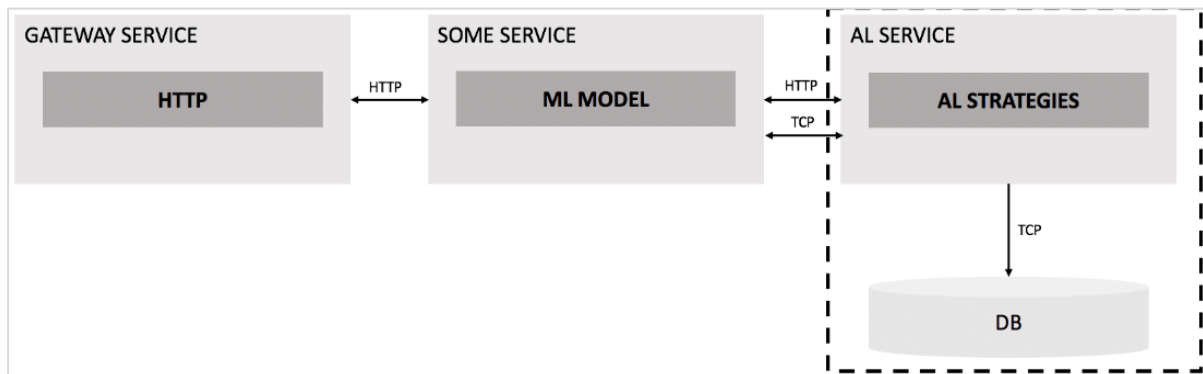


Figure 4: The Deployment Diagram for AL Service.

### 2.4.2 Physical View of the Explainable AI (XAI) Module

XAI for the STAR project is proposed to be deployed to a Kubernetes environment (Figure 5) and can be deployed in a containerized microservice. Regarding the data management and communication with other components of the platform Kafka queue, and Zookeeper, JDBC and REST API connection is utilized.

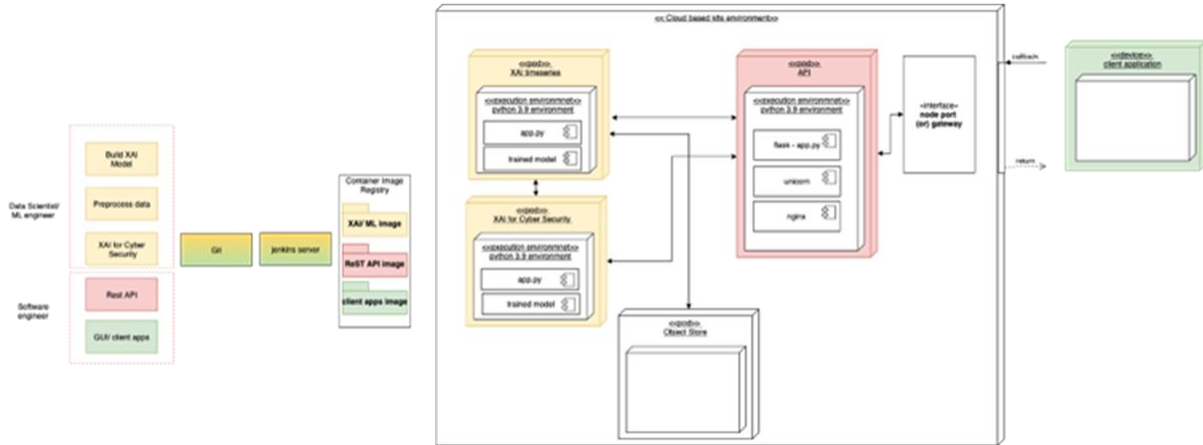


Figure 5: Physical View of the STAR XAI Component/Modules

### 2.4.3 Physical Views of the Reinforcement Learning Modules

The RL module consists of two different modules from WP4, and WP5.

1. Safety Zone Detection Module
2. Simulated Reality Module

### 2.4.4 Physical View of Safety Zones Detection Module

The safe movement and collaboration between workers and Automotive Mobile Robot (AMR) are based on two modules:

1. Safety Zone Detection.
2. AMR Fleet Optimizer.

The physical view of the two modules is depicted in Figure 6. Moreover, it is shown in this diagram that the two modules are communicating to each other relying on the Human Centric Digital Twin through an MQTT Broker. The main output of the safety zone detector is an “average spatial heatmap” representing a probabilistic occupancy of the production lines based on fixed RGB cameras deployed in the factory. This description represents the global environment, including human and object location as occupancy cells. The results from object/human detection and localisation allow to update the heatmap representation. Indeed, these heatmaps are simplified and anonymized representations of the occupancy of the work floor in real time. It serves to feed the reinforcement learning module. The Safety Zone detection module publishes the position of the occupied cells and the AMR Fleet Optimizer subscribes for this message and then forecasts Robotino trajectories in order to avoid potential crowded areas.



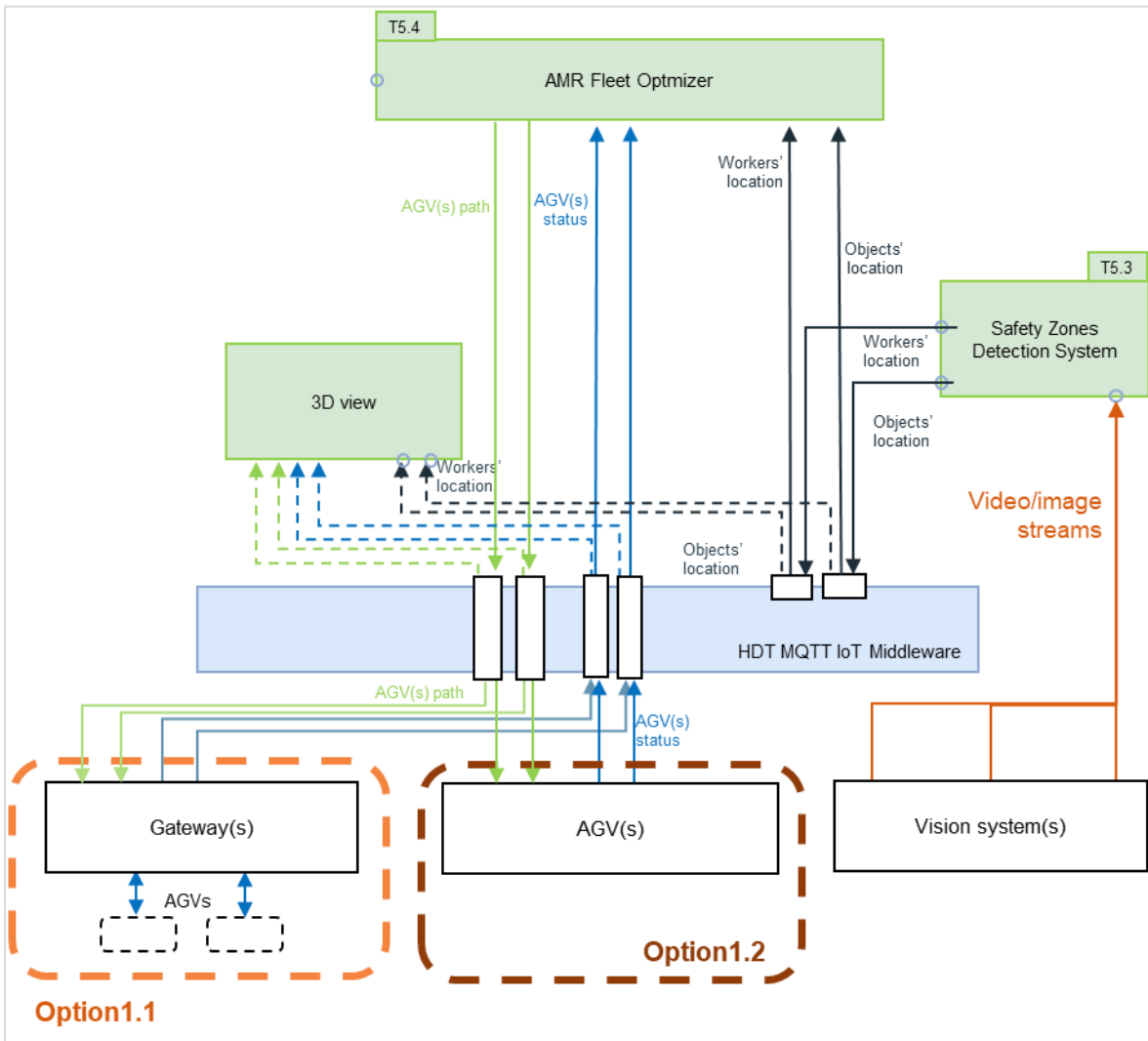


Figure 6: Physical View of the Safety Zone Detection & Fleet Optimizer modules.

Figure 7 presents each subcomponent of the Physical view of the Safety Zones Detection System. The systems start with exploiting videos from ceiling-mounted cameras in the testbed environment as input and will deliver the spatial heatmaps (worker location and object location) as results of the analytics. In the middle of the figure, the fusion 3D scene component combines the two deep learning algorithms namely:

1. The skeleton extraction to follow the human gesture and pose,
2. the detection and classification of unknown non-static objects in the scene.

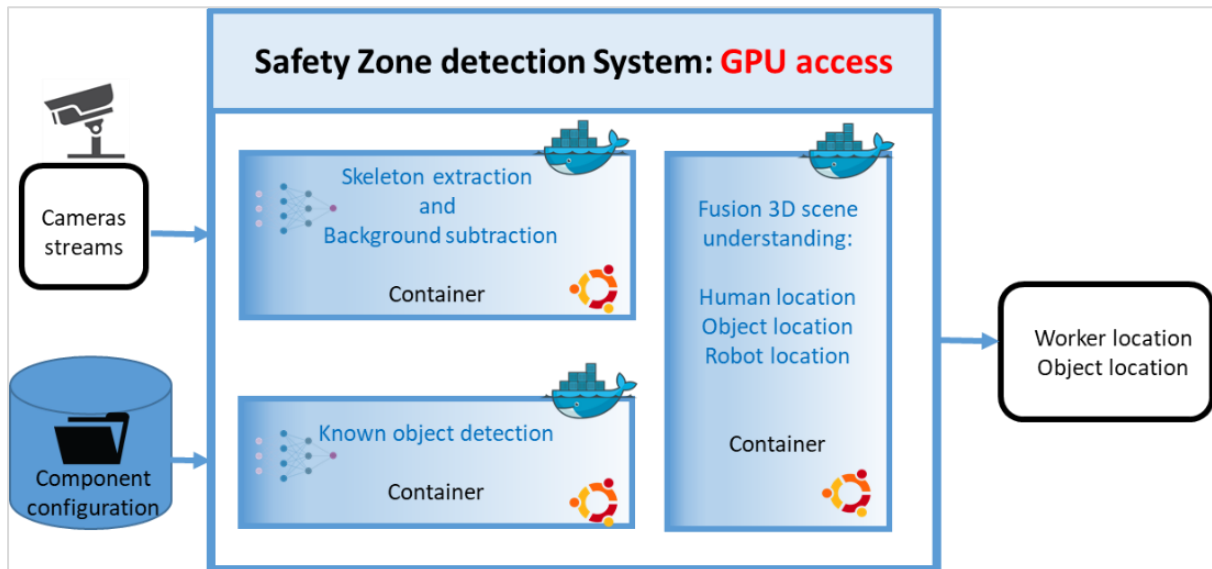


Figure 7: Physical View of the Safety Zone Detection.

All these components are dockerized to facilitate the deployment.

### 2.4.5 Physical View of Simulated Reality Module

The Simulated Reality component utilizes different batch jobs to generate synthetic data. The output of these batch jobs will be consumed by live services that will deliver synthetic data upon request, potentially in a way that can also categorize the data as easy or difficult to classify through the Confidence Assessment subcomponent. All subcomponents will need access to a common data store or shared filesystem containing the Input Dataset and Auxiliary Data needed including weights for pre-trained models (Figure 8). Their outputs will consist of a trained generator like generator model from GAN that can produce synthetic data upon request. These need to be accessible through the Data Serving services (e.g., through a Shared File System). Data Serving should return synthetic data fitting a certain use-case specification upon request together with a potential confidence assessment. The transferring and communication can be done with the REST API.

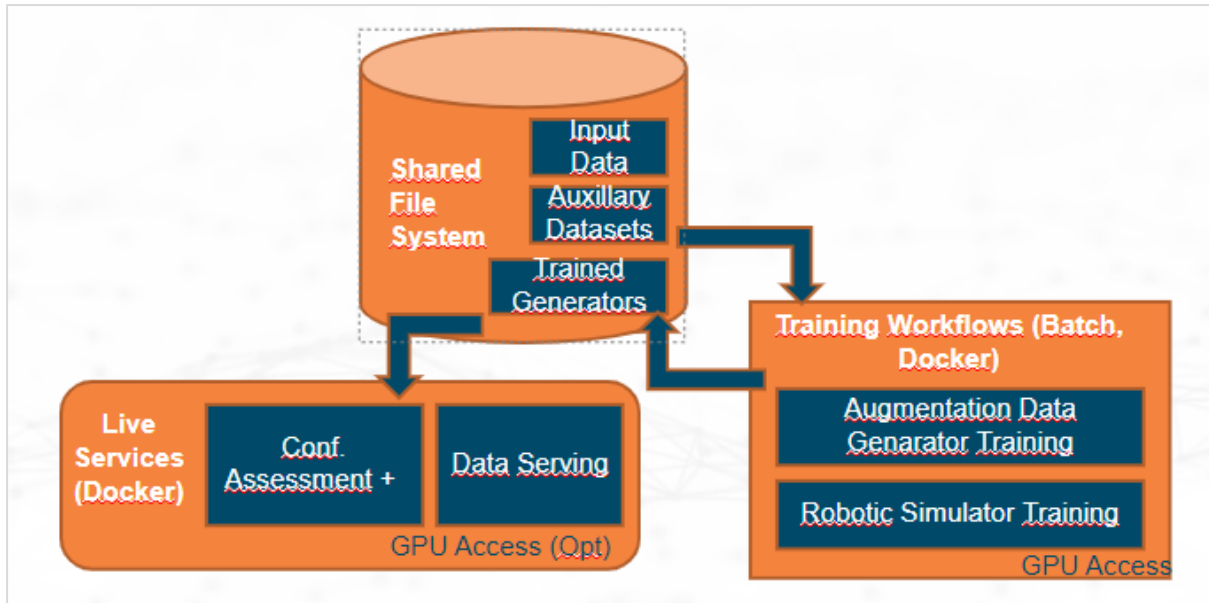


Figure 8: Physical View for the Simulated Reality.

## 2.5 Physical View of the STAR Human Centric Digital Twin Modules

The Human Digital Twin (HDT) and its components are deployed as cloud applications (Figure 9: Human Digital Twin Core Infrastructure deployment showcase). Specific instances are deployed to support the different use cases of the project.

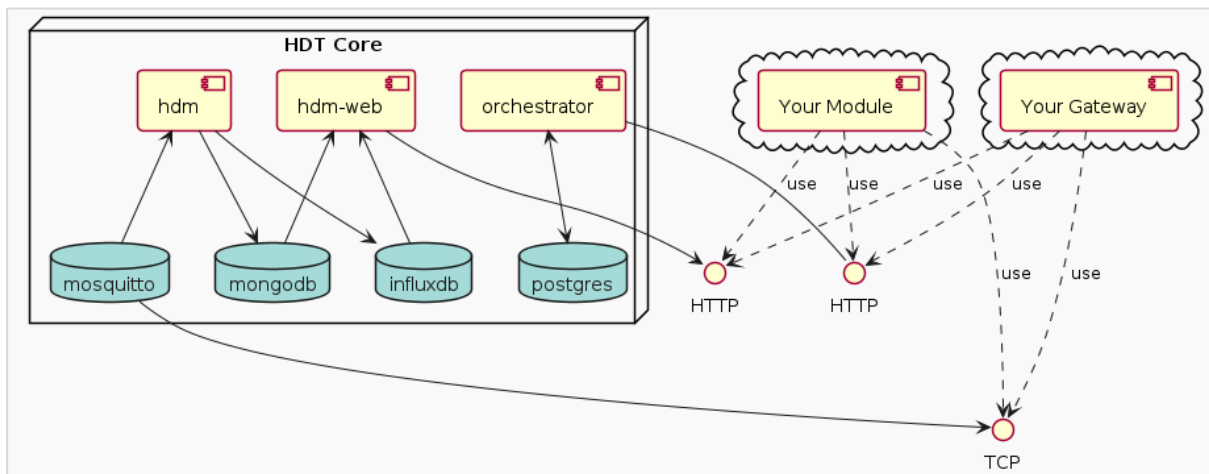


Figure 9: Human Digital Twin Core Infrastructure deployment showcase

The HDT Core Infrastructure is an extensible and flexible IIoT based platform supporting the creation of customised data representations of production systems and their entities, including humans. Thanks to its modular infrastructure with interchangeable components, which ease the digital twin instantiation and ramp-up, the HDT is applied in two different STAR use-cases: DFKI pilot supporting the integration AMR Fleet Optimizer and Safety Zone Detection; PHILIPS pilot supporting data collection from wearable devices and quality control through active learning.

## 3 Source Code, Repository & Tools

### 3.1 Technologies and Tools

This section provides a list of code management, packaging and deployment tools, along with their high-level description, that facilitate the STAR software development teams with the integration and deployment and validation of the STAR solution. A summarization of the technologies and software tools follows below:

- **Hetzner<sup>1</sup>**: The software components that comprise the STAR development platform (i.e. artifact repository) have been deployed on virtual hosts, which are, in essence, cloud servers instantiated on a public cloud provider, named Hetzner Cloud.
- **Git<sup>2</sup>**: A free and open-source distributed Version Control System (VCS). It is used for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. It has been designed to handle everything from small to very large projects with speed and efficiency. A Git repository (or repo for short) contains all of the project files and the entire revision history.
- **GitHub<sup>3</sup>**: A web-based open-source Git repository hosting service. It offers a graphical interface with several built-in features, such as version control, issue tracking, code review, wiki, etc. Multiple developers can concurrently create, merge and delete parts of the code they are working on independently, at their local system before applying the finalized changes to a shared GitHub repository.
- **Docker<sup>4</sup>**: A set of Platform-as-a-Service (PaaS) products that use OS-level virtualization to deliver software in lightweight packages called containers. Docker can package an application and its dependencies in a virtual container that can run seamlessly on any Linux, Windows, or macOS computer. This enables the application to run in a variety of locations, such as on-premises, in a public cloud, and/or in a private cloud.
- **Docker Daemon<sup>5</sup>**: Services running on multiple development virtual servers for the deployment of containerized services. Docker Daemon is a background process that manages Docker images, containers, networks, and storage volumes. The Docker Daemon constantly listens to Docker API requests and processes them.
- **JFrog Container Registry<sup>6</sup>**: An application that implements a private Docker Registry in which one can store and distribute the Docker images of the projects' artifacts. It is used to securely control where the images are being stored awaiting containerization, thus integrating image storage and distribution tightly into the STAR development workflow. For the needs of the STAR project, a self-hosted JFrog Container Registry instance has been deployed to Hetzner Cloud.
- **Portainer<sup>7</sup>**: An open-source tool for managing container-based applications in various virtualization environments. It can be used to set up and manage the environment,

---

<sup>1</sup> Cloud provider's official website: <https://www.hetzner.com/cloud>

<sup>2</sup> Git official website: <https://git-scm.com/>

<sup>3</sup> GitHub official website: <https://github.com/>

<sup>4</sup> Docker official website: <https://www.docker.com/>

<sup>5</sup> Docker overview on official website: <https://docs.docker.com/get-started/overview/>

<sup>6</sup> JFrog Container Registry official website: <https://jfrog.com/container-registry/>

<sup>7</sup> Portainer official website: <https://www.portainer.io/>

manage containers lifecycle, monitor application performance, triage problems, and enable role-based access control. For the needs of the STAR project, a Portainer instance has been deployed to Hetzner Cloud.

## 3.2 Version Control System and Repository: Git and GitHub

During collaborative software development projects, the source code is usually stored in shared remote repositories, accessible by all team members with various permission levels. Version control, also known as source control and revision control is the practice of tracking and managing changes to code stored in such repositories. Consequently, Version Control Systems (VCS) can be defined as software tools that help software teams manage changes to source code over time. Their value is twofold: on the one hand, they keep track of every modification to the code in a special kind of database, thus allowing reversion of the code to previous states in case a breaking bug is introduced; on the other hand, by employing clever branching strategies, they enable developers to simultaneously work collaboratively on the same codebase, without cancelling one another's effort.

As mentioned in D2.7 [STAR-D2.7], the STAR component's code management is based on two popular open-source technologies, Git and GitHub<sup>8</sup>. Git serves as the Version Control Systems (VCS), while GitHub is a powerful and intuitive Git repository hosting service. The latter offers a web-based graphical interface with several built-in features. It allows the creation of collaboratively owned and maintained code repositories, code branching and merging, version control, issue tracking, code review, wikis, etc. Multiple developers can concurrently create, merge and delete parts of the code they are working on independently at their local system, before pushing the changes back to branches of the shared GitHub repository. The instantiated STAR GitHub Organization can be found under the following URL: <https://github.com/star-eu> (see Figure 10 below). Under this Organization, several repositories and Teams for the STAR components and services have been created.

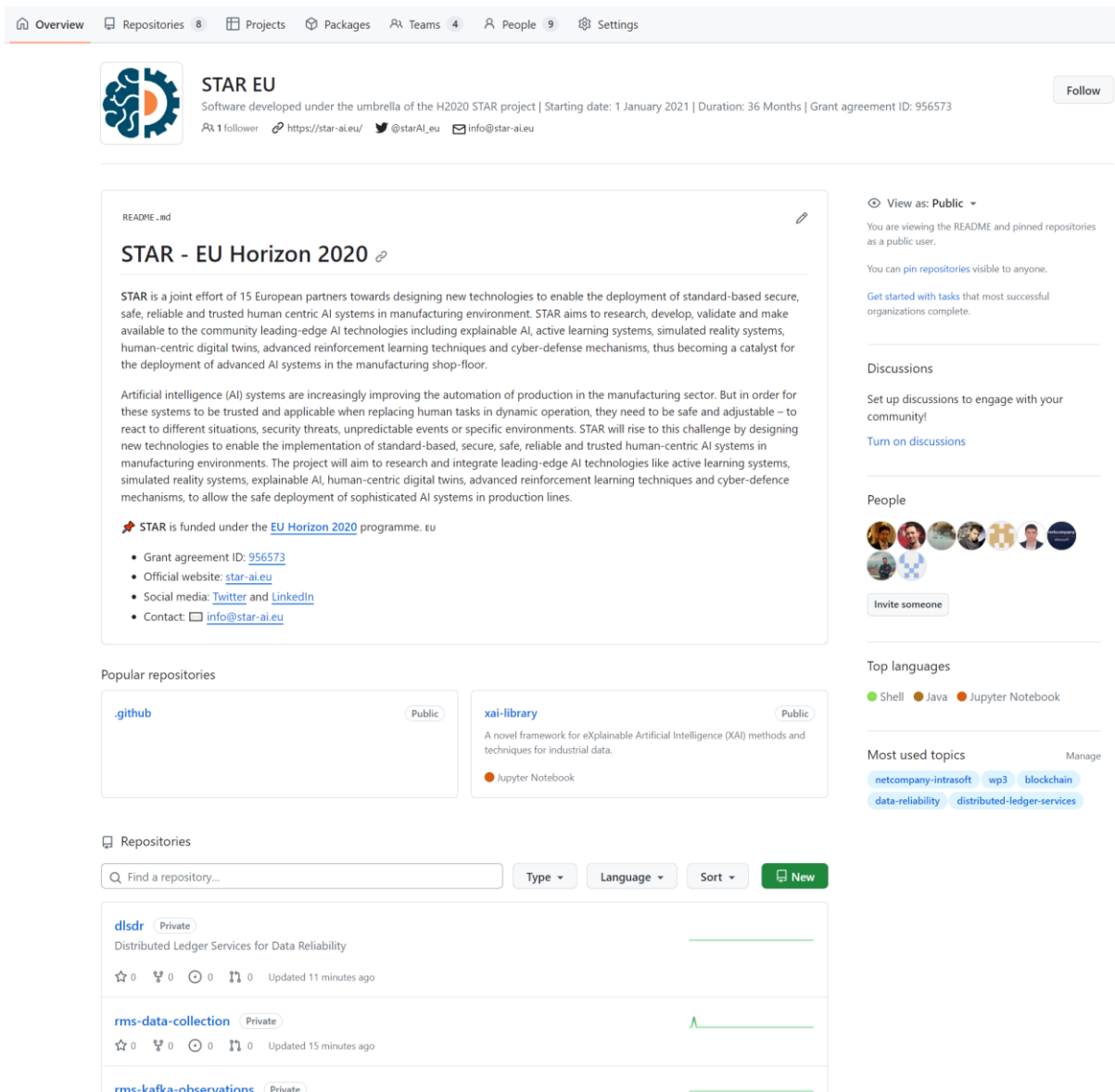
The private repositories GitHub STAR Organization host are accessible only to the project partners developing each tool and, in addition, to INTRA for administration purposes. Additionally, some of the repositories (i.e., XAI-Library<sup>9</sup>) which are offered as open source can be accessed and downloaded by anyone but code can be committed only from the repository's maintainers. Each component repository should include a project with source code and scripts for executing the testing and deployment pipelines. The minimal required files for such a project are the following:

- **Dockerfile:** A text-based script of instructions that is used to create a container image.
- **README.md:** A mark-up file with information on the module's purpose and deployment instructions.

---

<sup>8</sup> <https://github.com/>

<sup>9</sup> <https://github.com/star-eu/xai-library>



The screenshot shows the GitHub profile for STAR EU. At the top, there is a navigation bar with links for Overview, Repositories (8), Projects, Packages, Teams (4), People (9), and Settings. The profile header includes the STAR EU logo, the name 'STAR EU', and a bio: 'Software developed under the umbrella of the H2020 STAR project | Starting date: 1 January 2021 | Duration: 36 Months | Grant agreement ID: 956573'. It also shows 1 follower and social media links for GitHub, website, Twitter, and email. A 'Follow' button is present.

The main content area features a pinned README file titled 'STAR - EU Horizon 2020'. The README text describes STAR as a joint effort of 15 European partners to design secure, reliable, and trusted human-centric AI systems for manufacturing. It details the project's goals, including research in explainable AI, active learning, simulated reality, and human-centric digital twins. A funding note states: 'STAR is funded under the EU Horizon 2020 programme. eu'. A list of links includes the grant agreement ID (956573), official website (star-ai.eu), social media (Twitter and LinkedIn), and contact email (info@star-ai.eu).

Below the README, there are sections for 'Popular repositories' (showing .github and xai-library), 'Top languages' (Shell, Java, Jupyter Notebook), and 'Most used topics' (netcompany-intrasoft, wp3, blockchain, data-reliability, distributed-ledger-services). A 'Discussions' section is also visible.

At the bottom, the 'Repositories' section is shown with a search bar and filters for Type, Language, and Sort. It lists three repositories: dlsdr (Private), rms-data-collection (Private), and rms-kafka-observations (Private), each with a progress bar and update time.

Figure 10: STAR-AI GitHub page

Well-structured Git repositories usually follow a specific branching model in order to guide the developers on the commit methodology. A proposed branching model for STAR (or part of it) has been introduced by [Driessen 10] and a complete version of which is shown in Figure 11 below.

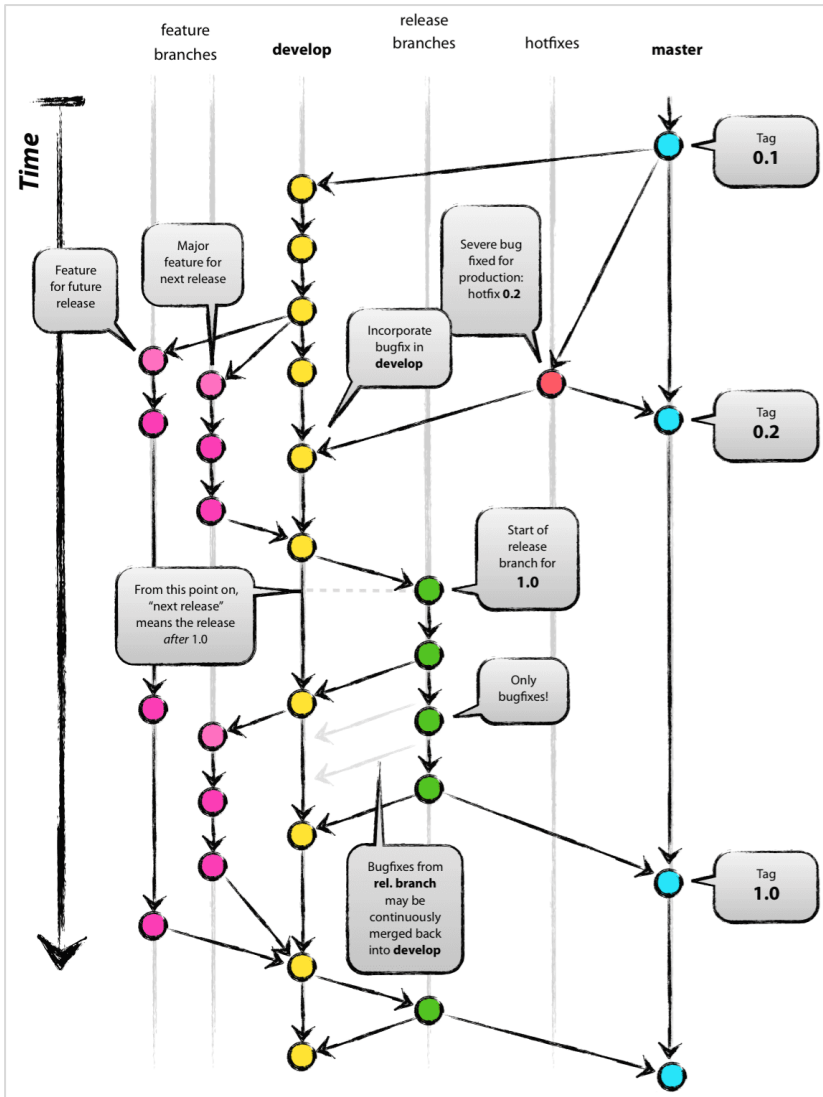


Figure 11: A Complete Git branching model<sup>10</sup>

Note that the development of a specific branching strategy remains at the discretion of each STAR component’s development team and is not restricted to a specific model.

### 3.3 Containerization

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

<sup>10</sup> <https://nvie.com/posts/a-successful-git-branching-model/>



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.

### 3.3.1 Docker

As mentioned in D2.7 [STAR-D2.7] for the STAR software packaging we have considered Docker<sup>11</sup> images which is currently the dominant technology/methodology and is considered a de facto. A Docker image is a file, comprised of multiple layers, used to execute code in a Docker container. An image is essentially built from the instructions for a complete and executable version of an application, which relies on the host OS kernel. Docker is an open platform for developing, shipping, and running applications. With Docker, an infrastructure can be managed in the same way applications are managed. Docker offers shipping, testing, and deploying methodologies easily and quickly, where the time between writing code and running it in production can be significantly reduced.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don't need the extra load of a hypervisor but run directly within the host machine's kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actual virtual machines [Docker].

Docker images can be published in a shared repository, such as the Docker Registry<sup>12</sup> or DockerHub<sup>13</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.

### 3.3.2 Dockerfile

Every Docker container starts with a simple text file containing instructions for how to build the Docker container image. This file, which is by convention named "*DockerFile*" without any file extension, automates the process of Docker image creation. It is essentially a list of command-line interface (CLI) instructions that Docker Engine will run in order to assemble the image. Dockerfiles can be as complicated as needed; they might even contain multiple parent Docker images, as well as some Build-Automation commands.

---

<sup>11</sup> <https://docs.docker.com/>

<sup>12</sup> Docker Registry official website: <https://docs.docker.com/registry/> (accessed August 2021)

<sup>13</sup> DockerHub official website: <https://hub.docker.com/> (accessed August 2021)



Last but not least, repositories might also contain a `“.dockerignore”` file. Such files allow developers to mention a list of files and/or directories which they might want to ignore while building the image. This would definitely reduce the size of the image and also help to speed up the Docker build process.

### 3.3.3 Docker Compose.

As mentioned in D2.7 [STAR-D2.7] Docker Compose is a tool for defining and running multi-container Docker applications. It uses YAML files to configure the application's services and performs the creation and start-up process of all the containers with a single command. The `“docker-compose.yml”` file is used to define an application's services and includes configuration options. In STAR as the preferred container runtime management method was Docker Compose every component is accompanied by a `“docker-compose.yml”` file which facilitates its installation. Additionally, different collections of interoperable components that are used as solutions for the STAR use cases are provided as ready to install `“docker-compose.yml”` files.

Information on how to edit a `“docker-compose.yml”` file can be found at Docker Docs [Docker] and more specifically at the Get started with Docker Compose<sup>14</sup>.

### 3.3.4 Docker Usage

There are many tutorials to containerize an application or a system and offer it thru a repository management service which span from beginners to more advanced ones depending on the technologies used. An intermediate one that doesn't focus on a specific technology and provides the relevant aspects that are necessary to establish a well-defined contract between Dev and Ops teams can be found in [Souza18], which provides a checklist on how to “dockerize” any application. Specifically, the following steps are suggested:

- Choice of a base Image.
- Installation of the necessary packages.
- Addition of custom files.
- Definition of users that will run your container.
- Definition of the exposed ports.
- Definition of the entry point.
- Definition of the configuration method.
- Externalization of the data.
- Logs handling.
- Logs rotation and other append only files

## 3.4 Container Repository & Registry Management

A container registry is a directory where container images are stored so they may be pulled and pushed. However, the physical places where images are kept are called repositories. Each repository maintains a set of related images with the same name. A repository's images each

---

<sup>14</sup> <https://docs.docker.com/compose/gettingstarted/>

reflect a distinct deployment of the same container. A specific image is identified by either its tag or its own unique reference [Kisler21].

As mentioned in D2.7 [STAR-D2.7] for the STAR project, JFrog Container Registry has been selected to be used to setup a secure private Docker Registry. The JFrog Container Registry supports Docker registries and Generic repositories, allowing users to build, deploy and manage container images while providing powerful features with fine-grained permission control behind a sleek and easy-to-use UI. JFrog Container Registry imposes no limitations on the number of Docker Registries one may apply. The STAR JFrog Container Registry services can be accessed from:

- Dashboard URL: [https:// http://88.198.191.126/ui/](https://http://88.198.191.126/ui/)
- Private docker registry URL is: <https://88.198.191.126/artifactory/starregistry/>

Both require the firewall rules to be configured appropriately (i.e., allow access from a remote location) to be accessed.

### 3.5 Management/Monitoring with Portainer

Since the preferred deployment strategy is the docker containerization to facilitate the ecosystem management and monitoring there are various offerings. One of the proposed for the STAR deployment, is the Community Edition (CE) of Portainer<sup>15</sup>.

Portainer CE is a lightweight management toolset that allows you to easily build, manage and maintain Docker environments. Portainer offers a GUI (Graphical User Interface) which alleviates the complexity of using CLI (Command Line Input) commands. More specifically, via Portainer one can execute, in a user-friendly manner, various actions otherwise typed in the operating system's command line. Below is a list of actions that can be performed thru Portainer:

- Build and remove Docker images
- Push Docker containers through the various states of their lifecycle (Start, Stop, Restart, Remove, etc.)
- Create networks between Docker Engines running on different machines
- Administer volumes assigned to containers
- Inspect container logs and parameters
  - Using Log viewer.
- Run commands directly on the operating system enveloped by the container
- Monitor memory, CPU and network usage
- Expert configuration built into the software.
  - Including pre-validation checks for complex deployments
- Management of access control and LDAP authentication.
- Remote console with process performance viewer.
- Manage Docker Swarm service stacks and nodes (if existent).

---

<sup>15</sup> <https://www.portainer.io/products-services/portainer-community-edition/>

- Aggregation view of swarm clusters.

Directions on how the technology providers can install the Portainer environment in a local Docker instance can be found at the Portainer's Deployment<sup>16</sup> documentation. General documentation along with user and configuration guides can be found in Portainer's Documentation<sup>17</sup>.

---

<sup>16</sup> <https://portainer.readthedocs.io/en/stable/deployment.html>

<sup>17</sup> <https://portainer.readthedocs.io/en/stable/#>

## 4 The STAR Components

### 4.1 Security and Data Governance

#### 4.1.1 Components

##### 4.1.1.1 Runtime Monitoring System

###### 4.1.1.1.1 Short Description

Runtime Monitoring System (RMS) enables a real time service that collects security-related data from monitored IoT system components or applications and stores them for further processing. Analytics algorithms, like the AI Cyber Defence component, analyse the collected data to detect abnormal patterns. Additionally, the collected data can be directly used by the Security Policy Manager after applying special filters for reporting data exceeding “normal” thresholds. The system also features monitoring probes responsible for the data collection and publishing to the monitoring platform. The RMS provides appropriate configuration and management mechanisms over the monitoring probes as well as appropriate data models and data transformation engines that will maintain the probe information along with their status and will enable the probe creation, reconfiguration, and discovery.

###### 4.1.1.1.2 Relation with the Reference Architecture

RMS, depicted in Figure 2 above, is a Data collection framework which provides the specifications and relevant implementation to enable a real time data collection, transformation, filtering, and management service to facilitate data consumers (e.g., AI Cyber Defence Module and Security Policy Manager). The framework can be applied in IoT environments supporting solutions in various domains (e.g., Industrial, Cybersecurity, etc.). For example, the solution may be used to collect security related data (e.g., network, system, solution proprietary, etc.) from monitored IoT systems and store them to detect patterns of abnormal behaviour by applying simple (i.e., filtering and pre-processing) mechanisms.

###### 4.1.1.1.3 Dependencies

The RMS component is using Elastic Stack<sup>18</sup> which is comprised of Elasticsearch, Kibana, Beats, and Logstash (also known as the ELK Stack) and Kafka for the Data Bus. The different components are used as follows:

- **MetricBeats, HeartBeat:** collects monitored data (i.e., CPU utilization data) and availability status (i.e., network Camera availability) using Beats deployed to the Manufacturing Plant (demo VM).
- **HTTP Poller:** collects user data (i.e., image production rate) by polling the exposed services (repository).
- **Logstash:** Raw monitored Data are transformed and filtered to match the used Data Model (i.e., Observations) and identified rules (i.e., report values between specific thresholds).
- **Kafka & ElasticSearch:** the collected preprocessed data are published to the Data Bus (Kafka) in order to be accessed by the Security Policies Manager & ElasticSearch for permeate persistence, visualization and monitoring.

---

<sup>18</sup> <https://www.elastic.co/elastic-stack/>

- Security Policies Manager retrieves the preprocessed data by the Data Bus (Kafka) in order to be combined with other alerts/data (i.e., the AI Cyber Defence Strategies).
- **Kibana:** for persisted data visualization.

RMS is also using MongoDB<sup>19</sup> for the configuration repository and Java Spring Boot<sup>20</sup> framework for developing its microservices with Spring Framework.

#### 4.1.1.1.4 Availability

The runtime monitoring system is available under the Runtime-Monitoring-System GitHub group<sup>21</sup>.

#### 4.1.1.1.5 Installation/Deployment guidelines

The RMS installation is supported by Docker Compose. A “docker-compose.yml” file is provided which automates the installation of the RMS infrastructure.

### RMS

To launch the RMS containers, run

```
sudo docker-compose up
```

where the docker-compose.yml file is located.

To undeployed RMS containers run

```
sudo docker-compose down
```

where the docker-compose.yml file is located.

### Data Collection (Metric Beat)

To start Metric Beat for collecting the remote system utilization run the following commands:

```
cd metricbeat/  
sudo sh runmb.sh
```

Where the metric beat folder is located.

In Kibana:

- In 'Index Management' you should see the 'logstash-xxx-yyy' index. Metricbeat outputs to logstash, so this is the only index shown.
- Go to 'Index patterns' and create a new index pattern. Name it 'logstash\*'. Select the @timestamp field for temporal ordering.
- Go to the 'Discover' section to see the incoming events.
- Optional: In the 'Dashboard' section we can create a visual representation (graphs) of certain event fields.

In the 'logstash' folder:

- config/

<sup>19</sup> <https://www.mongodb.com/>

<sup>20</sup> <https://spring.io/projects/spring-boot>

<sup>21</sup> <https://github.com/star-eu/rms-data-collection>

- logstash.yml: The Logstash settings file. Contains options that control Logstash execution. Pipeline settings, location of config files, etc. Replaces command-line flags.
  - pipelines.yml: Instructions for running multiple pipelines in a single Logstash instance. Contains the paths to the configuration file(s).
- pipeline/
  - logstash.conf: The main configuration file of a pipeline. It contains the logstash plugins to be used, the settings for each plugin, as well as the output (i.e. elastic). Allows the manipulation of event fields, the use of conditionals for process events, etc.

#### 4.1.1.1.6 Documentation

More information on the RMS component and its subcomponents can be found in section 3 of D3.6 [STAR-D3.6]. Information on the RMS API can be found in section 3.2 of D3.6 [STAR-D3.6].

#### 4.1.1.1.7 Test Cases

The following test cases have been identified for an initial validation of the RMS interactions with other STAR components (intercomponent testing).

Test ID	RMS-01		
Title	Persistence of a Data Source Manifest (DSM)		
Pre-Requisite	<ul style="list-style-type: none"> <li>• DSM should have been already specified.</li> <li>• The Resource Registry is deployed.</li> <li>• The Resource Registry interface is reachable.</li> </ul>		
Expected Outcome	A DSM is persisted, and an ID is assigned to it		
Actions	Expected Result	Result	Comment
The user sends an HTTP <b>POST</b> request to <b>/data_source/dsm</b> .	The user receives an HTTP response with the persisted DSM and an id assigned to it.	Receive the DSM JSON object with an ID assigned to it and an HTTP status code OK (200)	

Test ID	RMS-02		
Title	Retrieval of a Data Source Manifest (DSM) record		
Pre-Requisite	<ul style="list-style-type: none"> <li>• The Resource Registry is deployed.</li> </ul>		

	<ul style="list-style-type: none"> <li>• DSM should have already been persisted and its id should be known.</li> <li>• The Resource Registry interface is reachable.</li> </ul>		
Expected Outcome	The requested DSM instance is returned		
Actions	Expected Result	Result	Comment
The user sends an HTTP <b>POST</b> request to <b>/data_source/ :id</b> .	The user receives an HTTP response with the persisted DSM.	Receive the DSM JSON object and an HTTP status code OK (200)	

### 4.1.1.2 Distributed Ledger Services for Data Reliability (DLSDR)

#### 4.1.1.2.1 Short Description

DLSDR 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. More details can be found in the Decentralized Reliability for Industrial Data and Distributed Analytics deliverable [STAR-D3.1 and STAR-D3.2].

#### 4.1.1.2.2 Relation with the Reference Architecture

DLSDR, depicted in Figure 2 above, offers the following functionalities to the STAR Security & Data Governance framework:

- For persisting/retrieving the AI algorithms configurations metadata which can describe an algorithm type along with its various instantiation configurations across time by using the Analytics Engine Configuration (AEC) service (see D3.1 [STAR-D3.1] section 3.3.1), and
- For persisting/retrieving AI algorithm results by utilizing the Analytics Results Publishing (ARP) service (see D3.1 [STAR-D3.1] section 3.3.2) using the Observation data structure. Samples of the blockchain persisted Analytics' results can be consumed by the Security Policy Management component to confirm their validity compared to the results that are retrieved from the Data Bus. Additionally, for data validation, critical results can be directly retrieved from the Data provenance & Traceability component.

#### 4.1.1.2.3 Dependencies

The Blockchain MVP will assume the existence of three organizations where we will maintain a virtual machine hosting their personal Hyperledger Fabric node, as well as its companion applications. Those machines will require the following Docker containers:

- A Peer Node<sup>22</sup>
- A CouchDB where the ledger state is being persisted<sup>23</sup>
- A Certificate Authority (CA)<sup>24</sup>
- A Command-Line Interface (CLI)<sup>25</sup>
- A Java application exposing an API making available the Node's functionalities to the centralized Blockchain Service Backend and, eventually, the outside world.

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

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

#### 4.1.1.2.4 Availability

The Distributed Ledger Services for Data Reliability component is available under the Data-Provenance-And-Traceability GitHub <sup>26</sup> .

#### 4.1.1.2.5 Installation/Deployment guidelines

Images for all those Docker components that are required for the Hyperledger Fabric deployment are provided by the Hyperledger Fabric development team via DockerHub<sup>27</sup>. The deployment process has been made semi-automatic by employing Docker Compose<sup>28</sup> scripts to pull those images, containerize them and deploy them as Docker Swarm stacks (more details on that are following in the next section). It needs to be highlighted, however, that deployment of Fabric components and the configuration of the network between them is a process more complicated than a simple "docker compose up" command, since rather complex configuration files and TLS certificates ought to have been prepared in advance. More information about the blockchain infrastructure deployment can be found in D3.1 [STAR-D3.1] under section 5.

#### 4.1.1.2.6 Documentation

The Distributed Ledger Services for Data Reliability component documentation for data models and API specifications can be found in the Decentralized Reliability for Industrial Data and Distributed Analytics deliverable [STAR-D3.1] under section 4. More specifically the following services are documented:

---

<sup>22</sup> Peer Node downloadable from DockerHub: <https://hub.docker.com/r/hyperledger/fabric-peer>

<sup>23</sup> CouchDB downloadable from DockerHub: <https://hub.docker.com/r/hyperledger/fabric-couchdb>

<sup>24</sup> Fabric CA downloadable from DockerHub: <https://hub.docker.com/r/hyperledger/fabric-ca>

<sup>25</sup> Fabric CLI downloadable from DockerHub: <https://hub.docker.com/r/hyperledger/fabric-tools>

<sup>26</sup> <https://github.com/star-eu/dlsdr>

<sup>27</sup> Hyperledger's user account on DockerHub: <https://hub.docker.com/u/hyperledger>

<sup>28</sup> Docker Compose official documentation: <https://docs.docker.com/compose/>



- Distributed Ledger Node Management
  - Registration and Discoverability of the Platform Nodes in section 4.1.1 of D3.1 [STAR-D3.1].
- Data Provenance & Traceability Services
  - Analytics Engine Configuration (AEC) service: Information about the exposed API, data models and usage can be found in section 4.2.3 of D3.1 [STAR-D3.1]
  - Analytics Results Publishing (ARP) service: Information about the exposed API can be found in section 4.2.4 of D3.1 [STAR-D3.1].

#### 4.1.1.2.7 Test Cases

The following test cases have been identified for an initial validation of the DLSDR interactions with other STAR components (intercomponent testing).

Test ID	SDG-01		
Title	Persistence of new Processor Manifest (PM) record		
Pre-Requisite	PM should have been already specified		
Expected Outcome	A PM is persisted, and an ID is assigned to it		
Actions	Expected Result	Result	Comment
The user sends an HTTP <b>POST</b> request to <b>/processor_config/pm</b> .	The user receives an HTTP response with the persisted PM and an id assigned to it.	DSM	

Test ID	SDG-02		
Title	Retrieval of a Processor Manifest (PM) record		
Pre-Requisite	PM should have already been persisted and its id should be known		
Expected Outcome	The requested PM instance is returned		
Actions	Expected Result	Result	Comment
The user sends an HTTP <b>GET</b> request to <b>/processor_config/:id</b> .	The user receives an HTTP response with the persisted PM.	PM	

#### 4.1.1.3 AI Cyber-Defence Strategies (ACDS)

##### 4.1.1.3.1 Short Description

The AI Cyber-Defence module aims to defend the STAR-enabled manufacturing platforms against poisoning and evasion attacks. Smart manufacturing ecosystems nowadays consist of several AI-powered components in order to improve their production practices. However, AI systems, are susceptible to attacks that target both the training (i.e., poisoning) and the

operational (i.e., evasion) phases of Deep Neural Networks (DNNs). In this direction, the AI Cyber-Defence component will boost the robustness of DNNs against adversarial inputs and attempts to contaminate the training datasets, and against active attacks that aim to evade the inference process of AI models.

The implementation of the tool has been completed (see D3.4) and is served as a dockerised application and is based on a flask server which works in synergy with AI/ML libraries and is combined with KAFKA in order to enable the input digestion and output sharing. The internal architecture of the tool is given in Figure 18. The reader can refer to D3.4 for more details on the AI Cyber-Defence tool and the defence strategies put forth.

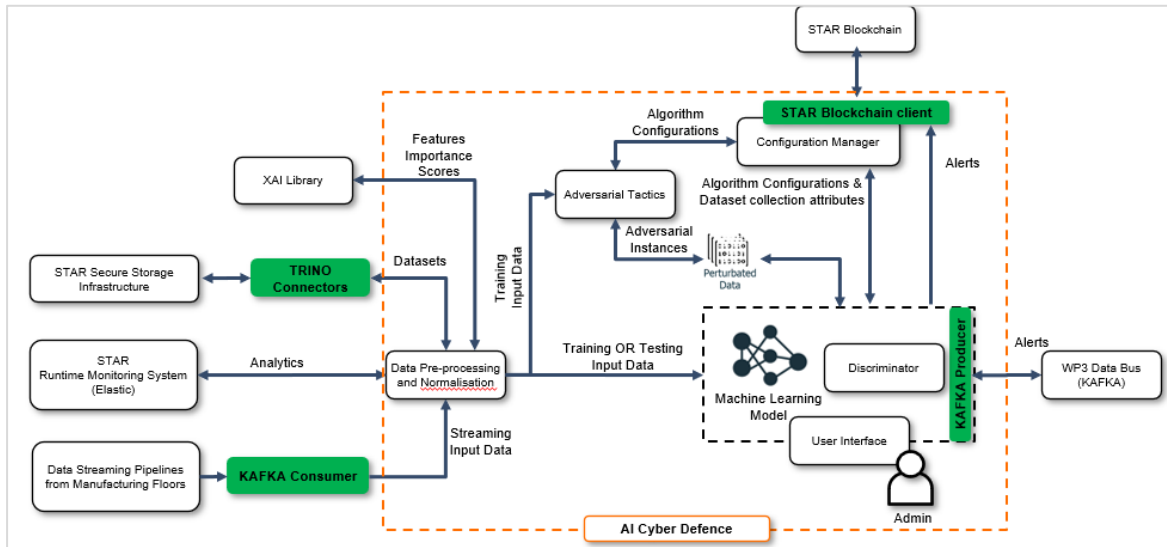


Figure 12: AI Cyber-Defence tool internal architecture

#### 4.1.1.3.2 Relation with the Reference Architecture

The AI Cyber-Defence tool is positioned at the AI Security and Data protection layer of the STAR architecture and works in synergy with the blockchain-based data provenance mechanisms, the Data management and Analytics engine, and the Explainable AI. The output of the AI Cyber-Defence mechanism will be used as input to the Security policy manager to perform a risk assessment and attack mitigation functionalities. Hence, the aim of the AI Cyber-Defence module coincides with the aim of the AI security and data protection layer which is to boost the safety, reliability and transparency of the functionalities of the upper operational layers of STAR.

#### 4.1.1.3.3 Dependencies

- Major libraries: The component is built in python, currently its major dependencies include: python >= 3.7, tensorflow >= 2.6.4, keras >= 2.6.0, torch >= 1.11.0, and Adversarial Robustness Toolbox (ART) v1.10, flask=2.1.2, Kafka-python=2.0.2, pandas=1.4.2, openpyxl=3.0.10, pillow=9.1.1
- Environment: The tool comes packaged as a docker container to be platform independent and make management of python, OS and Blockchain dependencies easier and more flexible.
- Components: The AI Cyber-Defence tool comes with KAFKA integrated in order to be able to handle inputs and outputs to and from the tool in an interoperable and unified manner.

#### 4.1.1.3.4 Availability

Currently, the code of the tool is under continuous development as new defence strategies are examined. The conditions on whether and/or under which license the code and the artifact will become available have not been clearly defined yet.

#### 4.1.1.3.5 Installation/Deployment guidelines

For the first ever execution through console one must navigate to the location of the .yml file and type the command:

```
docker-compose up -d
```

This first execution shall delay a little because it pulls pre-built images from online repositories (e.g. Docker Hub).

##### 4.1.1.3.5.1 Verification

To verify that everything is well, one may type:

```
docker ps
```

```
root@ubuntu:/home/ubuntu/star# docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
8747b491ade8   star_consumer                       "bash -c 'sleep 45; ..." 4 days ago    Up 20 seconds
                star_consumer_1
2ea39825a180   confluentinc/cp-kafka:7.1.1        "/etc/confluent/dock..." 4 days ago    Up 21 seconds    0.0.0.0:9092->90
->9092/tcp, 0.0.0.0:9999->9999/tcp, :::9999->9999/tcp   kafka1
424dc1495004   confluentinc/cp-zookeeper:7.1.1    "/etc/confluent/dock..." 4 days ago    Up 21 seconds    2888/tcp, 0.0.0.
p, :::2181->2181/tcp, 3888/tcp
ee0d19b6d5e9   star_producer                       "python3 serverProdu..." 4 days ago    Up 21 seconds    0.0.0.0:8080->80
->8080/tcp                star_producer_1
```

Figure 13: Console output of docker ps

As can be seen the application has two main services, the A) star\_consumer responsible for managing the inputs received and the B) star\_consumer which hosts the main business logic of the detection mechanism for the poisoning and evasion attacks. In addition, C) KAFKA is used as the component that will store the output of the detection process, as well as D) Zookeeper to complement the operation of KAFKA.

To open their logs, one may type: `docker logs -f <container name>`. To exit the log type **Ctrl+C**.

##### 4.1.1.3.5.2 Un-deployment

To stop and remove all containers one may type the command: `docker-compose down`

#### 4.1.1.3.6 Documentation

##### 4.1.1.3.6.1 Description of the input's outputs of the components

Input sources of AI Cyber-Defence tool:

- STAR Secure Storage infrastructure: This entity is a vital component in the STAR project architecture as it provides a unified datalake for all AI-enabled STAR components to consume data mainly for training reasons. That is, the AI Cyber-Defence module acquires datasets (e.g., images) for training purposes of the AI models. To establish this communication, the AI Cyber-Defence tool integrates the necessary TRINO connectors.

- STAR Runtime Monitoring system: The runtime monitoring system is used for the acquisition of statistical measurements stemming directly from core systems of the manufacturing floor. These measurements may indicate the presence of anomalous behaviours that could be used as inputs in the inference process. It has to be noted that this dependency with the RMS and XAI has been only approached from a scientific perspective. D3.4 provided a new chapter, based on the publications in (Afzal-Houshmand, 2021) and (Afzal-Houshmand, 2023) on the use of explainable artificial intelligence to enhance data trustworthiness in crowd-sensing systems. Thus, the actual connection in the context of the integrated framework has not been realised. However, the reader can refer to D3.4 for more details regarding the designed scientific approach on the use of sensory data to detect poisoning and evasion attacks.
- Data streaming pipelines: This entity refers to the data sources that are used during the actual deployment of the STAR platform to the pilot sites. Towards this phase of the project, the AI Cyber-Defence tool takes the necessary steps in order to be able to handle input data stemming from pilots in a streaming mode.

Output of AI Cyber-Defence:

The goal of the AI Cyber-Defence tool is to detect poisoning and evasion attacks against the STAR AI Systems. In this regard, the detection process, depending on the deployed detection model, a detection label is generated (evasion/poisoning) as well as a confidence level of the AI model. An indicative example of the output including additional metadata of the process is given below.

The APIs exposed by the AI Cyber-Defence tool has been documented in D3.4 in section 2.4.

*Table 2: Indicative output of AI Cyber-Defence tool.*

```
{
  "id": "1ee5f356-632e-11ec-90d6-0242ac120003",
  "dataSourceID": "3341e5b2-632e-11ec-90d6-0242ac120003",
  "assetID": "4daef4f2-487e-48e1-8f8f-d526d36aa5cd",
  "dataKindID": "65a7604e-9a94-4a74-9a34-3e44c6cebd49",
  "timestamp": "2022-01-10 13:01:29.709071",
  "location": {
    "geoLocation": {
      "latitude": "53.107731",
      "longitude": "6.088499"
    },
    "virtualLocation": "8.162.203.200"
  },
  "value": {
    "attackID": "f777d14b34c3cdf92468fbfa55aeeddd8298745",
    "attackContext": "Evasion attack",
    "Confidence": "90.12",
    "timestamp": "2022-01-10 13:01:22.709095"
  }
}
```

#### 4.1.1.3.7 Test Cases

Test ID	AICD-01		
Title	Detect the injection of an adversarial input into the AI pipelines of STAR.		
Pre-Requisite	<ul style="list-style-type: none"> <li>• Pretrained adversarial neural network that performs the detection</li> <li>• Preconfigured parametrization of the detection algorithm</li> </ul>		
Expected Outcome	Classification of the injected input to an adversarial category		
Actions	Expected Result	Result	Comment
An input is given into the STAR tools pipeline	<ul style="list-style-type: none"> <li>• Raise of alarm</li> <li>• provision of a confidence level on the prediction</li> </ul>	Executed	The structure of the output has defined above. The test has been completed in context of the experimental phases of the tool development and in the context of the integration actions with the pilots.

Test ID	AICD-02		
Title	Detect the presence of an adversarial example in the training datasets.		
Pre-Requisite	<ul style="list-style-type: none"> <li>• Dataset to be stored on the Storage infrastructure of STAR</li> <li>• Pre training model for the detection of poisoning attacks</li> </ul>		
Expected Outcome	Detection of poisonous instances in the datasets		
Actions	Expected Result	Result	Comment

<p>Adversarial instances are present in training datasets stored in the STAR Storage Infrastructure</p>	<ul style="list-style-type: none"> <li>• Raise of alarm</li> <li>• Detection of specific instance in a dataset.</li> </ul>	<p>Executed</p>	<p>The structure of the output has been defined above. The execution of the test case has been performed in the context of the experimental process in D3.4.</p>
---	--	-----------------	--

<p>Test ID</p>	<p>AICD-03</p>		
<p>Title</p>	<p>Correct acquisition/validation of tool configuration through STAR Blockchain</p>		
<p>Pre-Requisite</p>	<ul style="list-style-type: none"> <li>• Blockchain APIs are known</li> <li>• Initial configuration set has been stored on the blockchain</li> </ul>		
<p>Expected Outcome</p>	<p>Positive response if the configuration is valid and negative otherwise.</p>		
<p>Actions</p>	<p>Expected Result</p>	<p>Result</p>	<p>Comment</p>
<p>The AI Cyber-Defence tool triggers the Blockchain API to acquire the correct configuration.</p>	<p>Positive response if the configuration is valid and negative otherwise.</p>	<p>Executed.</p>	<p>The test has been executed in the context of the integration actions.</p>

#### 4.1.1.4 Risk Assessment and Mitigation Engine (RAME)

##### 4.1.1.4.1 Short Description

The Risk Assessment and Mitigation Engine is the technical component that complement the Security Policy Manager for the visualisation of the threats and the corresponding risks. The RAME is based on OLISTIC. More specifically, OLISTIC is UBITECH’s Risk Assessment tool which can support the security officer in getting an overview of the security status of the factory, and more specifically, of the production lines and business processes of interest. Overall, RAME enables the risk management and the identification and visualization of risks through comprehensive and reactive visualization, while it provides the means to the security officer to manage the life cycle of mitigation actions that helps to eliminate or control risk events that have been detected by the monitoring mechanisms of STAR.

The interested reader can refer to D3.6 where the final version of the RAME tool is provided. D3.6 concluded the technical work and the developments actions of WP3 tools, including RAME. In the D3.6, the reader can have a comprehensive report on the new APIs that enable the integration of RAME in the overall architecture of STAR.

#### 4.1.1.4.2 Relation with the Reference Architecture

OLISTIC contributes to the flow of the AI Security and Data protection layer, as the component that receives the security incidents that are being detected by the Security Policy Manager, as a result of policy violations, and offers to the security officer an interactive dashboard in order to understand the security posture of the manufacturing environment, considering the existing vulnerabilities and weak points of systems.

#### 4.1.1.4.3 Dependencies

- Major libraries: The component is a Quarkus-based application and currently its major dependency is Java SE JDK 11. In addition, the backend application is based on a proprietary application stack built by UBITECH. For fulfilling the purpose of internal storage and event management, OLISTIC uses also PostgreSQL (with pgAdmin), ELK stack (Elastic, Logstash, Kibana), Mongo DB and MinIO.
- Environment: The tool comes packaged as a docker container to be platform independent.
- Components: The RAME comes with PostgreSQL (with pgAdmin), ELK stack (Elastic, Logstash, Kibana), Mongo DB and MinIO integrated in order to be able to handle inputs and outputs and store the generated events. interoperable and unified manner.

#### 4.1.1.4.4 Availability

This is a proprietary component. UBITECH is responsible to manage the deployment of the tool on its premises.

#### 4.1.1.4.5 Installation/Deployment guidelines

For the first ever execution through console one must navigate to the location of the .yml file and type the command:

```
docker-compose up -d
```

*4.1.1.4.5.1 This first execution shall delay a little because it pulls pre-built images from online repositories (e.g. Docker Hub) Verification*

To verify that everything is well, one may type:

```
docker ps
```

The main services of the application are A) The back-end stack of UBITECH, B) PostgreSQL C) pgAdmin, D) ELK stack (Elastic, Logstash, Kibana), E) Mongo DB and F) MinIO.

To open their logs, one may type: `docker logs -f <container name>`. To exit the log type **Ctrl+C**.

#### 4.1.1.4.6 Documentation

As aforementioned, the RAME works in synergy with the security policy manager of STAR. More specifically, the latter detects security incidents against the monitored environment and generates attack events which are sent to the RAME. Then, RAME undertakes the task of performing the risk assessment and visualizing the risk and the offensive events to the security officer. After performing the risk assessment, the security administrator can check a

comprehensive report in the tool’s dashboard reflecting the security posture of the monitored production line.

#### 4.1.1.4.7 Test Cases

Test ID	RAME-01		
Title	Insertion of software and hardware assets into the RAME dashboard.		
Pre-Requisite	<ul style="list-style-type: none"> <li>The administrator has the knowledge on the assets taking part in the production lines of a factory</li> <li>The administrator has identified the relationship connecting the assets together</li> </ul>		
Expected Outcome	A graph-based representation of the environment is generated		
Actions	Expected Result	Result	Comment
The administrator adds the assets through a structured and well-defined data entry process.	<ul style="list-style-type: none"> <li>A list of assets is created.</li> <li>The backend databases store the provided information</li> </ul>	A graph-based representation of the environment is generated	

Test ID	RAME-02		
Title	Creation of an attack scenario denoting the presence of an abuse case and the assessed environment		
Pre-Requisite	<ul style="list-style-type: none"> <li>The SPM triggers the APIs of RAME to inform the latter about the detection of an abuse case.</li> <li>The database of RAME has already stored abuse cases that have been defined by the administrator.</li> </ul>		
Expected Outcome	A new attack scenario entry is generated in the risk assessment dashboard		
Actions	Expected Result	Result	Comment
The SPM identifies/detects an abuse case and triggers the RAME API.	<ul style="list-style-type: none"> <li>A new attack scenario is generated denoting</li> </ul>	A risk level is associated	



	<p>the presence of an abuse case.</p> <ul style="list-style-type: none"> <li>The risk assessment can be triggered in order to derive the risk level of the event.</li> </ul>	<p>with the assets that faced the abuse case.</p>	
--	--	---	--

#### 4.1.1.5 Security Policies Manager (SPM) - Security Policies Repository (SPR)

##### 4.1.1.5.1 Short Description

SSPM is a tool to be used by the factory personnel, in particular security/IT officers, to configure security policies according to specific business and security requirements. The main purpose of the SSPM is to detect poisoning and evasion attacks and report the related risk to the Risk Assessment module OLISTIC, to generate alerts.

SSPM integrates the Cyber-Defence mechanism of the Star Blockchain infrastructure, Data Provenance & Traceability, RMS, and AI Cyber-Defence module.

##### 4.1.1.5.2 Relation with the Reference Architecture

The SSPM acts as a middleman, as it aggregates the inputs received from the RMS and the AI Cyber-Defence Module, evaluates the received information based on security policies defined by the security officer, and interacts with OLISTIC to create and assess risk scenarios. The SSPM is implemented as a Python application, with a user interface and backend that exploits OPA<sup>29</sup>, an open-source, general-purpose policy engine that unifies policy enforcement across the stack, as an external service for policies evaluation.

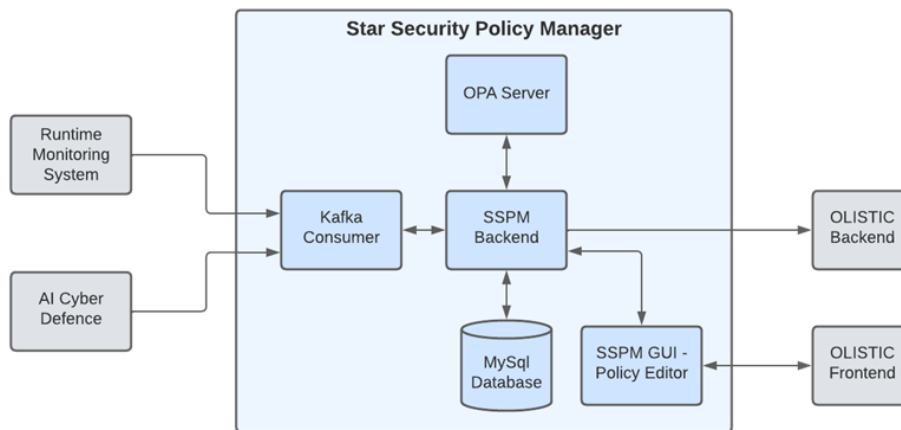


Figure 14: SSPM high level architecture

Figure 14 depicts a scheme of the SSPM architecture, which is composed of 5 modules:

- **Kafka Consumer:** it reads from the specific topics published by the RMS and AICD on the Data Bus and passes the received messages to the SSPM backend;
- **OPA Server:** the policy engine used to evaluate the received inputs against the

<sup>29</sup> <https://www.openpolicyagent.org/>

policies defined by the security officer;

- **SSPM database:** it stores security policies for persistence and SSPM configuration;
- **SSPM backend:** it manages inputs from the RMS and AICD components, interacts with OPA for policies evaluation and calls OLISTIC APIs based on the evaluation results;
- **SSPM GUI:** allows the security officer to create and update policies, and to configure the attack scenario to be created in OLISTIC when a policy has been violated.

More detailed information on SSPM can be found in deliverable D3.6.

SSPM supports the logic for multiple types of policies, evaluating the input received through the Kafka queue from the RMS and the AI Cyber-Defence Star's components. These policies can be applied to the following scenarios:

- Poisoning attack detection;
- System CPU workload detection;
- Heavy traffic or other probe's data that can signal a suspicious behaviour detection;
- Evasion attacks detection.

#### 4.1.1.5.3 Dependencies

- Major libraries – the SSPM is developed as a Python application, the frontend component uses the Flask Framework, whereas the backend exploits the kafka-python library. SSPM uses a MySQL database and peewee as an ORM.
- Environment - 2vCPU, 4GB RAM, 40GB disk space
- Components – SSPM backend, SSPM user interface to define and manage security policies, OPA, rule's storage.
- Output - interaction with OLISTIC through its APIs.

#### 4.1.1.5.4 Availability

The SSPM is hosted on GFT server, and the User Interface is available through OLISTIC.

#### 4.1.1.5.5 Installation/Deployment guidelines

The application is not contained within a single Docker image but is instead deployed as a Software as a Service (SAAS). In cases where installation on an external platform is required, it is possible to create a standalone Docker image. Currently, however, the solution is provided as a SAAS and is installed on a GFT server. It is accessible to OLISTIC and, therefore, to the STAR platform.

#### 4.1.1.5.6 Documentation

Here below a short report about the availability of components data:

- API doc (e.g., swagger availability) -> not applicable
- Description of the input's outputs of the components -> available and described in sub-paragraph 4.1.1.5.7
- High level API description -> not applicable

#### 4.1.1.5.7 Test Cases

Test ID	SDC-01		
Title	Attack evaluation		
Pre-Requisite	Kafka queue from RMS (through Data Bus) and AI Cyber-Defence Module		
Expected Outcome	OLISTIC APIs are triggered to alert the security officer about the possible threat.		
Actions	Expected Result	Result	Comment
The security officer can access SSPM interface and define security policies	The rules are saved and uploaded to OPA	The rules are saved and uploaded to OPA	
In the SSPM the security officer can create templates attack scenario	The templates attack scenarios are saved	The templates attack scenarios are saved	
The security officer can link a rule to an attack scenario that must be created in OLISTIC if the rule is violated	The association is saved.	The association is saved	
SSPM receives inputs from RMS and XAI Cyber-Defence module and validates the data against the defined security rules. SSPM communicates the existence of a threat to OLISTIC by triggering its APIs	The data is correctly evaluated, and threats are identified based on defined security policies. An attack scenario is created accordingly to the association defined by the security officer for the specific identified threat. A risk assessment is created and run in OISTIC.	The data is correctly evaluated, and threats are identified based on defined security policies. An attack scenario is created accordingly to the association defined by the security officer for the specific identified threat. A risk assessment is created and run in OISTIC	

#### 4.1.2 Use Cases

The purpose of this section is to be used as a reference on how the different components from WP3 explained above are going to be used within the context of WP6 and are prepared/tested in this task/deliverable. The table contains the components from WP3 that are planned to be utilised in the different pilots.

Use Case ID	Use Case title	Involved Component(s)	Short Description
PCL UC2	AI Cyber-Defence and decentralized reliability for	<ul style="list-style-type: none"> <li>Runtime Monitoring System</li> </ul>	The WP3 tools ensure the reliability of the production

	industrial data	<ul style="list-style-type: none"> <li>• Distributed Ledger Services for Data Reliability</li> <li>• AI Cyber-Defence Strategies (ACDS)</li> <li>• Risk Assessment and Mitigation Engine (RAME)</li> <li>• Security Policies Manager (SPM), Security Policies Repository (SPR)</li> </ul>	line and the integrity of the visual quality inspection system against AI adversarial attacks.
IBER Pilot #4	Agile Production Management System Data Integrity and Reliability	<ul style="list-style-type: none"> <li>• Runtime Monitoring System</li> <li>• Distributed Ledger Services for Data Reliability</li> <li>• AI Cyber-Defence Strategies (ACDS)</li> <li>• Risk Assessment and Mitigation Engine (RAME)</li> <li>• Security Policies Manager (SPM), Security Policies Repository (SPR)</li> </ul>	The WP3 tools ensure the reliability of the production line and the integrity of the visual quality inspection system against AI adversarial attacks.
DKFI UC3	Robot Reconfiguration based on the Dynamic Layout	<ul style="list-style-type: none"> <li>• Runtime Monitoring System</li> <li>• Distributed Ledger Services for Data Reliability</li> <li>• Risk Assessment and Mitigation Engine (RAME)</li> <li>• Security Policies Manager (SPM), Security Policies Repository (SPR)</li> </ul>	The WP3 tools ensure the reliability of Robotino's behavior by collecting, analyzing and validating crucial operational data for the sake of identifying abnormal behaviors or abuse cases that may alert the expected behavior of the Robotino.

### 4.1.3 Inter WP3 integration and communication

WP3 integration relies on the APIs which have been developed by each WP3 tool in the context of the development actions of WP3. These APIs are used for enabling the inter WP3 integration and communication. In this section we are not, referring to the internal APIs that enable the operation of each independent tool with its internal sub-components, but we refer on how the WP3 tools communicate among each other on the WP3-level basis.

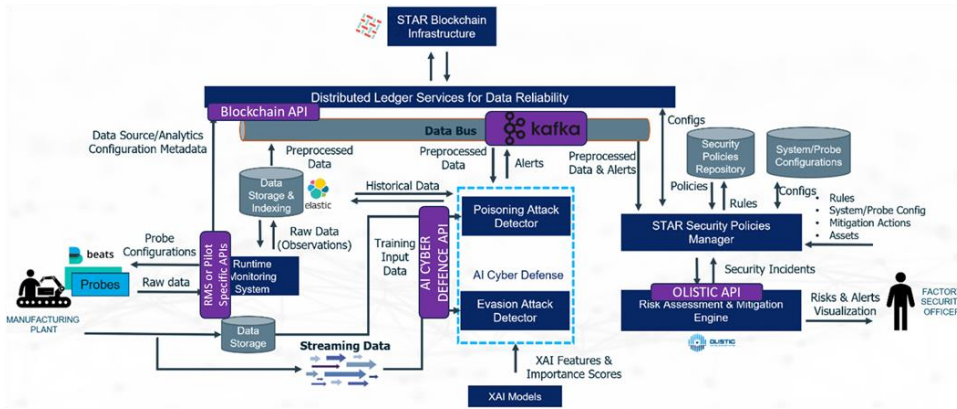


Figure 15: WP3 architecture and designed APIs

Thus, as can be seen in Figure 15, the purple boxes refer to the designed APIs of each component which have been designed in order to materialize all the information flows that the Security and Data Governance Layer of STAR requires in order to meet its functional objectives. A short description of the APIs is given in the following table with appropriate references to the technical deliverables of the components.

API	Description	Tool exposing the API	Tools using the API
AI Cyber-Defence API (See D3.4, Section 2.4)	This API is used for posting images to the AI Cyber-Defence Engine in order to trigger the Discriminator to infer whether the posted image is an adversarial example or not. This is the main endpoint triggered from the pilot’s environment in order to stream data to the AI Cyber-Defence tool in parallel to the production process.	AI Cyber-Defence.	Tools of the pilot partners that need to verify that the images being processed in the context of the manufacturing process are not a product of an adversarial attempt.
KAFKA (Data Bus) (See D3.6, Section 3.3)	Data Bus is a central communications channel through which all real time data is routed. Platform components may subscribe to the data bus to receive data of specific interest to them. The Data Bus is based on KAFKA, materializing a Publish/Subscribe	Used a central component for all WP3 tools. It is deployed as part of the RMS system.	<ul style="list-style-type: none"> <li>AI Cyber-Defence (Pub)</li> <li>Security Policy Manager (Sub)</li> <li>Runtime Monitoring System (Pub)</li> <li>Tools of pilot partners (Pub)</li> </ul>

	communication channel.		
RMS or Pilot Specific APIs (See D3.6, Section 3)	This API refers to the APIs used for connecting the pilot sites with the WP3 tools pipeline, and more specifically with the Runtime Monitoring System. Depending on the use case, the pilot partners may take advantage of the APIs exposed by the RMS tool in order to push data to the WP3 pipeline, or the RMS tool takes advantage of APIs exposed by the systems of the partners in order to consume vital information.	Depending on the use case, the APIs is exposed by the RMS tool or we rely on APIs designed by the use case partners	<ul style="list-style-type: none"> <li>• RMS</li> <li>• Systems of the use case partners</li> </ul>
BlockChain API (see D3.2 section 3.3.1)	The STAR blockchain infrastructure exposes an API for persisting/retrieving the AI algorithms configurations metadata which can describe an algorithm type along with its various instantiation configurations across time by using the Analytics Engine Configuration (AEC) service (see D3.2 section 3.3.1). Information about the exposed API, data models and usage can be found in section 4.2.3 of D3.2	STAR Blockchain Infrastructure	<ul style="list-style-type: none"> <li>• Any tool that needs to validate its configuration (e.g., the AI Cyber-Defence is using this API)</li> </ul>
OLISTIC API (see D3.6, Section 4.3.4)	UBITECH's technical team released a complete documentation of the APIs of the tool in order to enable full integration with other STAR tools that may	Risk Assessment and Mitigation Engine, based on UBITECH's OLISTIC risk assessment tool.	Security Policy Manager

	<p>need to interact with the Risk Assessment and Mitigation Engine of STAR. The APIs enable the exchange of information that refer to the identification of abuse cases, as a result of the detection processes of WP3 tools, aiming to the visualization of the events to the RAME dashboard</p>		
--	---	--	--

Leveraging these APIs, each module can effortlessly establish connections with the other components in the WP3 architecture. The main integration currently is completed. Other interactions can be easily implemented if needed thanks to the APIs exposed by the tools.

## 4.2 Safe, Transparent and Reliable Human-Robot Collaboration

### 4.2.1 Components

#### 4.2.1.1 Simulated Reality (SR)

##### 4.2.1.1.1 Short Description

The scope of the Simulated Reality component is to assist in the Automated Quality Inspection use-case of the project where it generates simulated images to balance defect datasets that suffer from skewed class data. Additionally, it is also intended to make the quality inspection algorithms more robust by leveraging highly generalizable GAN architectures to generate out-of-distribution images that will help visual classifiers recognize novel inputs.

##### 4.2.1.1.2 Relation with the Reference Architecture

Simulated reality aims to support components in the context of the Trusted Human-AI interactions pillar of the reference architecture. In its current form, it interacts with the ML Algorithms of the STAR Machine Learning and Analytics Platform and with Human-AI interaction components such as Active Learning, by providing them with synthetic data. The goal is for the simulated reality component to serve as a loosely coupled utility that can augment a ML algorithm’s training data on demand.

##### 4.2.1.1.3 Dependencies

- Major libraries: The component was built in python, currently its major dependencies include: python >= 3.7, tensorflow >= 2.6.4, keras >= 2.6.0, torch >= 1.11.0
- Environment: Since the SR component is intended to serve the ML Algorithms at training time it is delivered in the form of long-running scripts/batch jobs, currently open-sourced in the form of Jupyter notebooks. These notebooks can be run on demand on a cluster with Jupyter installed or, alternatively, they can be converted to .py files and run as python scripts. Computationally heavy tasks such as GAN training and fine-tuning require access to a GPU.



- (Sub-)Components: Two batch jobs (as Jupyter notebooks) for training data generators, one for rebalancing the training data and the second for generating out-of-distribution defects.

#### 4.2.1.1.4 Availability

The open-sourced Jupyter notebooks can be found under two GitHub repositories:

<https://github.com/tspyrosk/oversampling-defect-recognition>

<https://github.com/tspyrosk/osr-data-augmentation>

The first repository corresponds to the data augmentation algorithm aimed at balancing an input dataset and the second aimed at making the algorithm robust to novel inputs.

#### 4.2.1.1.5 Installation/Deployment guidelines

The notebooks can be run on a cluster with Jupyter installed or on a local installation. They can also be easily converted to .py files through Jupyter and run as python batch jobs/scripts. For better performance they need access to a GPU (NVidia K80 GPU was used for the evaluation).

#### 4.2.1.1.6 Documentation

Short Documentation for each notebook can be found in the corresponding repository README.md files.

#### 4.2.1.1.7 Test Cases

Test ID	SR-01		
Title	Generation of Synthetic Images for Visual Quality Inspection		
Pre-Requisite	Access to original training images and potentially required pre-trained models		
Expected Outcome	Generation of images that are visually similar to the originals (as judged through human perception, perceptual loss, FID etc.)		
Actions	Expected Result	Result	Comment
A STAR ML classifier requests synthetic data for balancing its input dataset (e.g., shaver shell prints)	The requested data is generated and saved in a specific location that should be accessible to the requesting component	The final augmented dataset and a classifier trained on it	

Test ID	SR-02		
Title	Generation of Novel/OOD defects for Visual Quality Inspection		



Pre-Requisite	Access to original training images and potentially required pre-trained models		
Expected Outcome	Generation of images that represent plausible defects but lie outside of the training classes' distributions		
Actions	Expected Result	Result	Comment
A STAR ML classifier requests synthetic data to improve its robustness (e.g., shaver shell prints)	The requested data is generated and saved in a specific location that should be accessible to the requesting component	The final augmented dataset and a classifier trained on it	

### 4.2.1.2 Active Learning (AL)

#### 4.2.1.2.1 Short Description

The active learning module implements multiple active learning strategies to assist the machine learning models in learning from the most meaningful data and avoid devoting time to label data instances that would eventually provide little or no performance enhancement to the machine learning model at hand.

#### 4.2.1.2.2 Relation with the Reference Architecture

Active Learning aims to support components in the context of the Trusted Human-AI interactions pillar of the reference architecture. In its current form, it may interact with the machine learning algorithms, and requires selecting data (natural or synthetic - the synthetic one generated from the Simulated Reality).

#### 4.2.1.2.3 Dependencies

**Major libraries:** The AL component is built in python, currently its major dependencies include: python >= 3.7, modAL<sup>30</sup>, scikit-learn >= 0.18, FastAPI.

**Environment:** The active learning component is envisioned to be packaged as a Docker container to be platform independent and make management of python and OS dependencies easier and more flexible. The Docker instance will be running some Linux distro. The application has not been dockerized yet.

**Components:** The Active Learning module has a dependency on the MongoDB to store data that is relevant to the service operation.

#### 4.2.1.2.4 Availability

The code has been made available in a project-specific repository (<https://github.com/star-eu/module-active-learning>). The Docker images are yet to be made available in the artefact store.

<sup>30</sup> <https://modal-python.readthedocs.io/en/latest/>

#### 4.2.1.2.5 Installation/Deployment guidelines

The installation and deployment guidelines have been provided at the README.md file at the root of the project-specific repository.

#### 4.2.1.2.6 Documentation

The REST API documentation is provided as a Swagger REST API endpoint. The REST API documentation can be accessed at the following endpoint: [http://\[ip\]:\[port\]/docs](http://[ip]:[port]/docs)

#### 4.2.1.2.7 Test Cases

Test ID	AL-01		
Title	Add feature vector data		
Pre-Requisite	<ul style="list-style-type: none"> <li>• A dataset exists with some labeled data</li> <li>• A supervised machine learning model exists, trained on the abovementioned dataset</li> <li>• A feature vector has been assembled.</li> </ul>		
Expected Outcome	A request response indicating whether the operation was successful or not.		
Actions	Expected Result	Result	Comment
The user sends an HTTP GET request to <code>MODEL_REPOSITORY_BASE_URL/module-active-learning/feature-vectors</code> .	The user receives an HTTP response with a <b>200</b> status code and a JSON exposing the ID assigned to such feature vector.	The feature vector was successfully registered.	

Test ID	AL-02		
Title	Add model prediction for a particular feature vector		
Pre-Requisite	<ul style="list-style-type: none"> <li>• A supervised machine learning model exists, trained on a particular dataset and has been registered in the model repository</li> <li>• A feature vector has been registered.</li> <li>• Predictions for the given feature vector has been created with the abovementioned model</li> </ul>		
Expected Outcome	A request response indicating whether the operation was successful or not.		
Actions	Expected Result	Result	Comment
The user sends an HTTP GET request to <code>MODEL_REPOSITORY_BASE_URL/module-active-learning/model-predictions</code>	The user receives an HTTP response with a <b>200</b> status code and a JSON informing the feature vector ID, the model ID and the predictions assigned to	The model predictions for this particular feature	

	the feature vector in the scope of this particular request.	vector have been successfully registered.	
--	---	---	--

Test ID	AL-03		
Title	Get top N unlabeled feature vectors given a particular machine learning model and active learning strategy.		
Pre-Requisite	<ul style="list-style-type: none"> <li>• A supervised machine learning model exists, trained on a particular dataset and has been registered in the model repository</li> <li>• A set of feature vector has been registered.</li> <li>• Predictions for the given feature vector has been created and registered with the abovementioned model and the abovementioned feature vectors</li> <li>• The amount of unlabeled feature vectors to be retrieved has been specified</li> </ul>		
Expected Outcome	A request response indicating whether the operation was successful or not.		
Actions	Expected Result	Result	Comment
The user sends an HTTP GET request to MODEL_REPOSITORY_BASE_URL/module-active-learning/model-predictions	The user receives an HTTP response with a 200 status code and a JSON informing feature vector ID, the dataset ID, the feature vector (list of values), and feature vector label (value should be -1, indicating no label has been assigned so far).	The unlabeled feature vectors have been successfully retrieved.	

Test ID	AL-04		
Title	Provide label for feature vector.		
Pre-Requisite	<ul style="list-style-type: none"> <li>• A supervised machine learning model exists, trained on a particular dataset and has been registered in the model repository</li> <li>• A set of feature vector has been registered.</li> </ul>		

	<ul style="list-style-type: none"> <li>• Predictions for the given feature vector has been created and registered with the abovementioned model and the abovementioned feature vectors</li> <li>• An unlabelled feature vector has been retrieved for annotation</li> </ul>		
Expected Outcome	A request response indicating whether the operation was successful or not.		
Actions	Expected Result	Result	Comment
The user sends an HTTP GET request to MODEL_REPOSITORY_BASE_URL/ /module-active-learning/feature-vectors/{id}/{label}	The user receives an HTTP response with a 200 status code and a JSON informing feature vector ID, the dataset ID, the feature vector (list of values), and feature vector label (value should be the one provided by the user).	The unlabelled feature vector has been successfully updated with the label.	

### 4.2.1.3 Production Processes Knowledge Base (PPKB)

#### 4.2.1.3.1 Short Description

PPKB is a knowledge base that is optional to use. It can be used to store meaningful data related to the manufacturing environment. In particular, two demo applications have been developed: (i) an application to register information related to a demand forecasting use case and recommendations provided based on knowledge related to the client location and information related to logistics (e.g., estimated transport delivery time and costs), and (ii) an application to retrieve information on how users consider class activation maps and anomaly maps (obtained from explainable artificial intelligence and unsupervised machine learning methods, respectively) should be recoloured.

#### 4.2.1.3.2 Relation with the Reference Architecture

The PPKB is an optional component that has no strong requirements on the interactions with other components. The relation to specific components is based on particular use cases and desired data to be collected. Please notice, that the knowledge base is a set of collected knowledge. Therefore, the applications used to generate it have to support a particular use case objective and may not relate to the reference architecture to achieve so.

#### 4.2.1.3.3 Dependencies

**Major libraries:** The PPKB component is built in python, currently its major dependencies include: python >= 3.7, and owlready2<sup>31</sup>.

**Environment:** The PPKB has been generated and the knowledge gathered in (ii) has been used to generate predictive machine learning models. The knowledge base itself has been recorded in memory, and the collected knowledge has been exported for experiments performed with (ii).

#### 4.2.1.3.4 Availability

The knowledge bases will not be published.

#### 4.2.1.3.5 Installation/Deployment guidelines

No installation guidelines are provided, given only proof-of-concept applications for the knowledge-base generation have been implemented and are not released publicly.

#### 4.2.1.3.6 Documentation

No documentation is provided, given only proof-of-concept applications for the knowledge-base generation have been implemented and are not released publicly.

#### 4.2.1.3.7 Test Cases

The following tests refer to the application (i) mentioned above.

Test ID	PPKB-APP1-01		
Title	Records user provided feedback for particular target entity (whose type can be forecast or explanation).		
Pre-Requisite	No prerequisites are required for this case.		
Expected Outcome	New instance of the feedback entity is created and connected with the target entity the feedback is provided about. New knowledge and relationships are created in the knowledge base.		
Actions	Expected Result	Result	Comment
The user sends an HTTP POST request to MODEL_REPOSITORY_BASE_URL/record_feedback that contains the information about the target entity and content of the feedback as specified in the HTTP query syntax.	The user receives an HTTP response with a 200 status code and a JSON body with the ID of the resource created and other relevant information.		

Test ID	PPKB-APP1-02		
Title	Retrieves a set of forecasts that meet the specified criteria.		
Pre-Requisite	No prerequisites are required for this case.		

<sup>31</sup> <https://owlready2.readthedocs.io/>

Expected Outcome	Instances of forecasts and their IDs are retrieved from the knowledge base.		
Actions	Expected Result	Result	Comment
The user sends an HTTP GET request to MODEL_REPOSITORY_BASE_URL/get_forecasts that contains the search criteria. The search query should be specified in the SPARQL ( <a href="https://www.w3.org/TR/sparql11-query/">https://www.w3.org/TR/sparql11-query/</a> ) language. Other parameters are specified following the HTTP query syntax.	The user receives an HTTP response with a 200 status code and a JSON body with the IDs of the forecasts together with other requested fields, such as number of items forecasted, timestamp, type of items, etc.		

Test ID	PPKB-APP1-03		
Title	Retrieves an explanation associated with the given forecast.		
Pre-Requisite	No prerequisites are required for this case.		
Expected Outcome	Instances of explanations for the given forecast are retrieved from the knowledge base.		
Actions	Expected Result	Result	Comment
The user sends an HTTP GET request to MODEL_REPOSITORY_BASE_URL/get_explanations that contains the search criteria. The search query should be specified in the SPARQL ( <a href="https://www.w3.org/TR/sparql11-query/">https://www.w3.org/TR/sparql11-query/</a> ) language. Other parameters are specified following the HTTP query syntax.	The user receives an HTTP response with a 200 status code and a JSON body with the IDs of the explanations together with other requested fields, such as type of explanation, XAI technique used for creating it, most significant features (if applicable), etc.		

#### 4.2.1.4 Natural Language Processing (NLP)

##### 4.2.1.4.1 Short Description

NLP is an optional module that facilitates communication between the operators and the machine, especially for web browser-based applications.

In our particular case, the functionality offered by this module is related to Speech-To-Text (STT), Text-to-Speech (TTS) technologies and conversational agents. In this way, UIs that require multimodal interaction can make use of these technologies to provide support for voice communication.

It should be noted that the module has focused on evaluating the different alternatives available for human-machine voice interaction (local, cloud, and mixed) and is not so much a

packageable component as an example code and a proof-of-concept that web-based UI components can implement in their respective modules.

It is also important to mention that the NLP module is not only focused on STT and TTS. The term NLP covers a wide spectrum of technologies, within it is the part dedicated to Sentiment Analysis or conversational agents and chatbots that we consider may be of interest to understand and give context to the interaction.

#### 4.2.1.4.2 Relation with the Reference Architecture

As mentioned above it is an optional component, the NLP is not necessary in all cases or pilots and even in some cases it is not recommended or possible to use it (e.g., noisy environments or environments where the operator cannot wear a headset for safety reasons).

In any case, it is possible to integrate it in any of the user interfaces that have interaction with this user through browser (Since is part of the NLP solution, the one dealing with quick TTS and STT has been developed in JavaScript).

The image below (Figure 16) shows the architecture of the component. The box indicated as web-app represents the web applications of the STAR architecture with Voice interaction needs or interest in Sentiment Analysis or Polarity Detection.

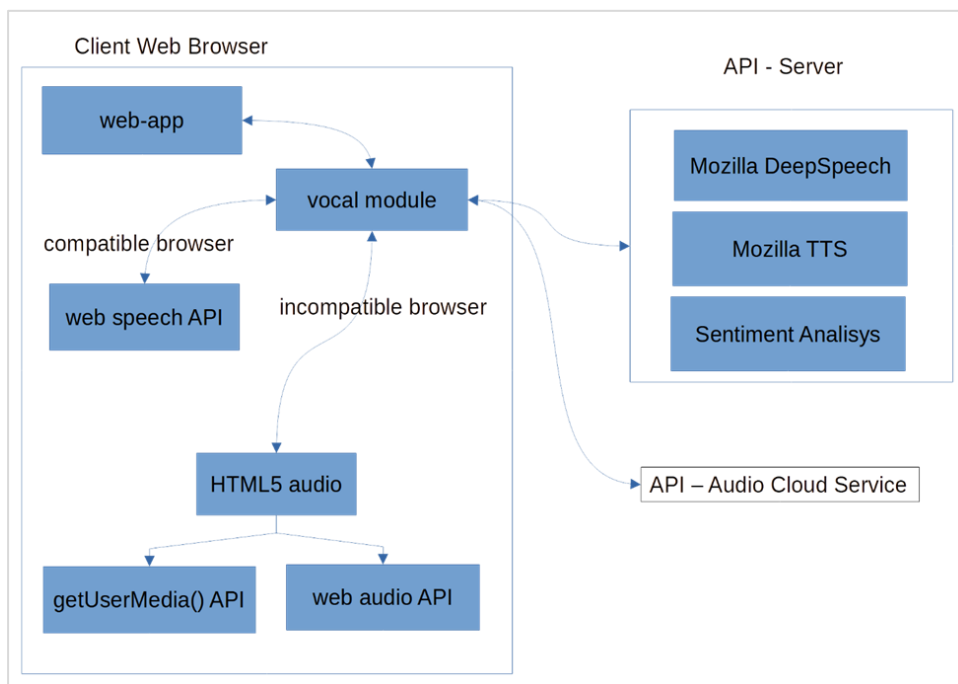


Figure 16: Architecture of the components for Nature Language Processing

The main route, if the application is browser-based, and if the browser used by the PC or machine with which the operator is interacting supports it, is through the Web Speech API. In some cases, it is possible to use the HTML5 Audio module to support incompatible browsers and clients.

The alternative route is for specific cases where more control over the audio is required (for example because extra processing is required) or cases where the client application is integrated with a third-party service. In the first case, TTS/STT can be implemented in a server using Mozilla DeepSpeech and similar libraries, in the second case the developer could

make use of the APIs of any cloud service. It is important to mention that although sentiment analysis (SA) can be performed in the browser itself, the results are not particularly good, and it is common to implement SA on a (cloud) server where previously trained sentiment analysis AI models can be executed.

#### 4.2.1.4.3 Dependencies

This module uses the functions offered by browsers compatible with the Web Speech API for speech synthesis and recognition.

In the case of Speech Synthesis, the support is included by default in most modern browsers. Speech Synthesis is supported on Google Chrome, Microsoft Edge, Mozilla Firefox (only voices installed on the client machine), Opera and Safari for PCs and on Chrome, Firefox and Safari for Android. Regarding speech recognition, the support also comes by default in most modern browsers and is a matter of implementing some support in JavaScript and fine tuning the grammar to enable the functionality.

#### 4.2.1.4.4 Availability, Installation/Deployment guidelines and Documentation

This module is not packaged since its functionality needs to be integrated into the application that is going to use voice interaction. Information related to the tests carried out with different STT and TTS solutions, grammar tests and two proofs of concept and reference implementations that demonstrate their use are offered.

#### 4.2.1.4.5 Test Cases

Test ID	NLP-01		
Title	Test the Speech-To-Text recognition.		
Pre-Requisite	Microphone enable and not blocked in the browser		
Expected Outcome	The speech or at least the part of the speech considered in the interaction grammar has been converted to text.		
Actions	Expected Result	Result	Comment
The user clicks the "voice interaction" button and records the message	The user receives the converted text.	The voice is converted to text, this text can be sent to the user or uploaded to the server for processing.	

Test ID	NLP-02		
Title	Test the Text-To-Speech		
Pre-Requisite	A textbox with some text and the selection of a voice (optional)		
Expected Outcome	Audio output of the text		



Actions	Expected Result	Result	Comment
The user writes a text in a text box.	The UI outputs the same text in audio format.	The UI outputs the text in audio (in the selected language)	

Test ID	NLP-03		
Title	Test extended NLP functionalities		
Pre-Requisite	-		
Expected Outcome	A NLP functionality is demonstrated apart from STT and TTS		
Actions	Expected Result	Result	Comment
The user writes a text in a text box.	The system analyses the text input to provide advanced functionalities	iConversation with the backend	Sentiment Analysis was the original idea for an extended NLP functionality, but due the integration with the Workers Training Platform, a chatbot was more interesting as a demonstrative case.

#### 4.2.1.4.6 Evaluation

This module is not finally included in shopfloor pilots, and has been integrated with the Workers Training Platform (WP5), Secure Storage (WP2/WP3) and Marketplace and project website (WP7/WP8) to allow public access to any interested user. This platform is accessible from the STAR project website and allows interaction with conversational agents through voice and text using NLP technologies, as mentioned in the module. We have called this integrated solution Multimodal Workers Training Platform, being the word "multimodal" the one that refers to NLP technologies.

To evaluate this tool, we conducted sessions with 15 different workers and focused on the usability of the solution. For this, we used the System Usability Scale (SUS) methodology (<https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>), which is relevant for interface evaluation. The results placed the solution above 80/100 which is

considered grade A, and therefore a more than positive result. The system was considered easy to use and the user felt confident in using it. The evaluation also gave us some clues for future development and research, such as the possibility of using more complex language models to enable open domain discussions.

#### 4.2.1.5 Feedback Module

##### 4.2.1.5.1 Short Description

The feedback module implements means to obtain users' explicit or implicit feedback regarding particular information displayed to them at the user interface.

##### 4.2.1.5.2 Relation with the Reference Architecture

The Feedback module provides means to gather feedback regarding predictions, decision-making options and explanations created by means of explainable artificial intelligence techniques. Such feedback is recorded along with the specific containers where the information is presented at the user interface. The recorded feedback (information displayed and particular feedback on usefulness or user choices) can be used to further enhance the applications and particular components (e.g., the quality of explanations, or the quality of predictions, among others).

The feedback module does not directly interact with other architecture components, but rather gathers useful information that can be later consulted by them.

##### 4.2.1.5.3 Dependencies

**Major libraries:** The Feedback module was built from scratch in Python. Currently, its major dependencies include: python >= 3.7, and Fast API.

**Environment:** The Feedback module is envisioned to be packaged as a docker container to be platform independent and make management of python and OS dependencies easier and more flexible. The Docker instance will be running some Linux distro.

**Components:** It is provided as a service interfacing with other services, with no intelligence of its own. It could be eventually evolved to apply some level of intelligence.

##### 4.2.1.5.4 Availability

The code has been made available in a project-specific repository (<https://github.com/star-eu/module-feedback>). The Docker images are yet to be made available in the artefact store.

##### 4.2.1.5.5 Installation/Deployment guidelines

The installation and deployment guidelines have been provided at the README.md file at the root of the project-specific repository.

##### 4.2.1.5.6 Documentation

The REST API documentation is provided as a Swagger REST API endpoint. The REST API documentation can be accessed at the following endpoint: `http://[ip]:[port]/docs`.

##### 4.2.1.5.7 Test Cases

Test ID	FM-01
Title	Add feedback container.

Pre-Requisite	<ul style="list-style-type: none"> <li>A user interface exists, for which the feedback container will be registered.</li> <li>Some name to be assigned to the feedback container.</li> </ul>		
Expected Outcome	<ul style="list-style-type: none"> <li>A feedback container is created and persisted in the database</li> <li>A response is given to the user, providing details (id and feedback container name) regarding the feedback container.</li> </ul>		
Actions	Expected Result	Result	Comment
The user sends an HTTP GET request to MODEL_REPOSITORY_BASE_URL/feedback-containers/ REST endpoint, with a JSON body (details provided at the Swagger endpoint).	The user receives an HTTP response with a <b>200</b> status code and a JSON body with details regarding the feedback container.	Matches the expected result.	

Test ID	FM-02		
Title	Retrieve feedback container by name.		
Pre-Requisite	A feedback container with the given name exists.		
Expected Outcome	The feedback container is retrieved and the details provided to the one performing the request.		
Actions	Expected Result	Result	Comment
The user sends an HTTP GET request to MODEL_REPOSITORY_BASE_URL/feedback-containers?name={some-container-name}.	The user receives an HTTP response with a <b>200 (OK)</b> status code and the feedback container details are displayed in the response content in JSON format.	Matches the expected result.	

Test ID	FM-03		
Title	Add feedback.		
Pre-Requisite	Add a particular feedback		
Expected Outcome	Feedback provided by the user is persisted into the database.		

Actions	Expected Result	Result	Comment
The user sends an HTTP POST request to MODEL_REPOSITORY_BASE_URL/feedbacks/ REST endpoint, with a JSON body (details provided at the Swagger endpoint)	The user receives an HTTP response with a <b>200 (OK)</b> status code and the feedback is successfully persisted into the database.	Matches the expected result.	

### 4.2.2 Use Cases

The purpose of this section is to be used as a reference on how the different components from WP4 explained above are going to be used within the context of WP6 and are prepared/tested in this task/deliverable. The table contains the components from WP4 that are planned to be utilised in the different pilots.

Use Case ID	Use Case title	Involved Component(s)	Short Description
PLC UC #1	Easy reconfiguration for automated part handling	<ul style="list-style-type: none"> <li>Active Learning (AL)</li> <li>XAI Techniques</li> </ul>	High flexibility is important and therefore this use case aims to test the flexibility of the AI algorithm to change from one product to another one within a limited timeframe.
PCL UC #2	Human supervised learning for visual quality inspections	<ul style="list-style-type: none"> <li>Simulated Reality (SR)</li> <li>XAI Techniques</li> </ul>	Once the AI algorithm is trained and taken in operation, we should still be confident in its performance. This use case aims to support workers with tools to enable them to do this.

### 4.2.3 Inter WP4 integration and communication

Our current emphasis lies in examining the interaction and synergy among the components developed and designed within this particular Work Package (WP4). Our goal is to present a coherent depiction of how inputs/outputs of these components intertwine and establish communication across different components within this WP4 of the project:

- **Simulated reality module** requires data instances to train a generative model. Once the model is trained, the generative model can be used to generate new data instances and contribute them to a particular database or dataset.
- **Active Learning module** may require interacting with the machine learning model to understand how the data instances are perceived by the machine learning model (e.g., whether the model is uncertain about their classification). Active learning strategies are used to retrieve data instances from a database or dataset, and then provided to train the machine learning model. After a new machine learning model is

available, a new iteration can be performed.

- **Explainable Artificial Intelligence** requires inputs related to a machine learning model. Such inputs depend on the kind of prediction being made and the desired explanation. Usually, Explainable Artificial Intelligence may require the data used to perform a prediction (e.g., image or feature vector), and the machine learning model used to perform a prediction. The forecast explanation is served to a particular stakeholder.
- **Feedback module** gathers feedback from different components displayed to the users. While not directly associated to other modules, the collected feedback can be used to improve the abovementioned modules (e.g., by collecting feedback regarding a particular prediction (e.g., was the prediction correct?) or explanation (e.g., was the explanation useful?)).

The relationship between the abovementioned components is depicted in the following Figure:

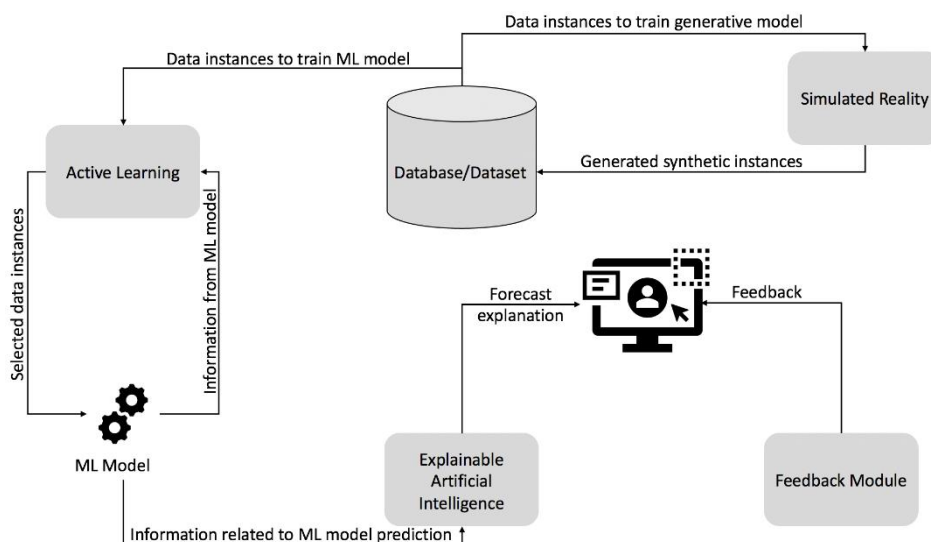


Figure 17: Interaction between WP4 components.

## 4.3 Human Centred Simulation and Digital Twins

### 4.3.1 Components

#### 4.3.1.1 AMR Safety

##### 4.3.1.1.1 Short Description

This component merges two subsystems

1. The safety zone detection
2. The AMR fleet management

The first component, described in D5.6, is devoted to improving the understanding of a global picture of the workflow. The safety zone detection system has the aim to complete the global awareness of the factory with a computer vision approach providing “spatial heatmaps” containing information on objects and workers’ positions in the workflow on the IoT Middleware in order to update the HDT picture.

The fleet management component, presented in D5.8, accesses to the heatmaps allowing the AMR module to dynamically define and update the best paths for the robots, avoiding obstacles and introducing factually safety when humans and robot share the same spaces.

#### 4.3.1.1.2 Relation with the Reference Architecture

The building blocks presented in the previous section, are integrated into the HDT system described in D5.1 and presented in the figure below (Figure 18). To support the Privacy by Design principle, the Safety Zones Detection System publishes on the HDT system only the results of the computation as heatmaps with the information on objects and workers' positions in the workflow. From the HDT Middleware, the AMRs access to the results of this module and consequently, they will adapt their behaviour.

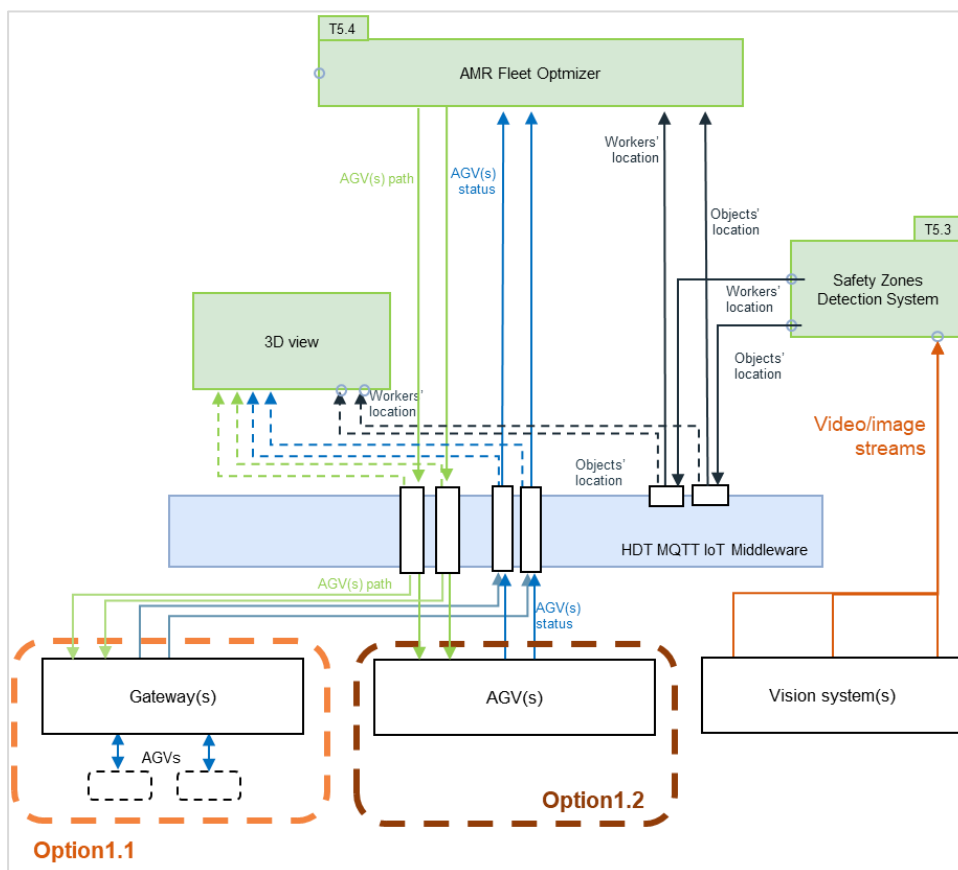


Figure 18: Safety Zone Detection & AMR Fleet Optimizer physical view

#### 4.3.1.1.3 Dependencies

The Safety Zones Detection is composed of several Docker images and a Docker compose file is used to manage these containers.

#### 4.3.1.1.4 Availability

The component (with the 2 sub-systems) is still in progress. It is not published yet. The software will be not publicly available. However, the "Safety Zones Detection" Docker images will be made available within the project to the partners of use case Human Behavior Prediction and Safety Zones Detection.

#### 4.3.1.1.5 Installation/Deployment guidelines

The Safety Zones Detections deployment will be fully based on Docker Compose. The component (with the 2 sub-systems) is still in progress. It is not available yet.

#### 4.3.1.1.6 Documentation

The documentation is in progress.

The Inputs:

- For the Safety Zones Detection: Stream from cameras RTSP
- For the AMR fleet Optimizer: FactoryOccupancy (Safety Zones outputs), the status of the AMRs, their current positions, current speed, current orientation.

	Data	From	To
battery status/autonomy	<p>State descriptor (UUID3) structure:</p> <p>StructBased</p> <ul style="list-style-type: none"> <li>– batteryStatus: NumberBased</li> <li>– unitOfMeasure: StringBased</li> <li>– factoryEntityID: StringBased</li> </ul> <p>Message example</p> <p>TOPIC: HDT/<u>UUID1</u>/state/UUID3</p> <p>MESSAGE: 1652965322501#{ "batteryStatus": 5, "unitOfMeasure": "aUnit", "factoryEntityID": "UUID1" }</p>	AMR (Robotino)	AMR Fleet Optimizer
current position	<p>State descriptor (UUID4) structure:</p> <p>StructBased</p> <ul style="list-style-type: none"> <li>– currentX: NumberBased</li> <li>– currentY: NumberBased</li> <li>– coordinateSystem: StringBased</li> <li>– factoryEntityID: StringBased</li> </ul> <p>Message example</p> <p>TOPIC: HDT/<u>UUID1</u>/state/UUID4</p> <p>MESSAGE:</p>	AMR (Robotino)	AMR Fleet Optimizer

	<pre>1652965322501#{"currentX": 12.3, "currentY": 27.5, "coordinateSystem":"ABC","factoryEntityID" :"UUID1"}</pre>		
current speed (if may be sent together with current position)	<p>State descriptor (UUID5) structure:</p> <p>StructBased</p> <ul style="list-style-type: none"> <li>– currentSpeed: NumberBased</li> <li>– unitOfMeasure: StringBased</li> <li>– factoryEntityID: StringBased</li> </ul> <p>Message example</p> <p>TOPIC: HDT/<a href="#">UUID1</a>/state/UUID5</p> <pre>1652965322501#{"currentSpeed": 2.9 "unitOfMeausure": "km/h", "factoryEntityID": "UUID0"}</pre>	AMR (Robotino)	AMR Fleet Optimizer
current orientation (optional)	<p>State descriptor (UUID6) structure:</p> <p>StructBased</p> <ul style="list-style-type: none"> <li>– currentOrientation: NumberBased</li> <li>– factoryEntityID: StringBased</li> </ul> <p>Message example</p> <p>TOPIC: HDT/<a href="#">UUID1</a>/state/UUID6</p> <pre>1652965322501#{"currentOrientation": 12.4, "factoryEntityID": "UUID1"}</pre>	AMR (Robotino)	AMR Fleet Optimizer
GlobalOccupancy (a.k.a. heatmap)	<p>State descriptor (UUID10) structure:</p> <p>StructBased</p> <ul style="list-style-type: none"> <li>– GlobalOccupancy: ListBased</li> <li>– HumanOccupancy: ListBased <ul style="list-style-type: none"> <li>– StructBased <ul style="list-style-type: none"> <li>– cellIndex : NumberBased</li> <li>– occupancy : NumberBased</li> </ul> </li> </ul> </li> </ul>	Video Analytics (a.k.a. SafetyZone Detection)	AMR Fleet Optimizer



	<p>– ObstacleOccupancy: StructBased</p> <p>– Type: StringBased</p> <p>– ListBased</p> <p>– cellIndex : NumberBased</p> <p>Message example</p> <p>TOPIC: HDT/UUID2/state/UUID10</p> <p>1652965322501#{"FactoryOccupancy": ["HumanOccupancy" : [{"cellIndex": 3, "occupancy": 1.0}, {"cellIndex": 5, "occupancy": 2.0}, {"cellIndex": 12, "occupancy": 1.0}], "ObstacleOccupancy" : {" Type": "WokingStation1", [1,2,3]}, "ObstacleOccupancy" : {" Type": "WokingStation4", [24,25,26]} ]], "factoryEntityID": "UUID2"}</p>		
--	---	--	--

Note that the FactoryOccupancy is also called spatial heatmap.

The Outputs:

- From the Safety Zones Detection: This sub-system provides the spatial heatmaps with the object and human occupancies.
- From the AMR fleet Optimizer: This sub-system will send moving commands to each AMR for the fleet. For a list of possible commands, see the table below.

<p>nextWaypoint</p>	<p>State descriptor (UUID7) structure:</p> <p>StructBased</p> <p>– nextX: NumberBased</p> <p>– nextY: NumberBased</p> <p>– factoryEntityID: StringBased</p> <p>Message example</p> <p>TOPIC: HDT/UUID2/state/UUID7</p> <p>1652965322501#{"nextX": 12.2, "nextY": 12.3, "factoryEntityID": "UUID2"}</p>	<p>AMR Fleet Optimizer</p>	<p>AMR (Robotino )</p>
---------------------	--	------------------------------------	--------------------------------

<p>nextPath</p>	<p>State descriptor (UUID8) structure:</p> <p>StructBased</p> <ul style="list-style-type: none"> <li>– nextPath: ListBased</li> <li>– StructBased             <ul style="list-style-type: none"> <li>– x: NumberBased</li> <li>– y: NumberBased</li> </ul> </li> <li>– factoryEntityID: StringBased</li> </ul> <p>Message example</p> <p>TOPIC: HDT/UUID2/state/UUID8</p> <p>1652965322501#{"nextPath": [{ "x": 12.3, "y": 12.4}, { "x": 13.3, "y": 13.4}, { "x": 14.3, "y": 14.4}], "factoryEntityID": "UUID2"}</p>	<p>AMR Fleet Optimizer</p>	<p>AMR (Robotino )</p>
<p>nextAction</p>	<p>State descriptor (UUID9) structure:</p> <p>StructBased</p> <ul style="list-style-type: none"> <li>– action: StringBased</li> <li>– factoryEntityID: StringBased</li> </ul> <p>Message example</p> <p>TOPIC: HDT/UUID2/state/UUID9</p> <p>1652965322501#{"action": "move"}, "factoryEntityID": "UUID2"}</p> <p>Actions may be predefined, e.g., [move, stop, charge].</p>	<p>AMR Fleet Optimizer</p>	<p>AMR (Robotino )</p>

#### 4.3.1.1.7 Test Cases

<p>Test ID</p>	<p>AMR-01</p>
<p>Title</p>	<p>Detect the position of the human using real-time RGB cameras</p>
<p>Pre-Requisite</p>	<ul style="list-style-type: none"> <li>• Pretrained neural network parameters for human detection</li> <li>• RTSP Cameras</li> <li>• Cameras’ calibrations and global map of the factory</li> </ul>

Expected Outcome	People are well positioned		
Actions	Expected Result	Result	Comment
Operator inspects the module(s)	<ul style="list-style-type: none"> <li>Humans are detected</li> <li>People are well positioned in 3D real scene</li> </ul>	a Docker container ready to be deployed Several tests on DFKI factory videos	

Test ID	AMR-02		
Title	Detect the position of the moving object (robot for example) using real-time RGB cameras		
Pre-Requisite	<ul style="list-style-type: none"> <li>Background model or a few minutes of the empty scene</li> <li>RTSP Cameras</li> <li>Cameras' calibrations and global map of the factory</li> </ul>		
Expected Outcome	Moving object is well positioned		
Actions	Expected Result	Result	Comment
Operator inspects the module(s)	<ul style="list-style-type: none"> <li>Moving objects are detected</li> <li>Moving objects are well positioned in 3D real scene</li> </ul>	Docker container ready to be deployed Several tests on DFKI factory videos	

Test ID	AMR-03		
Title	Detect the position of the static object of interest (new object for example) using real-time RGB cameras		
Pre-Requisite	<ul style="list-style-type: none"> <li>Pretrained neural network parameters for object detection and classification</li> <li>Background model or a few minutes of the empty scene</li> <li>RTSP Cameras</li> <li>Cameras' calibrations and global map of the factory</li> </ul>		
Expected Outcome	Moving object is well positioned		
Actions	Expected Result	Result	Comment

Operator inspects the module(s)	Objects of interest are detected and are well positioned in 3D real scene	The algorithm detects and classifies the object of interest (i.e. robotino)	
---------------------------------	---	---	--

Test ID	AMR-04		
Title	Static obstacles avoidance at the planning level		
Pre-Requisite	<ul style="list-style-type: none"> <li>• Empty map of the environment</li> <li>• Obstacle presence (coming from Video analytics)</li> </ul>		
Expected Outcome	Path avoiding obstacles		
Actions	Expected Result	Result	Comment
Launching pathfinding requests (from a specific origin to a specific destination)	Path can be displayed (in a virtual environment) which allows to check wrong behavior (collision)	results ok in simulation	Simulation capabilities allow to test in a wider scope than just the real environment

Test ID	AMR-05		
Title	Human avoidance anticipation at the planning level		
Pre-Requisite	<ul style="list-style-type: none"> <li>• Empty map of the environment</li> <li>• Obstacle presence (coming from Video analytics)</li> <li>• Factory occupancy (a.k.a. heatmaps)</li> </ul>		
Expected Outcome	Path anticipating crowded areas		
Actions	Expected Result	Result	Comment
Launching pathfinding requests (from some specific origin to a specific destination)	Path can be displayed (in a virtual environment) which allow to check wrong behavior (passing through crowded areas)	results ok in simulation	Simulation capabilities allow to test in a wider scope than just the real environment

### 4.3.1.2 Human Centred Digital Twin

#### 4.3.1.2.1 Short Description

The Human Centred Digital Twin (HDT) is an extensible and flexible IIoT-based platform supporting the creation of customised data representations of production systems and their entities, including humans. It features a modular infrastructure with interchangeable components, which ease the digital twin instantiation and ramp-up.

#### 4.3.1.2.2 Relation with the Reference Architecture

The HDT is a core component within the Reference Architecture (see Figure 2 from D2.6). Data from the shopfloor are sent to the HDT, which models all the entities living in the factory (including equipment, devices, and humans), as well as functional modules providing evidence about such entities (e.g., Fatigue Monitoring System).

#### 4.3.1.2.3 Dependencies

The HDT is composed of different backend services to manage digital models and their orchestration. The main service (*orchestrator*) is a Java Spring application server that orchestrates the other components and allows users to customise their digital factory representation. To support the data flows involving devices in the shopfloor, the HDT includes an MQTT-based broker. Finally, to persist data flowing on the broker, the HDT employs an additional Java application (*hdm*) that persists static data into a NoSQL database (i.e., MongoDB), and dynamic data into a timeseries database (i.e., InfluxDB). The persisted historical data are exposed by means of another Java Spring Application (*hdm-web*). Both *orchestrator* and *hdm-web* services are accessible via HTTP REST APIs.

#### 4.3.1.2.4 Availability

The software of HDT components is not publicly available. However, the Docker images of the components are made available within the project to all the interest partners. Docker images are stored in a proprietary Docker registry, accessible via token-based authentication.

#### 4.3.1.2.5 Installation/Deployment guidelines

The deployment of HDT components is fully based on Docker Compose and deploys all the HDT components as depicted in Figure 19. The installation/deployment guidelines are available in the [project repository](#).

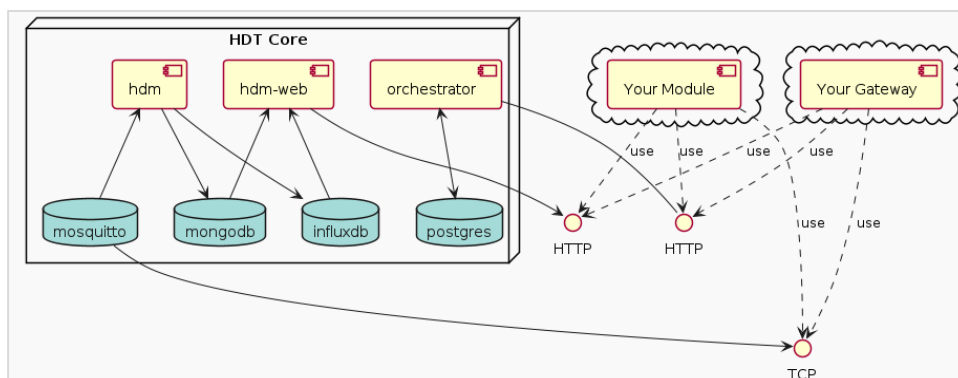


Figure 19: Deployment view of HDT components

#### 4.3.1.2.6 Documentation

The components documentation is available in the [project repository](#), with some code examples. Services exposed via REST APIs are documented with Swagger (OpenAPI specification).

#### 4.3.1.2.7 Test Cases

Test ID	HDT-01		
Title	Register a new functional module to the HDT.		
Pre-Requisite	The <i>orchestrator</i> service is up and running		
Expected Outcome	The <i>orchestrator</i> service registers the new functional module.		
Actions	Expected Result	Result	Comment
The client sends an HTTP <b>POST</b> request to <b>ORCHESTRATOR_BASE_URL/api/v1/functional-module</b> , passing a valid <code>FunctionalModuleDto</code> in the body.	The user receives an HTTP response with a <b>200</b> status code and a body containing a <code>ResponseFunctionalModuleDto</code> .	Test successful	DTOs are described in Swagger at <code>ORCHESTRATOR_BASE_URL/swagger-ui/index.html</code>

Test ID	HDT-02		
Title	Activate a functional module, by ID		
Pre-Requisite	The functional module has been already registered in the HDT.		
Expected Outcome	The functional module is activated. The event is published on the MQTT broker.		
Actions	Expected Result	Result	Comment
The user sends an HTTP POST request to <code>MODEL_REPOSITORY_BASE_URL/api/v1/functional-module/activate/functional-module/{functionalModuleId}</code> .	The user receives an HTTP response with a <b>200</b> status code. The response body contains a <code>ResponseFunctionalModuleActivationResponseDto</code> , which briefly summarises the current status of the HDT (e.g., list of active entities in the factory).	Test successful	DTOs are described in Swagger at <code>ORCHESTRATOR_BASE_URL/swagger-ui/index.html</code>

Test ID	HDT – 03 (FaMS - 01)		
Title	Communication between the HDT Core Infrastructure and the FaMS		
Pre-Requisite	<ul style="list-style-type: none"> <li>• HDT and FaMS configured, deployed and functioning.</li> <li>• Gateway and wearables are active.</li> </ul>		
Expected Outcome	Data streamed from the wearables are stored in the MQTT broker and becomes available to FaMS via HDM-Web		
Actions	Expected Result	Result	Comment
The user logs in and starts a new session on the gateway.	The streaming of physiological data starts; data are written to the MQTT broker, and the HDM automatically persists them in InfluxDB. FaMS is then able to access physiological data by querying the HDM-Web API. FaMS estimates the fatigue exertion level and publishes the result on the MQTT broker.	Test successful	

Test ID	HDT - 04		
Title	Communication between the HDT Core Infrastructure and the Robotino		
Pre-Requisite	<ul style="list-style-type: none"> <li>• HDT Core Infr. deployed and functioning</li> <li>• Robotino’s agent deployed and functioning.</li> </ul>		
Expected Outcome	Data (e.g., battery level) are streamed from the Robotino to the HDT Core Infrastructure		
Actions	Expected Result	Result	Comment
The Robotino’s agent is configured in the HDT Core Infr.	Streaming of Robotino’s data are available and published on the HDT Core Infr. Robotino’s is able to read data on the HDT Core Infr.	Not yet tested.	

Test ID	HDT - 05		
---------	----------	--	--

Title	Communication between the HDT Core Infrastructure, Safety Zones Detection System and the AMR Fleet Optimizer		
Pre-Requisite	HDT Core Infr., Safety Zones Detection System and AMR Fleet Optimizer configured, deployed and functioning.		
Expected Outcome	Safety Zones Detection System and AMR Fleet Optimizer are capable to read and publish data on the HDT Core Infr.		
Actions	Expected Result	Result	Comment
The user activates a session. The two modules read data on the HDT Core and publish the results of their computations.	AMR paths and workers positions are published on the HDT Core Infrastructure. AMR Fleet Optimizer reads Robotino's data and workers positions published on the HDT Core Infr.	Not yet tested.	

### 4.3.1.3 Fatigue Monitoring System

#### 4.3.1.3.1 Short Description

The Fatigue Monitoring System (FaMS) uses a machine learning model that estimates the exertion level of subjects based on static data (e.g., age, weight, etc.) and dynamic data (e.g., HR, EDA, skin temperature). Wearable devices are used to collect the subjects' physiological data, while static data are collected through a questionnaire. With the implemented algorithm it is then possible to derive the stress level of the user. This 'AI module' can be used alone to understand the exertion level of the workers who are performing a task, or it can also be used by 'decision maker modules' to make human-aware decisions.

#### 4.3.1.3.2 Dependencies

The Fatigue Monitoring System is a plain Python application exploiting the sci-kit-learn library. It depends on the Human Centred Digital Twin from which it retrieves static and dynamic data, and where it publishes new computed exertion levels.

#### 4.3.1.3.3 Availability

The software is not publicly available. However, the Docker image of FaMS is made available within the project to all the interest partners. The Docker image is stored in a proprietary Docker registry, accessible via token-based authentication.

#### 4.3.1.3.4 Installation/Deployment guidelines

The deployment is fully based on Docker Compose. The user guide is available in the [project repository](#).

#### 4.3.1.3.5 Documentation

The documentation is available in the [project repository](#).



#### 4.3.1.3.6 Test Cases

Test ID	FaMS 01 (HDT-03)		
Title	Communication between the HDT Core Infrastructure and the FaMS.		
Pre-Requisite	HDT and FaMS configured, deployed and functioning; Gateway and wearables active.		
Expected Outcome	Data are streamed from the wearables to the HDT Core Infrastructure and available for the FaMS		
Actions	Expected Result	Result	Comment
The user logs in and activates the session on the gateway.	Streaming of physiological data, FaMS is able to access to physiological data streams and quasi-static data in the HDT Core Infrastructure; FaMS estimates the fatigue exertion level and publishes the result on the HDT Core Infrastructure.	Test successful	

#### 4.3.1.4 Workers Activity Recognition

##### 4.3.1.4.1 Short Description

Worker’s activity recognition is to prevent collision between the worker and mobile robot in the manufacturing line. The mobile robot moves from a module to another module. Mobile robot should predict the worker’s next action to prevent the accident with the worker. To recognize the worker’s activity, we attach wearable sensors on the worker’s body, such as on both wrists. The collected data are processed by designed neural networks for activity recognition.

##### 4.3.1.4.2 Relation with the Reference Architecture

The activity recognition module receives the input signals from the wearable sensors and sends the output of the recognized worker’s activity. In its current form, it connects to the STAR machine learning and analytics platform.

##### 4.3.1.4.3 Dependencies

- Major libraries: PyTorch
- Environment: Python
- Components: IMU sensors

##### 4.3.1.4.4 Availability

This module is still in progress. It is not published yet.

#### 4.3.1.4.5 Installation/Deployment guidelines

Not available yet.

#### 4.3.1.4.6 Documentation

- Input: CSV files from the IMU sensor
  - 1<sup>st</sup> column: timestamp
  - 2-4<sup>th</sup> column: ACC 3-channels from left wrist sensor
  - 5-7<sup>th</sup> column: Gyroscope 3ch data from left wrist sensor
  - 8-10<sup>th</sup> column: Magnetometer 3ch data from left wrist sensor
  - 11<sup>th</sup> column: Capacitive sensor data from left wrist sensor
  - 12-14<sup>th</sup> column: ACC 3ch data from right wrist sensor
  - 15-17<sup>th</sup> column: Gyroscope 3ch data from right wrist sensor
  - 18-20<sup>th</sup> column: Magnetometer 3ch data from right wrist sensor
  - 21<sup>st</sup> column: Capacitive sensor data from right wrist sensor
- Output: JSON file. An example is given below:

```
currentActivity: openDoor,
nextActivities: {
  1: lockDoor,
  1Perc: 90%,
  2: goToNextModule,
  2Perc: 10%
}
```

#### 4.3.1.4.7 Test Cases

The following test is foreseen to validate the component:

Test ID	HAR-01		
Title	Detect the activity of the human using real-time sensor data		
Pre-Requisite	<ul style="list-style-type: none"> <li>• Pretrained neural network parameters for human activity recognition</li> <li>• Attached wearable sensors on worker’s body</li> </ul>		
Expected Outcome	Human activity is classified		
Actions	Expected Result	Result	Comment
Human inspects the module(s)	<ul style="list-style-type: none"> <li>• The current activity is detected</li> <li>• Precision of the activity detection is returned</li> <li>• Possible next actions with their accuracies are returned</li> </ul>	Not yet executed	The return format is still to be defined.

### 4.3.2 Use Cases

The purpose of this section is to be used as a reference on how the different components from WP5 explained above are going to be used within the context of WP6 and are prepared/tested in this task/deliverable. The table contains the components from WP5 that are planned to be utilised in the different pilots.

Use Case ID	Use Case title	Involved Component(s)	Short Description
PCL Pilot #3	Worker fatigue and mental stress in quality inspection	<ul style="list-style-type: none"> <li>Human Centred Digital Twin</li> <li>Fatigue Monitoring System</li> </ul>	The use-case aims to detect and monitor workers' fatigue when performing manual labelling of images. In particular, we tackle the visual inspection use case (Philips UC2). Detecting workers' fatigue, attention, and/or mental stress during the labelling process can be helpful at least in two ways: understand whether the labelled data can be trusted or should be reviewed by multiple workers, to ensure the accuracy of the final label provided; suggest a break or change of activity to the user, to avoid disengagement.
DFKI Pilot #1	Human Intention Recognition.	<ul style="list-style-type: none"> <li>Human Centred Digital Twin</li> <li>Workers Activity Recognition</li> </ul>	This use-case aims to detect worker's activities to prevent collisions between mobile robot and the worker. The worker wears wearable gloves and watch with IMU and capacitive sensors.
DFKI Pilot #2	Robot Reconfiguration Based on the Dynamic Layout.	<ul style="list-style-type: none"> <li>Human Centred Digital Twin</li> <li>AMR Safety zone</li> <li>Safety zone detection</li> </ul>	Plan to dynamically update the navigation map of the scene, by considering human and/or other (non-) moving objects in the environment by two ceiling cameras installed in the testbed. This use case also enables easier reconfiguration of the robot in case the layout of the environment (including the production stations) changes.
DFKI Pilot #3	Dynamic Path Planning Using	<ul style="list-style-type: none"> <li>AMR Safety</li> </ul>	The two use cases for DFKI Pilot #1 and #2 are going to

	Both First and Second Use Cases	<ul style="list-style-type: none"> <li>• Safety zone detection</li> <li>• Human Centred Digital Twin</li> <li>• Workers Activity Recognition</li> </ul>	be combined to have a safe environment for the workers and the hardware equipment considering the AMR in the scene. The prediction of next human activity is utilized in the path planning and remapping the unoccupied area
IBER Pilot #3	Employee Training for Reduction of Human Errors	<ul style="list-style-type: none"> <li>• Fatigue Monitoring System</li> </ul>	The third case study of the IBER pilot project foresees the identification of the mental fatigue of operators on the assembly line, through the monitoring of recorded errors (non-conforming product).

### 4.3.3 Inter WP5 Integration and Communication

WP5 integration relies on the HDT Core Infrastructure. Each component developed within the workpackage is considered as a Functional Module. The Functional Modules are pluggable components, including AI modules, that can be easily added or removed from the HDT. These Functional Modules can be subscribed to the IIoT Middleware topics of their interest, in order to use data streams from sensors and other Functional Modules’ outputs to perform their computations. In addition, Functional Modules can also retrieve quasi-static data from the HDT by means of REST API provided by the Orchestrator. The Functional Modules also have the possibility to publish their outputs to the IIoT Middleware.

Crucially, the efficacy of this system hinges upon the HDT Core Infrastructure model, epitomized by its entities, namely FunctionalModuleInput and FunctionalModuleOutput. Leveraging these entities, each module can effortlessly establish connections with the HDT Core Infrastructure, thereby facilitating access to other interconnected Modules. Figure 20 provides the main integration currently in completed. Other can be easily implemented if needed thanks to the functions of the HDT Core Infrastructure.

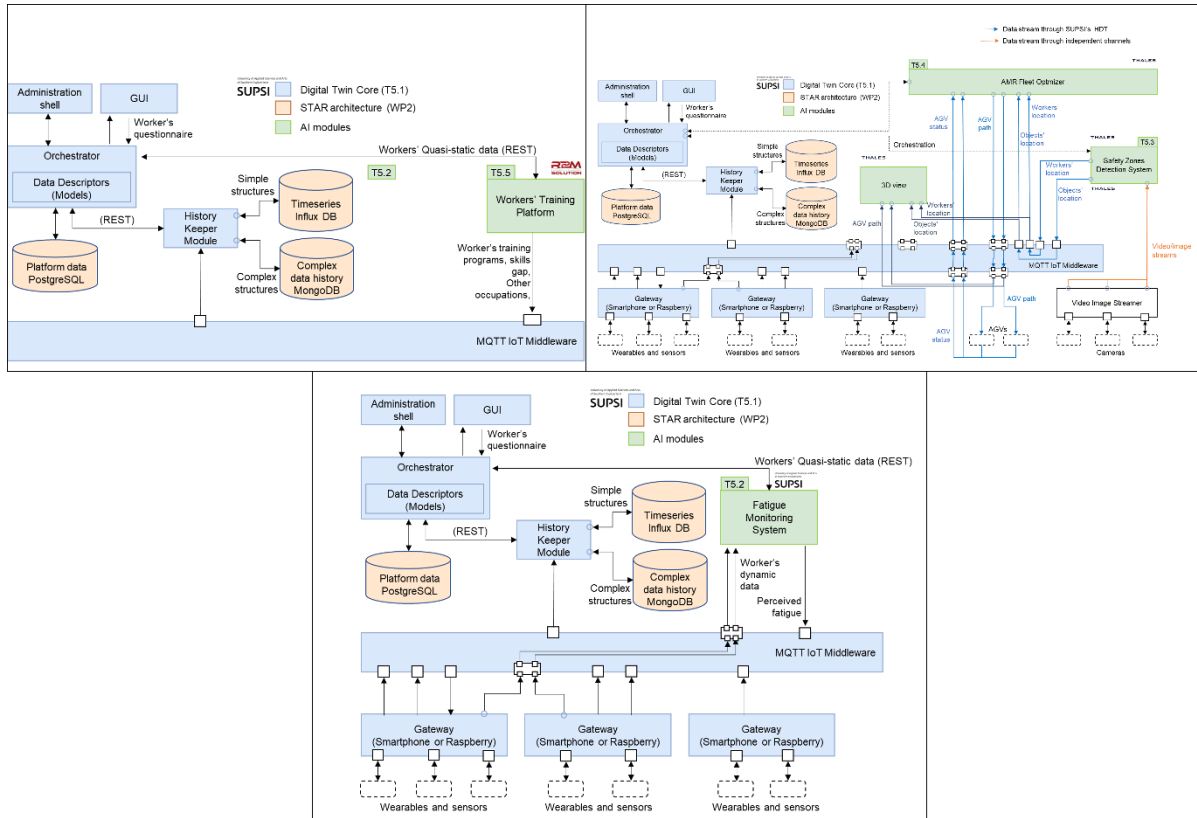


Figure 20: WP5 main integrations

For a comprehensive understanding of this integration paradigm, refer to the detailed insights presented in documents D5.1 and D5.2.

## 5 Testing, Validation, and Integration Roadmap

### 5.1 Lab Requirements, and Environment

In this deliverable, as previously stated, the primary focus is to offer an extensive range of knowledge pertaining to the testing, validation, and integration of the STAR platform. To facilitate the testing, validation, and integration of various services and modules provided by technology providers, DFKI is tasked with providing a dedicated server capable of hosting and executing all components from WP3, WP4, and WP5 module providers. Access to this server is extended to all component owners.

#### 5.1.1 General Asset List

At the outset of Task 6.2, we initiated the creation of a *General Asset List* and shared it with all project partners, requesting them to continually update the information for each respective component. Consequently, we now possess an encompassing, detailed, and referenceable Excel file that houses vital information for each component, along with relevant insights into integrated use cases and pilots.

This list encompasses various fields. In the following section, we will delve into the various information available in this list:

1. Component: The components directly associated with T6.2, intended for validation, testing, and integration into the DFKI server, provided within the scope of the STAR project.
2. Component Owner: The technology provider(s) responsible for offering the mentioned modules/technologies.
3. Related Task: The task within the designated work package that pertains to the respective technology/component.
4. Pilots to be Applied: This section identifies the intended pilot and corresponding use case scenarios where the component is utilized, tested, and validated.
5. Dependency with other components (Input/output): If the component/technology is interconnected with other component(s)/technology(ies) from different task(s). This relationship could involve providing inputs for other technologies or utilizing outputs from other technologies.
6. Component Version: The current version of the respective component.
7. Communication Protocol: The available protocol(s) to establish connectivity with the technology(ies) (e.g., Kafka, TCP-IP, MQTT, HTTPS/HTTP).
8. Availability (Code, Artifact): Indicates whether code or artifact is available from the asset(s) (e.g., Available, Proprietary).
9. Dockerization: Specifies whether the code/artifact can be dockerized. This is crucial for the effective packaging of delivered software into containers using Docker services, which ease the utilization in the service platform and CI/CD.
10. Hardware Requirements: The specific hardware requirements suitable for operating the component/asset (e.g., CPU, HDD, Memory).
11. Possible Lab Deployment Date: The anticipated date when the deployment of the component becomes feasible.
12. Status of the Components: Indicates whether the component is fully developed or currently under development.

13. API Documentation: Provides information about the presence of any documentation or link to documentation available for the component/asset.
14. Test Availability (Yes/No): Indicates the availability of any automated/code tests for the component/asset.
15. Bottom-up Requirement mapping: The bottom-up requirements for the corresponding component, which have been captured from WP2.
16. Status of the Requirement: This section specifies the status of each bottom-up requirement, indicating whether it is for example, "Done," "Partially Done," or "Pending."
17. Link to Source Code: Offers a hyperlink to the code, if any open-source code is available for the component/asset.
18. Contact Person: This section provides the information of the responsible person(s) for each of the components, including their names, roles, and contact details.

### 5.1.2 Hardware requirement

Of particular significance within this context is the specification of minimum requirements for each component. Building on this information, we have configured the service platform to embody capabilities that align with the specific needs of our ecosystem.

In order to accomplish this objective, we have documented the minimum hardware requirements necessary for the STAR platform to undergo testing, validation, and seamless integration within the service platform. The service platform has been designed to accommodate the usage and accessibility of all components. The hardware specifications for this service platform, hosted in the DFKI Kaiserslautern and accessible via the internet, are outlined below:

- *Disk Space: 100GB*
- *CPU: 8 Core*
- *RAM: 64GB*
- *Accessibility through the Internet*
- *Docker Service package installed*

Please note that these specifications have been established to ensure optimal performance and efficiency while utilizing the STAR platform within the service ecosystem. Consequently, they can test and evaluate their components on the server and finally all the components can be integrated into the STAR platform.

## 5.2 Supported Scenarios

In T6.2, "Service Platform Integration and Lab Validation," we performed validation exercises for both inter- and intra-Work Package (WP) components. However, in the initial version of this deliverable (D6.3), our focus was primarily on the validation of inter-WP components. Now, in this final version of the document (D6.4), we have expanded our scope to encompass both inter- and intra-WP components. In this specific task, we meticulously considered various supported scenarios that can be effectively implemented in the subsequent stages of this task.

Following, there are some of the potential scenarios explained.

### 5.2.1 Automation Tools

To facilitate validation, testing, and integration, we initially employ a manual setup. As each module/asset undergoes updates, technology providers are manually testing their artifacts to meet the task requirements. If automation CI/CD tools are necessary, we can leverage an integrated test service such as Jenkins or GitHub CI/CD services. This approach ensures efficient and reliable validation processes throughout the project.

### 5.2.2 Validation the Components

To validate the components based on the KPIs, technology providers begin with straightforward scenarios. Each Work Package (WP) completed its respective validation phase. Subsequently, in the next stage, they have conducted the intra WP validation. This approach ensures comprehensive and thorough validation across the project.

### 5.2.3 Evaluation the accuracy of the Architectures

The validation of each component is the responsibility of the respective component providers. They will perform validation for their artifacts/components each time they make updates. This validation process encompasses not only the validation of the components themselves but also the interaction and communication of the components with other components and pilots within and between Work Packages (WP). This approach ensures that each component is thoroughly tested and validated in the context of its integration with the overall system and other components.

### 5.2.4 The Process of Accessing the Data

To optimize the validation and testing phase, data samples and recordings for each component can be collected either offline or online. To access the essential input data for the components/software from technology providers, we have devised a comprehensive plan to store samples for each data source, encompassing various scenarios. These stored samples are served as the basis for testing the artifacts.

In the updated version, it may become necessary to establish communication with the hardware providers responsible for supplying data to the artifacts. This communication could involve interacting with devices like cameras, sensors, and actuators to ensure a seamless data flow, resulting in effective validation of the components.

The technology providers bear the responsibility of handling this communication and integration between the components and the data sources. This approach ensures a collaborative effort to ensure the successful functioning of whole STAR system.

## 5.3 Integration of technical components with the Star Secure Storage.

A pivotal aspect of this task 6.2 involves the seamless integration of components within the STAR platform, establishing a robust ecosystem. Our focus centers on identifying the infrastructural support required to interconnect various components effectively. This integration plays a crucial role in shaping our test pilots' success and, hence, in the STAR ecosystem.

To accomplish this objective, we have delved into the intricate details of the components that exhibit internal connections within each Work Package (subsections 4.3.3, 4.2.3, 4.1.3). This



thorough examination facilitates a comprehensive understanding of the interrelationships between each of the WP-level components.

In this section, we transition from an internal perspective to an intra-WP outlook, focusing specifically on the integration actions between selected technical components from various WPs with the Star Secure Storage.

The integration actions of the STAR Secure Storage were focused on onboarding various technical tools and assisting their training processes. More specifically, the STAR storage solution is a unified solution for storing/querying/managing data. Thus, it acts as a central component in the STAR architecture in order to facilitate mainly the necessary data pipelines used for the training of various tools.

One of the key tools is the AI Cyber Defence module which is now integrated with the secure storage in order to acquire Binary files, i.e., images, which are used for evaluating the mechanisms for the identification of poisoning and evasion attacks. To establish this communication, the AI Cyber Defence tool integrates the necessary TRINO connectors to query the respective resources to acquire the data required to perform the training of the tool. More specifically, the AI Cyber Defence tool has been built in order to support image streaming, as it is destined to perform in the context of visual quality inspection systems. It is used to support the quality inspection processing for the PCL and IBER pilots. Thus, in order to support the training of the tool, the STAR Secure Storage created the respective data buckets, including the images of the products that will be used as the point of reference for training the AI Cyber Defence models. The Shaver-shell, the deco-cap, and the soothers datasets have been onboarded on the secure storage for supporting the visual inspection actions for the PCL pilot, while the Assembly of Horizontal Lamellas Dataset was onboarded for the visual inspection actions for the IBER pilot. The integration of the AI Cyber Defence tool with the STAR storage has been completed, and the interconnection of the tools is also highlighted as part of the internal architecture of the AI Cyber defence Tools which can be found in D3.4.

In addition to the above-mentioned tools, in the context of the cross-work package integration actions, the Fatigue Monitoring System (FaMS) is integrated with the STAR Secure Storage. The former uses a machine learning model that estimates the exertion level of subjects based on static data (e.g., age, weight, etc.) and dynamic data (e.g., HR, EDA, skin temperature). The tool utilizes subjects' physiological data, while static data are collected through a questionnaire, so that to derive the stress level of the user. This 'AI module' can be used alone to understand the exertion level of the workers who are performing a task, or it can also be used by 'decision maker modules' to make human-aware decisions. In this context, the FaMS has been integrated with the STAR storage in order to take advantage of the timeseries DB features. The FaMS take advantage of the Hive connector in order to perform time-based queries.

One of the most significant integrations, due to the fact that it includes components and elements from multiple work packages, is the one demonstrated in the Workers Training Platform. Specifically, the Worker Training Platform began to be developed in work package 5, where different modules for access to occupational information were implemented. Subsequently, the benefit of integrating it with the NLP component that was being defined in WP4 was seen. In this way, the Multimodal Workers Training Platform was built, which appears as assets in the list of results of the STAR project.

Not only that, after discussions with WP7 and WP8, it was agreed that part of the training assets selected in these work packages would be part of the training course recommendation system offered by the Workers Training Platform. Also, that the Workers Training Platform would be integrated into the project's website, as a value-added service for visitors. Finally, and thanks to the capabilities offered by the WP2 and WP3 platform and security modules, the Worker Training Platform is carrying out integration tests with the secure storage module so that the database of training resources can be safely stored on the STAR platform and can be updated there, before it is periodically consulted by the Multimodal Training Platform.

These two paragraphs exemplify the cross-workpackage collaboration that has taken place at STAR and that continues to be addressed even during the last phases of the project. It is also important to mention that at the content level there has also been collaboration between work packages, since part of the theoretical-practical content generated in other workpackages, such as the materials on Human Centricity developed in WP5, have been added to the Marketplace and to the WP7 training assets.

The STAR secure storage has been integrated with the aforementioned tools, while potential integration actions with other tools can be done until the end of the pilots testing, as the interfaces are known and the tool onboarding process is straight forward.

## 6 Conclusion

The STAR project encompasses a diverse range of functionalities and technologies tailored for research and industrial applications. The project's primary objective is to research, implement, validate, and demonstrate trusted AI technologies that can be effectively utilized across various production scenarios. This deliverable focuses on activities related to testing, validation, and integration of components into the STAR platform, covering the following key themes:

- **Integration Platform and its relation to the STAR Reference Architecture:** We delve into the STAR reference architecture and the deployment/physical diagram at the WP level, specifically designed for lab validation.
- **Source code Repository and Tools:** We make efforts to establish a repository to host codes and artifacts for each component and technology, with a focus on Dockerization and its support.
- **Comprehensive Documentation of Components:** A detailed list of target components from WP3, WP4, and WP5, utilized in T6.2, is provided. Each component is accompanied by a brief description, its relation to the STAR reference architecture, dependencies on other packages/components, installation/deployment guidelines, documentation, and test cases. The test cases, presented in separate tables for each component, outline actions and expected results for atomic inter-component and WP-level communication. Furthermore, each component is, presented in a separate table, linked to the specific scenarios in which they are utilized within the pilots. Finally, in each WPs section, we provide an overall overview of how the components within the WPs communicate with each other at a detailed level.
- **Test, Validation, and Integration Roadmap:** This section outlines the activities carried out in T6.2 throughout the task's lifespan, aimed at facilitating communication among different technology providers and pilots to ensure a robust test, validation, and integration phase for the STAR ecosystem. It encompasses the preparation of the lab environment, general asset list documentation, and component requirements for initializing an optimal service platform.
- **Supported Scenarios:** We provide guidance for technology providers to conduct and manage both inter- and intra-WP testing, validation, and integration. This includes communication approaches, CI/CD methods, and data sampling for the components.
- **Lastly, we offer a comprehensive insight into the STAR ecosystem, illustrating how the integration of components from various Work Packages enables us to achieve the final version of the STAR ecosystem. This integration binds together diverse components, shaping the ultimate form of the STAR ecosystem. Furthermore, we also noted that there are a few remaining tasks to address in the coming months of the project's duration.**

Overall, this deliverable contributes to the realization of a cohesive and reliable STAR ecosystem by addressing crucial aspects related to testing, validation, and integration of its components and technologies.

## References

Reference	Name of document
[Afzal-Housmand 21]	Afzal-Housmand, S. H. (2021, September). A Perfect Match: Deep Learning Towards Enhanced Data Trustworthiness in Crowd-Sensing Systems. <i>In 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)</i> , (pp. (pp. 258-264). IEEE.).
[Afzal-Housmand 23]	Sam Afzal-Housmand, D. P. (2023). Explainable Artificial Intelligence to Enhance Data Trustworthiness in Crowd-Sensing Systems. <i>19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)</i> , DOI 10.1109/DCOSS-IoT58021.2023.00093 .
[Driessen 10]	Driessen, V. (2010, January 5). <i>A successful Git branching model</i> . Retrieved August 2021, from nvie.com: <a href="https://nvie.com/posts/a-successful-git-branching-model/">https://nvie.com/posts/a-successful-git-branching-model/</a>
[IBM 21]	IBM Cloud Education. (2021, June 23). <i>Containerization</i> . Retrieved August 2021, from <a href="https://www.ibm.com/cloud/learn/containerization">https://www.ibm.com/cloud/learn/containerization</a>
[Kisller 21]	E. Kisller, "What Is a Container Registry? And Why Do I Need One?," 26 March 2021. [Online]. Available: <a href="https://jfrog.com/knowledge-base/what-is-a-container-registry/">https://jfrog.com/knowledge-base/what-is-a-container-registry/</a> . [Accessed August 2021]
[Souza 18]	H. Souza, "How to dockerize any application", May 2018, available at: <a href="https://hackernoon.com/how-to-dockerize-any-application-b60ad00e76da">https://hackernoon.com/how-to-dockerize-any-application-b60ad00e76da</a> , last accessed at: March 2020
[STAR-D2.7]	STAR, "D2.7: STAR Reference Architecture and Blueprints- Final version", 2022-06-30.
[STAR-D3.1]	STAR, "D3.1 – Decentralized Reliability for Industrial Data and Distributed Analytics- Initial version", 2022-04-20
[STAR-D3.6]	STAR, "D3.6 – Security and Data Governance Infrastructure-Final version", 2023-08