

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

D6.3 – Integrated STAR Platform-Initial version

Dissemination level	PU -Public
Type of Document	Report
Contractual date of delivery	30/06/2022
Deliverable Leader	DFKI
Status - version, date	Final v1.0, 18/07/2022
WP / Task responsible	WP6
Keywords:	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 initial version of the Integrated STAR Platform is the document that provides the activities and the progress that has been done by the task 6.2-Service Platform Integration and Lab Validation (M5-M30).

The 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). The relationship between the STAR reference architecture and the integrated platform is addressed in this document. We explained the repository's technologies/tools which are planned to be utilized in the STAR project to ease the CI/CD and collaboration between different partners. Furthermore, we gather 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.

Finally, we state the roadmap for task 6.2, and what are the possible action points for the next step in this task.

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

Document History			
Version	Date	Contributor(s)	Description
V0.1	03/05/2022	DFKI	First document release with ToC
V0.2	10/05/2022	INTRA	Revised ToC and added descriptions of contributions
V0.3-0.5	24/06/2022	INTRA / DFKI	Added Source Code, Repository & Tools section. Added RMS and DLSDR components
V0.6	30/06/2022	JSI	Review
V0.7	30/06/2022	DFKI	Review, finalize the format
V0.8	15/07/2022	DFKI	Integrate review comments
V0.9	15/07/2022	DFKI	Integrate components in the list of use cases
V1.0	18/07/2022	INTRA	QA and creation of the final submitted version

Table of Contents

- EXECUTIVE SUMMARY 2**
- TABLE OF FIGURES..... 6**
- DEFINITIONS, ACRONYMS AND ABBREVIATIONS 7**
- 1 INTRODUCTION..... 8**
 - 1.1 OVERVIEW AND PURPOSE..... 8
 - 1.2 RELATIONSHIP TO OTHER DELIVERABLES..... 8
 - 1.3 DELIVERABLE STRUCTURE 9
- 2 FROM REFERENCE ARCHITECTURE TO INTEGRATED PLATFORM 10**
 - 2.1 THE STAR REFERENCE ARCHITECTURE 11
 - 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 14
 - 2.4.1 *Physical View of Active Learning Module..... 15*
 - 2.4.2 *Physical View of the Explainable AI (XAI) Module 15*
 - 2.4.3 *Physical Views of the Reinforcement Learning Modules 15*
 - 2.4.4 *Physical View of Safety Zones Detection Module..... 16*
 - 2.4.5 *Physical View of Simulated Reality Module..... 17*
 - 2.5 PHYSICAL VIEW OF THE STAR HUMAN CENTRIC DIGITAL TWIN MODULES 18
- 3 SOURCE CODE, REPOSITORY & TOOLS 19**
 - 3.1 TECHNOLOGIES AND TOOLS 19
 - 3.2 VERSION CONTROL SYSTEM AND REPOSITORY: GIT AND GITLAB 20
 - 3.3 CONTAINERIZATION 22
 - 3.3.1 *Docker 23*
 - 3.3.2 *Dockerfile 24*
 - 3.3.3 *Docker Compose..... 24*
 - 3.3.4 *Docker Usage..... 24*
 - 3.4 CONTAINER REPOSITORY & REGISTRY MANAGEMENT 25
 - 3.5 MANAGEMENT/MONITORING WITH PORTAINER 25
- 4 THE STAR COMPONENTS 27**
 - 4.1 SECURITY AND DATA GOVERNANCE 27
 - 4.1.1 *Components..... 27*
 - 4.1.2 *Use Cases..... 42*
 - 4.2 SAFE, TRANSPARENT AND RELIABLE HUMAN-ROBOT COLLABORATION 42
 - 4.2.1 *Components..... 42*
 - 4.2.2 *Use Cases..... 52*
 - 4.3 HUMAN CENTRED SIMULATION AND DIGITAL TWINS 53
 - 4.3.1 *Components..... 53*
 - 4.3.2 *Use Cases..... 69*
- 5 TESTING, VALIDATION, AND INTEGRATION ROADMAP 70**
 - 5.1 LAB REQUIREMENTS, AND ENVIRONMENT 70
 - 5.2 SUPPORTED SCENARIOS 71
- 6 CONCLUSION 72**

REFERENCES 73
APPENDIX A 74

Table of Figures

FIGURE 1: HIGH LEVEL REFERENCE MODEL FOR THE FUNCTIONALITIES OF THE STAR PLATFORM.	11
FIGURE 2: STAR FUNCTIONAL MODULES AND LOGICAL VIEW OF THE ARCHITECTURE [STAR-D2.7].....	12
FIGURE 3: DEPLOYMENT DIAGRAM FOR THE CYBERSECURITY MODULES OF THE STAR ARCHITECTURE (I.E., MODULES DEVELOPED IN WP3)- CAPTURED FROM D2.7	14
FIGURE 4: THE DEPLOYMENT DIAGRAM FOR AI SERVICE.....	15
FIGURE 5: PHYSICAL VIEW OF THE STAR XAI COMPONENT/MODULES	15
FIGURE 6: PHYSICAL VIEW OF THE SAFETY ZONE DETECTION & FLEET OPTIMIZER MODULES.....	16
FIGURE 7: PHYSICAL VIEW OF THE SAFETY ZONE DETECTION.	17
FIGURE 8: PHYSICAL VIEW FOR THE SIMULATED REALITY.	18
FIGURE 9: HUMAN DIGITAL TWIN CORE INFRASTRUCTURE DEPLOYMENT SHOWCASE	18
FIGURE 10 STAR-AI GITLAB PAGE.....	21
FIGURE 11 A COMPLETE GIT BRANCHING MODEL.....	22
FIGURE 12 AI CYBER DEFENCE TOOL INTERNAL ARCHITECTURE	33
FIGURE 13 CONSOLE OUTPUT OF DOCKER PS	34
FIGURE 14 SSPM HIGH LEVEL ARCHITECTURE	40
FIGURE 15 ARCHITECTURE OF THE COMPONENTS FOR NATURE LANGUAGE PROCESSING	48
FIGURE 16 SAFETY ZONE DETECTION & AMR FLEET OPTIMIZER PHYSICAL VIEW	54
FIGURE 17 DEPLOYMENT VIEW OF HDT COMPONENTS.....	63

Definitions, Acronyms and Abbreviations

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

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 will provide 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-defense 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 supposed to be applied in the industrial environment and need to be proven and tested in different testbeds with different scenarios. The WP6 aims to integrate, validate, and evaluation of 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 will continue 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 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 will be 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 needed to be fulfilled within this task. Finally, for the validation phase, we consider the integrated platform and its components, and functionalities are supposed to be 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 D6.3 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 6.3, 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- "Decentralized Reliability for Industrial Data and Distributed Analytics".

The D3.1 is more 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 will 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 will be 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.
- Finally, the next step of task 6.2 is related to the testing, validating, and integrating different artifacts in the integrated platform. Lab requirement, initialization and supported scenarios related to the task.

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-defense 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-defense 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, our project also researches 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 introduces 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 is driven by the project’s activities during the first semester of STAR’s lifetime, while in the next phase, the STAR-RA will incorporate inputs from the final versions of the STAR reference scenarios and technologies specifications. Moreover, between the two iterations, the project will use the first version of the architecture to drive the development and integration of technical/technological systems in WP3, WP4, WP5, and WP6. This will enable the project to receive feedback from the actual, implementation, deployment, and use of the first version of the architecture. Accordingly, the project will use this feedback to fine-tune the specification of the architecture. In consequence, there is 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:

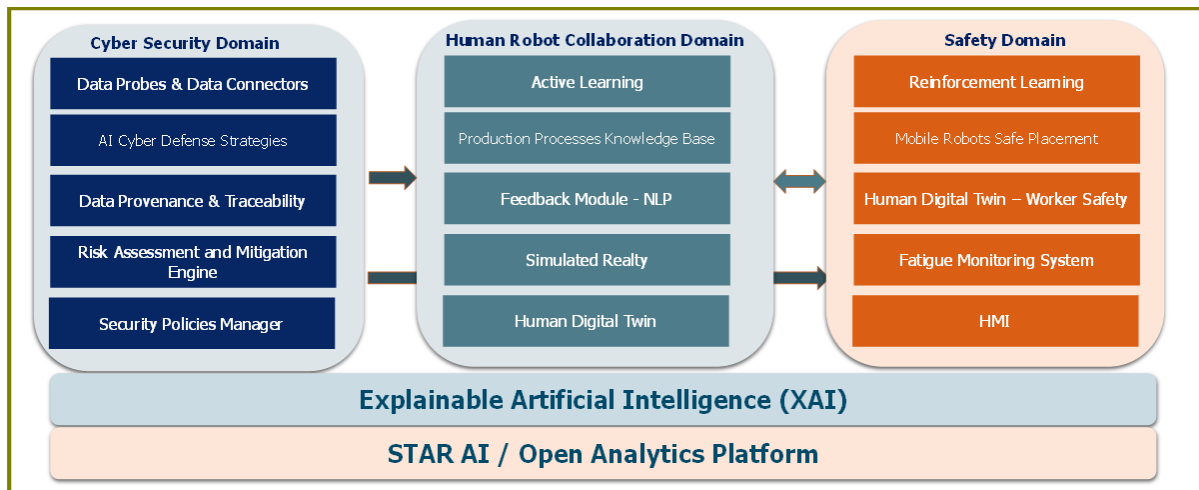


Figure 1: High Level Reference Model for the Functionalities of the STAR Platform.

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 fulfill 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 defense 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.

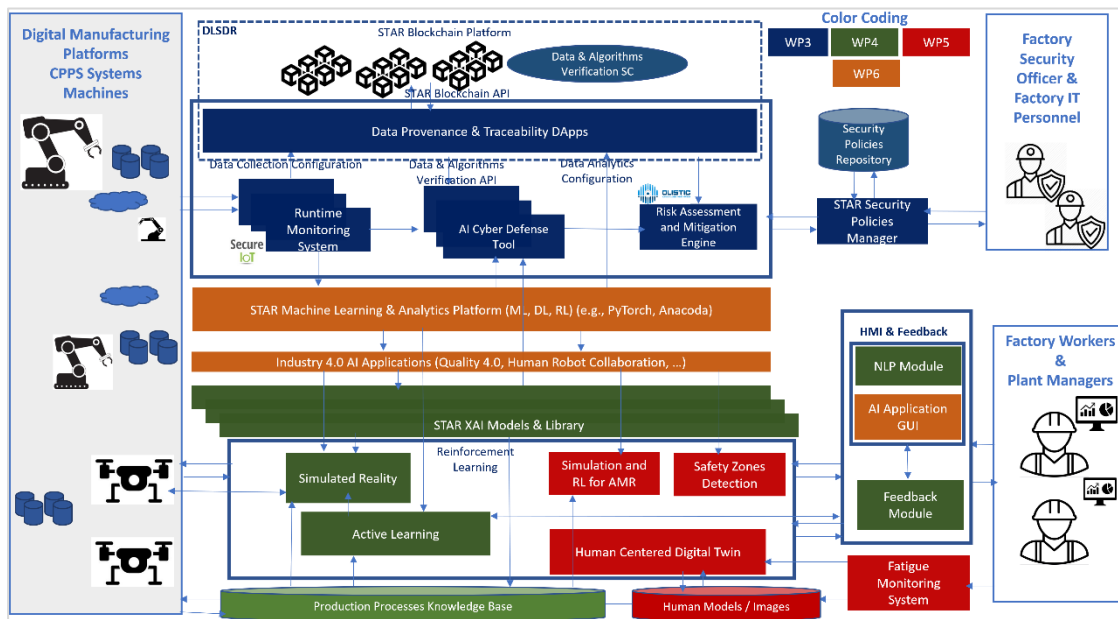


Figure 2: STAR Functional Modules and Logical View of the Architecture [STAR-D2.7]

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 Defense 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.

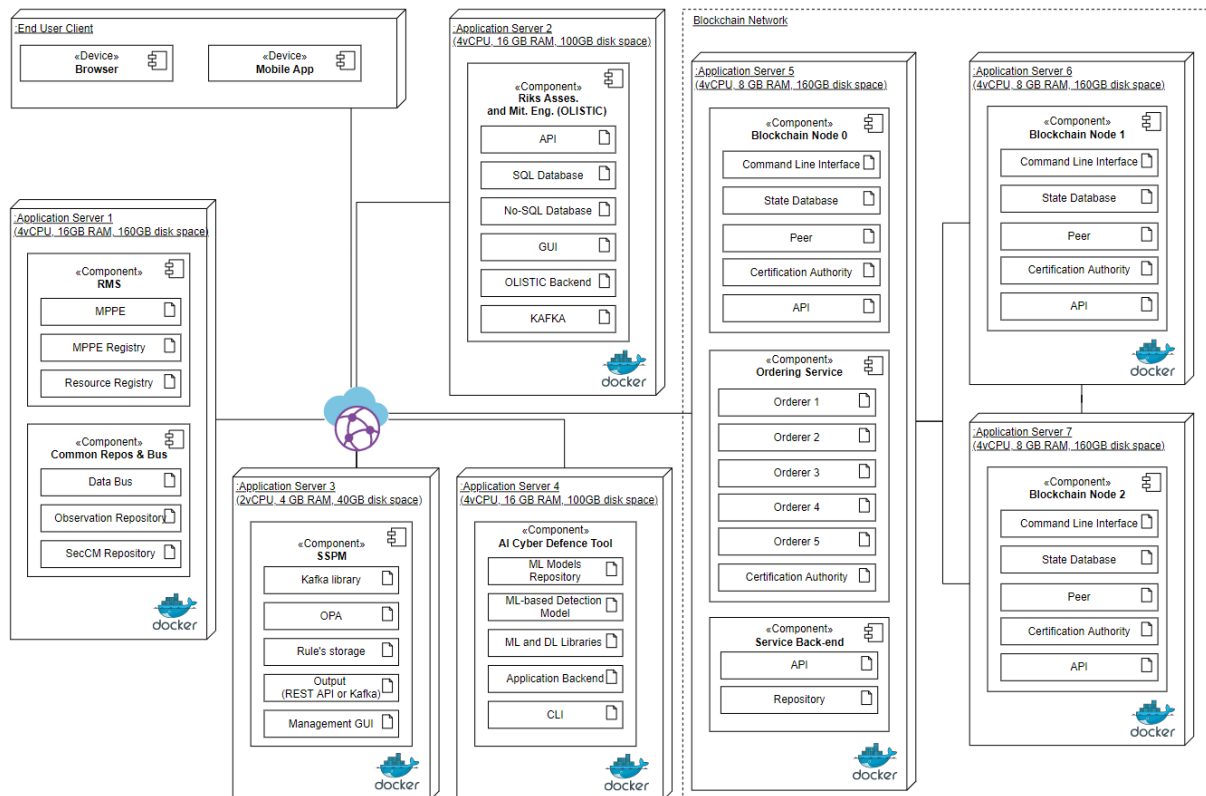


Figure 3: Deployment Diagram for the Cybersecurity Modules of the STAR Architecture (i.e., modules developed in WP3)- Captured from D2.7

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, and also possible to be containerized in docker images.

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.

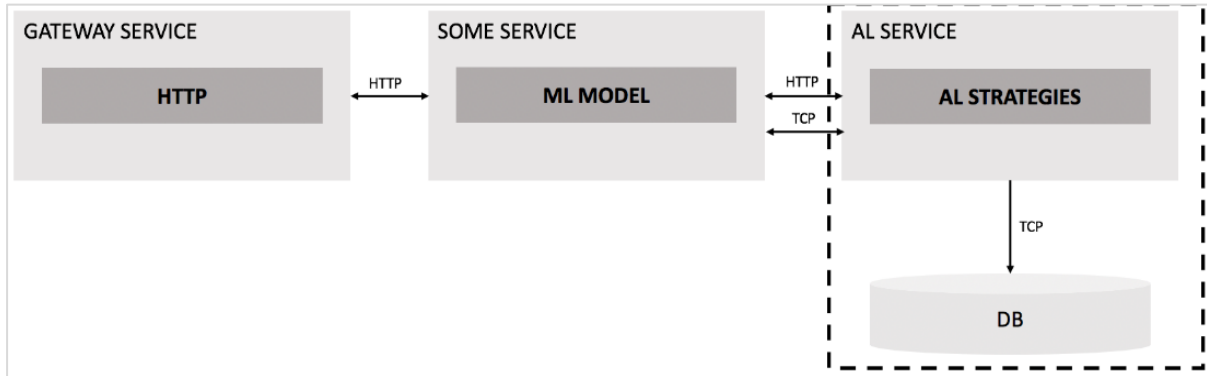


Figure 4: The Deployment Diagram for AL Service.

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 will be utilized.

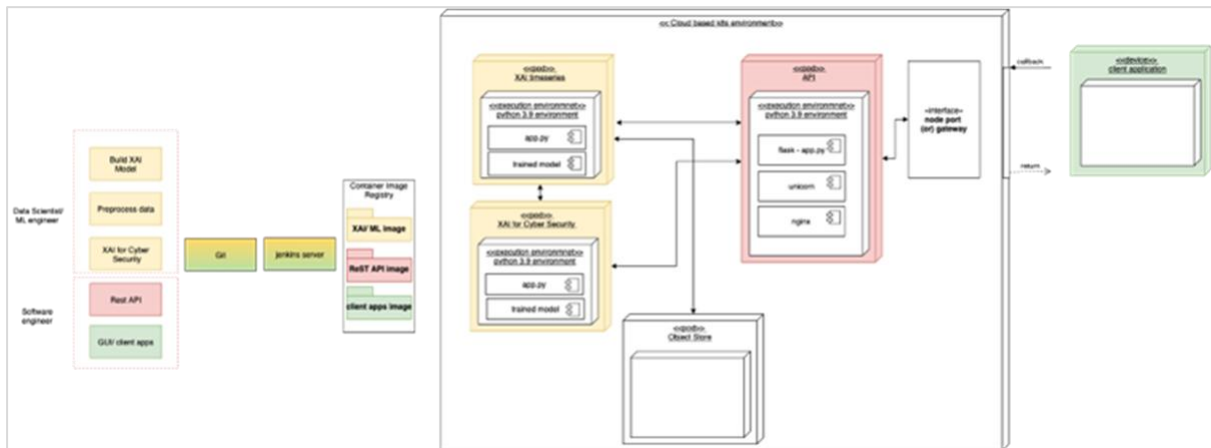


Figure 5: Physical View of the STAR XAI Component/Modules

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 communicated to each other relying on the Human Centric Digital Twin.

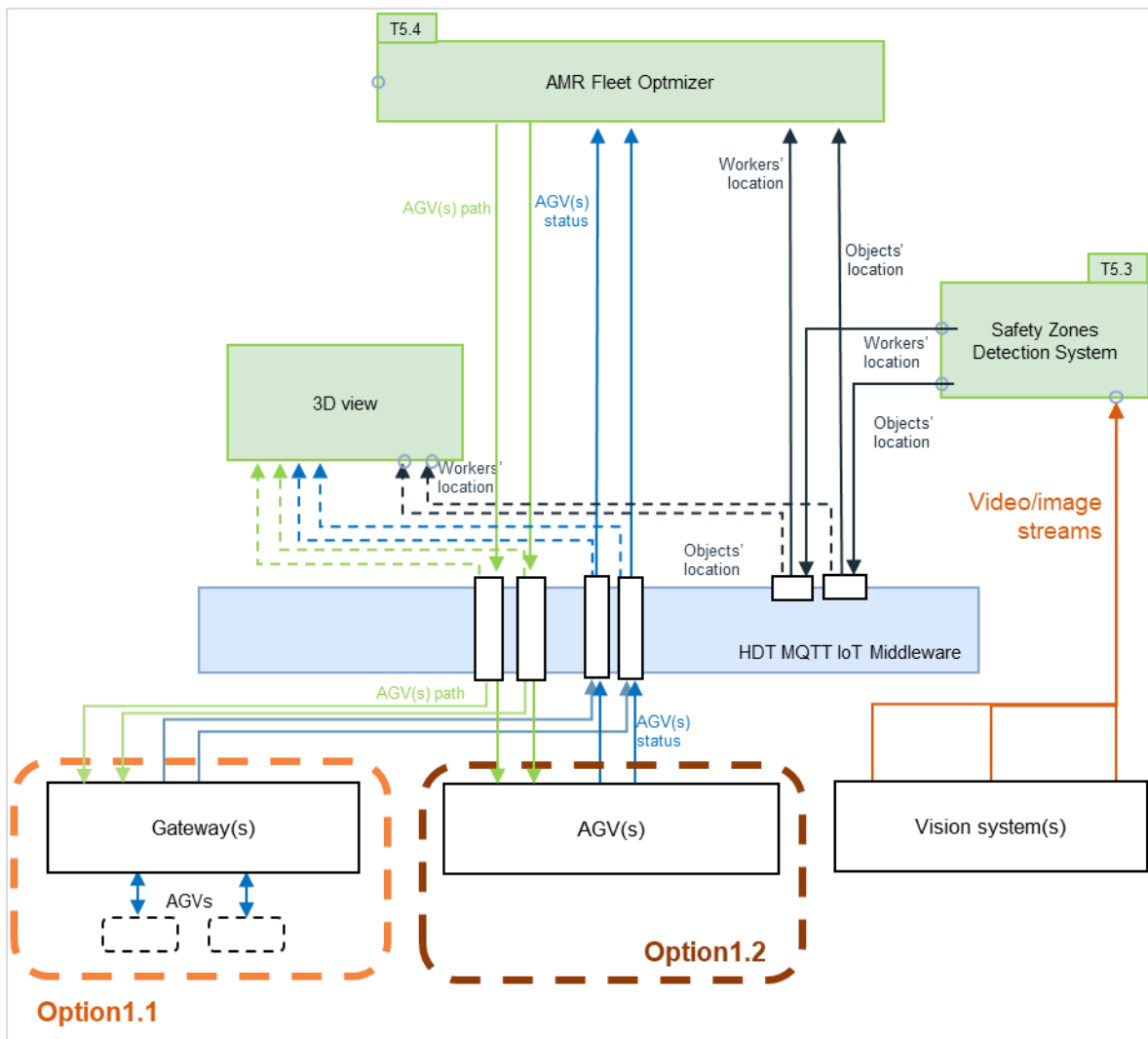


Figure 6: Physical View of the Safety Zone Detection & Fleet Optimizer modules.

Figure 7 presents 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. Finally, the elements extractor engine combines the two deep learning algorithms

(The skeleton reconstruction to follow the human gesture and pose, and the detection and classification of non-static objects in the scene).

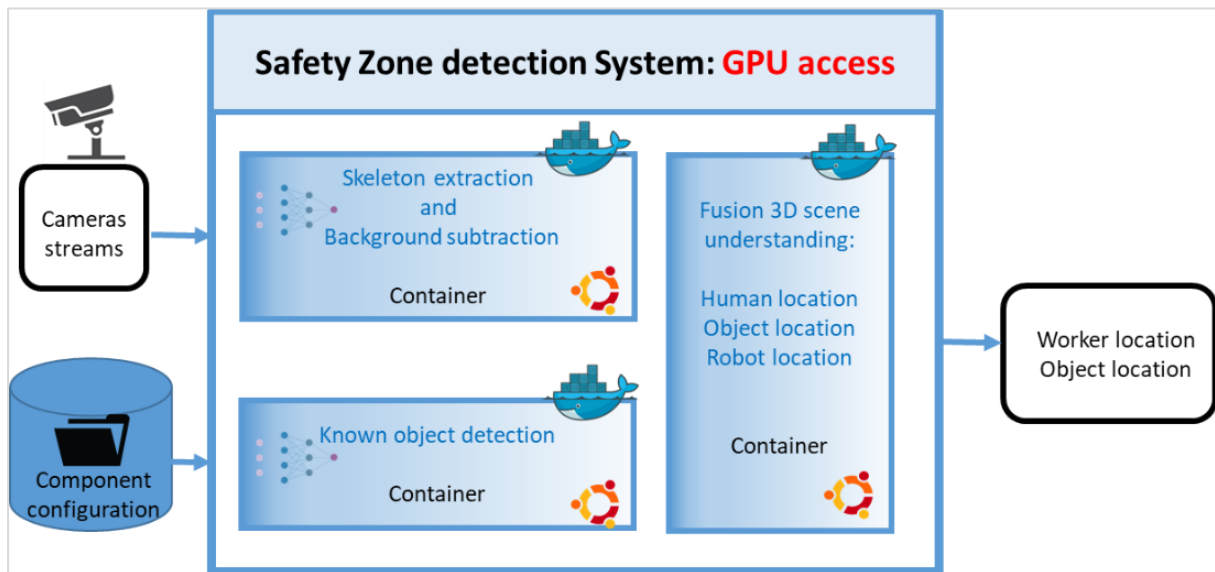


Figure 7: Physical View of the Safety Zone Detection.

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.

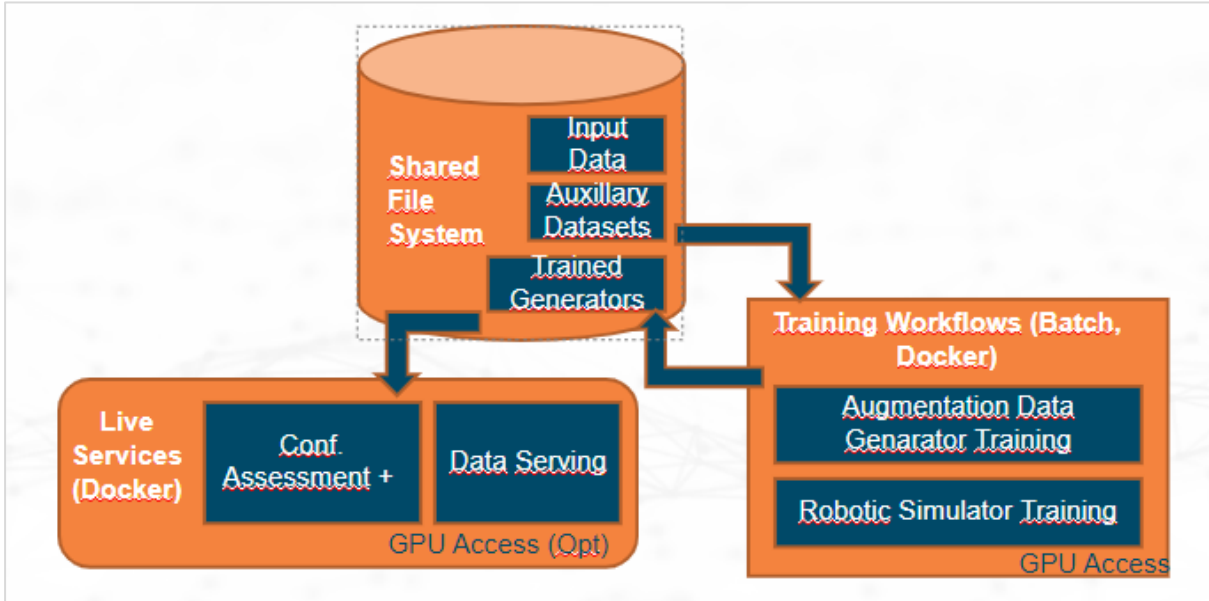


Figure 8: Physical View for the Simulated Reality.

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). Specific instances will be deployed to support the different use cases of the project.

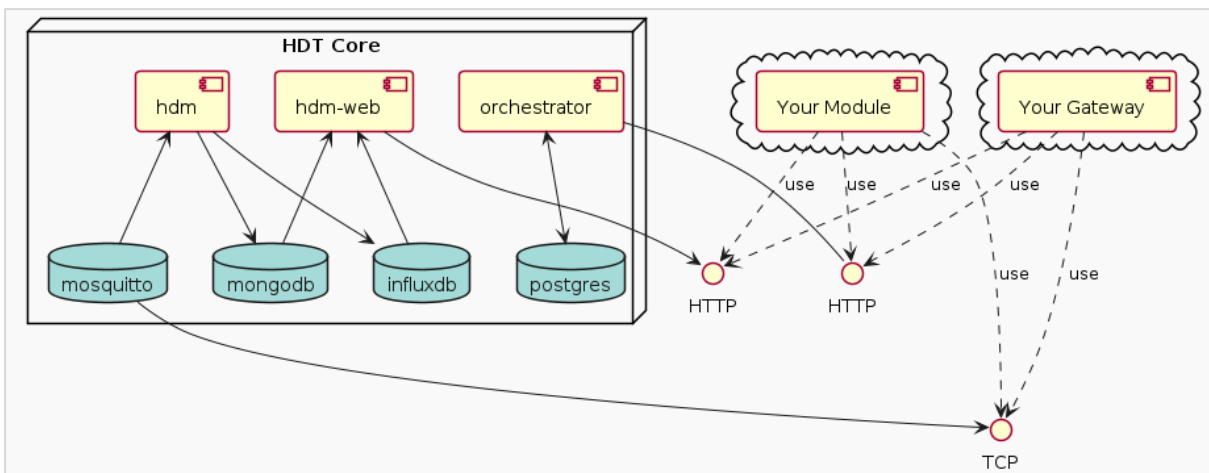


Figure 9: Human Digital Twin Core Infrastructure deployment showcase

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 will 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:

- **Hetzner¹**: 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.
- **Git²**: 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.
- **GitLab³**: 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 GitLab repository. GitLab offers the option of self-hosting a GitLab server on-premises, however, for the needs of the STAR project a GitLab instance hosted on gitlab.com has been deemed sufficient, provided that the access to the code be restricted to the project developers.
- **Docker⁴**: 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 Daemon⁵**: 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 Registry⁶**: An application that implements a private Docker Registry in which one can store and distribute the Docker images of the projects'

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

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

³ GitLab official website: <https://about.gitlab.com/>

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

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

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

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**⁷: An open-source tool for managing container-based applications in various virtualization environments. It can be used to set up and manage the environment, 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 GitLab

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 GitLab⁸. Git serves as the Version Control Systems (VCS), while GitLab 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 GitLab repository. The instantiated STAR GitLab group can be found under the following URL: <https://gitlab.com/star-ai> (see Figure 10 below). Under this group, several sub-groups for the STAR components and services have been created (and will continue to be created throughout the development process). Each subgroup shall host a set of repositories, corresponding to the various modules comprising the respective component.

Subgroups and the repositories they host are accessible only to the project partners developing each tool and, in addition, to INTRA for administration purposes. Each component sub-group 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:

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

⁸ <https://gitlab.com/>

- **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.

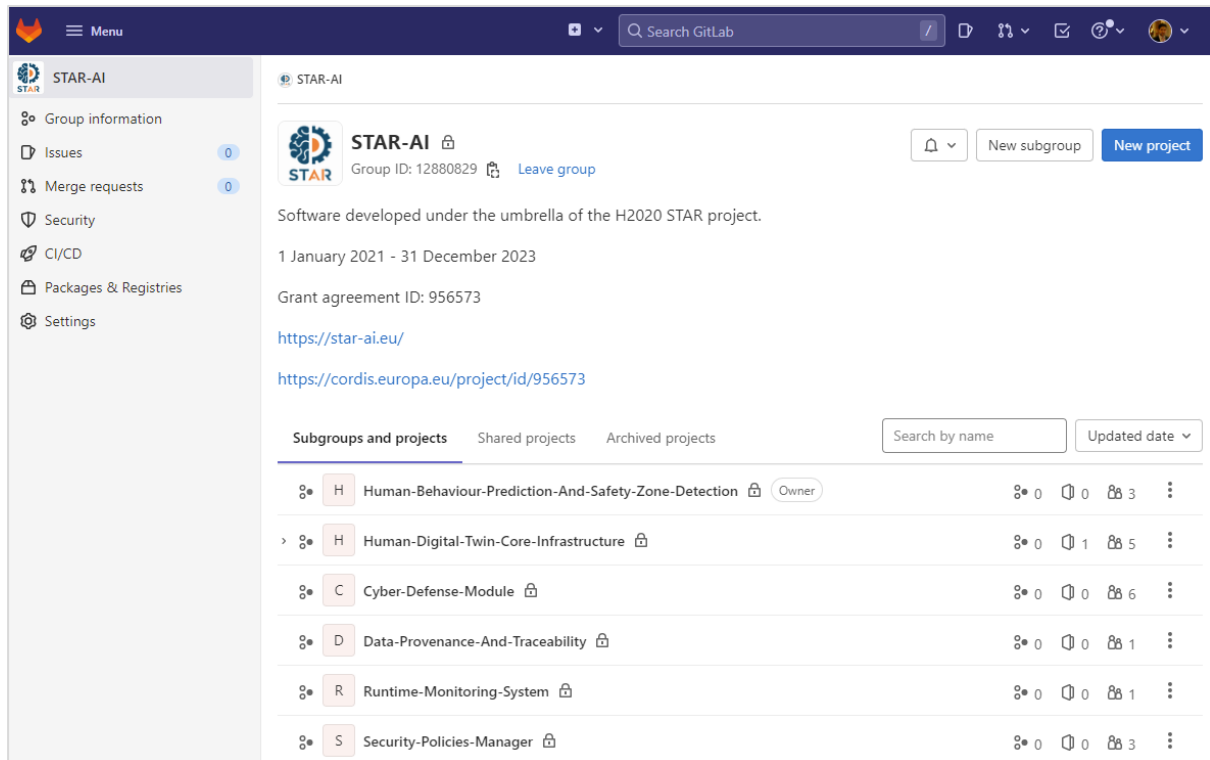


Figure 10 STAR-AI GitLab page

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 Mr. Vincent Driessen "A successful Git branching model" and a complete version of which is shown in Figure 11.

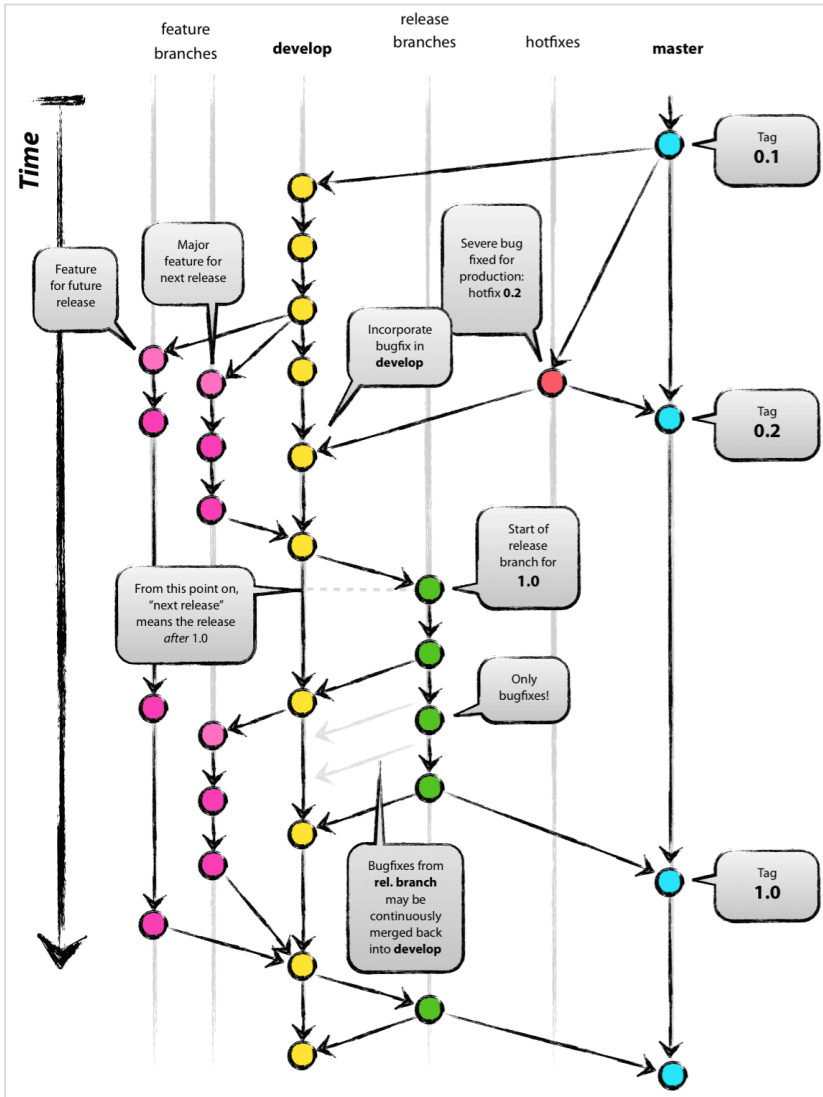


Figure 11 A Complete Git branching model⁹

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

⁹ <https://nvie.com/posts/a-successful-git-branching-model/>

resource-efficient than virtual machines (VMs), containers have become the de facto compute units of modern cloud-native applications.

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

3.3.1 Docker

As mentioned in D2.7 [STAR-D2.7] for the STAR software packaging we have considered Docker¹⁰ 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 Registry¹¹ or DockerHub¹², 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

¹⁰ <https://docs.docker.com/>

¹¹ Docker Registry official website: <https://docs.docker.com/registry/> (accessed August 2021)

¹² DockerHub official website: <https://hub.docker.com/> (accessed August 2021)

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.

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 will be accompanied by a "*docker-compose.yml*" file which will facilitate its installation. Additionally, different collections of interoperable components that will be used as solutions for the STAR use cases will be 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 Compose¹³.

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.

¹³ <https://docs.docker.com/compose/gettingstarted/>

- 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 reflect a distinct deployment of the same container. A specific image is identified by either its tag or its own unique reference [Kisller21].

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 Portainer¹⁴.

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

¹⁴ <https://www.portainer.io/products-services/portainer-community-edition/>

- 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).
 - 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 Deployment¹⁵ documentation. General documentation along with user and configuration guides can be found in Portainer's Documentation¹⁶.

¹⁵ <https://portainer.readthedocs.io/en/stable/deployment.html>

¹⁶ <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 stack¹⁷ 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:** to collect monitored data by using Elastic MetricBeats deployed to the Manufacturing Plant.
- **Logstash:** Raw monitored Data are transformed and filtered to match the used Data Model and identified rules.

¹⁷ <https://www.elastic.co/elastic-stack/>

- **Kafka & ElasticSearch:** the collected pre-processed data are published to the Data Bus (Kafka) to be accessed by the Security Policies Manager & ElasticSearch for permeate persistence, visualization and monitoring.
- **Kibana:** for persisted data visualization

RMS is also using MongoDB¹⁸ for the configuration repository and Java Spring Boot¹⁹ framework for developing its microservices with Spring Framework.

4.1.1.1.4 Availability

The runtime monitoring system will be available under the Runtime-Monitoring-System GitLab group²⁰.

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.

¹⁸ <https://www.mongodb.com/>

¹⁹ <https://spring.io/projects/spring-boot>

²⁰ <https://gitlab.com/star-ai/runtime-monitoring-system>

In the 'logstash' folder:

- config/
 - 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 D3.5 [STAR-D3.5]. Information on the RMS API can be found in section 3.2 of D3.5 [STAR-D3.5].

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	DSM should have been already specified		
Expected Outcome	A DSM is persisted, and an ID is assigned to it		
Actions	Expected Result	Result	Comment
The user sends an HTTP POST request to /data_source/dsm .	The user receives an HTTP response with the persisted DSM and an id assigned to it.	DSM	

Test ID	RMS-02		
Title	Retrieval of a Data Source Manifest (DSM) record		
Pre-Requisite	DSM should have already been persisted and its id should be known		
Expected Outcome	The requested DSM instance is returned		
Actions	Expected Result	Result	Comment

The user sends an HTTP POST request to <code>/data_source/ :id</code> .	The user receives an HTTP response with the persisted DSM.	DSM	
--	--	-----	--

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].

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 Node²¹

²¹ Peer Node downloadable from DockerHub: <https://hub.docker.com/r/hyperledger/fabric-peer>

- A CouchDB where the ledger state is being persisted²²
- A Certificate Authority (CA)²³
- A Command-Line Interface (CLI)²⁴
- 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 will be available under the Data-Provenance-And-Traceability GitLab group²⁵.

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 DockerHub²⁶. The deployment process has been made semi-automatic by employing Docker Compose²⁷ 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:

- Distributed Ledger Node Management

²² CouchDB downloadable from DockerHub: <https://hub.docker.com/r/hyperledger/fabric-couchdb>

²³ Fabric CA downloadable from DockerHub: <https://hub.docker.com/r/hyperledger/fabric-ca>

²⁴ Fabric CLI downloadable from DockerHub: <https://hub.docker.com/r/hyperledger/fabric-tools>

²⁵ <https://gitlab.com/star-ai/data-provenance-and-traceability>

²⁶ Hyperledger’s user account on DockerHub: <https://hub.docker.com/u/hyperledger>

²⁷ Docker Compose official documentation: <https://docs.docker.com/compose/>

- 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 POST request to /processor_config/pm .	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 GET request to /processor_config/:id .	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 current implementation of the tool 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 12. The reader can refer to D3.3 for more details on the AI Cyber Defence tool and the defence strategies put forth.

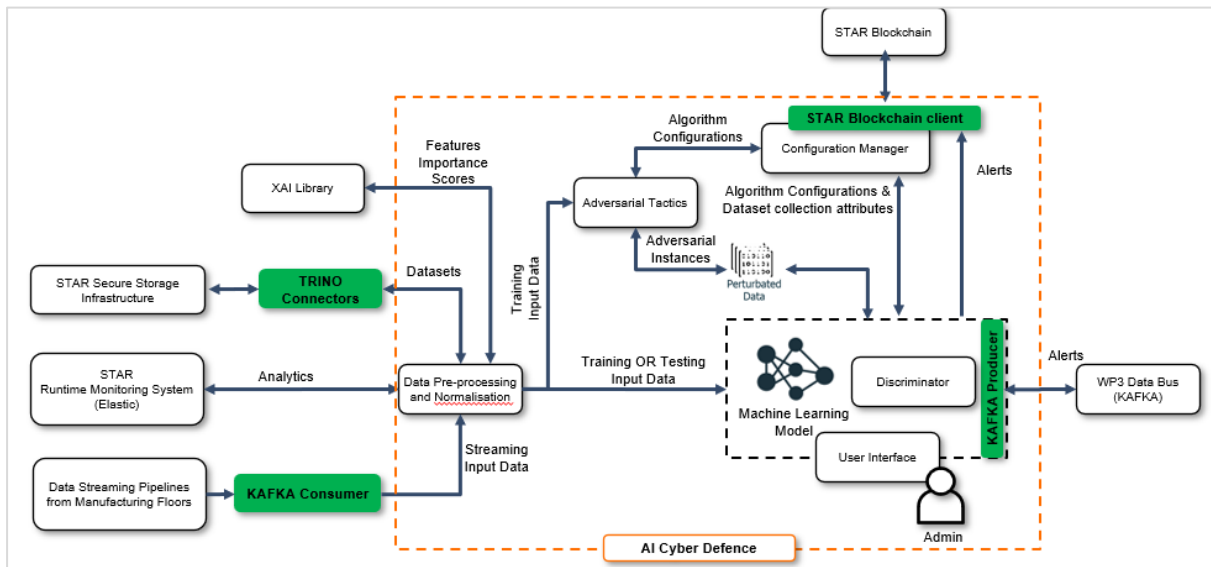


Figure 12 AI Cyber Defence tool internal architecture

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 will work 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). Although the specific docker compose will be in reality configured based on each demonstrator needs, the docker-compose.yml file currently used is provided in Appendix XXX.

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
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
```

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 will acquire datasets (e.g., images) for training purposes of the AI models. To establish this communication, the AI Cyber Defence tool will integrate the necessary TRINO connectors.
- STAR Runtime Monitoring system: As will be also explained in Section 3.5.1, the runtime monitoring system will be 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.
- Data streaming pipelines: This entity refers to the data sources that will be 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 not been documented yet and are still under development.

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"> • Pretrained adversarial neural network that performs the detection • Preconfigured parametrization of the detection algorithm

Expected Outcome	<ul style="list-style-type: none"> 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"> Raise of alarm provision of a confidence level on the prediction 	Not yet executed.	The structure of the output has been defined in D3.5

Test ID	AICD-02		
Title	Detect the presence of an adversarial examples in the training datasets.		
Pre-Requisite	<ul style="list-style-type: none"> Dataset to be stored on the Storage infrastructure of STAR Pre training model for the detection of poisoning attacks 		
Expected Outcome	<ul style="list-style-type: none"> Detection of poisonous instances in the datasets 		
Actions	Expected Result	Result	Comment
Adversarial instances are present in training datasets stored in the STAR Storage Infrastructure	<ul style="list-style-type: none"> Raise of alarm Detection of specific instance in a dataset. 	Not yet executed.	The structure of the output has been defined in D3.5

Test ID	AICD-03		
Title	Correct acquisition/validation of tool configuration through STAR Blockchain		
Pre-Requisite	<ul style="list-style-type: none"> Blockchain APIs are known Initial configuration set has been stored on the blockchain 		
Expected Outcome	<ul style="list-style-type: none"> Positive response if the configuration is valid and negative otherwise. 		
Actions	Expected Result	Result	Comment

The AI Cyber Defence tool triggers the Blockchain API to acquire the correct configuration.	Positive response if the configuration is valid and negative otherwise.	Not yet executed.	
---	---	-------------------	--

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 will 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 will enable the risk management and the identification and visualization of risks through comprehensive and reactive visualization, while it will provide the means to the security officer to manage the life cycle of mitigation actions that will help to eliminate or control risk events that have been detected by the monitoring mechanisms 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 will be 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
```

This first execution shall delay a little because it pulls pre-built images from online repositories (e.g. Docker Hub). Although the specific docker compose will be in reality configured based on each demonstrator needs, the docker-compose.yml file currently used is provided in APPENDIX A.

4.1.1.4.5.1 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, a report is generated in PDF format or in a structured JSON format to be shared with other interested components.

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"> The administrator has the knowledge on the assets taking part in the production lines of a factory The administrator has identified the relationship connecting the assets together 		
Expected Outcome	<ul style="list-style-type: none"> A graph-based representation of the environment is generated 		
Actions	Expected Result	Result	Comment
The administrator adds the assets	<ul style="list-style-type: none"> A list of assets is 	A graph-	

through a structured and well-defined data entry process.	<p>created.</p> <ul style="list-style-type: none"> The backend databases store the provided information 	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"> The SPM triggers the APIs of RAME to inform the latter about the detection of an abuse case. The database of RAME has already stored abuse cases that have been defined by the administrator. 		
Expected Outcome	<ul style="list-style-type: none"> 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"> A new attack scenario is generated denoting the presence of an abuse case. The risk assessment can be triggered in order to derive the risk level of the event. 	A risk level is associated with the assets that faced the abuse case.	

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 personnel of the factory, 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 this risk to the Risk Assessment module Olistic, generating 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 is implemented as a Python application that wraps OPA as an external service and acts as a middle-man regarding the other components by gathering the input from RMS and the XAI component, evaluating the defined policies on the given input, and returning the output. The input is collected, using a Kafka consumer, via the specific topics published by the RMS and the AI Cyber Defence components on the Data Bus. OPA manages both input and output in JSON format.

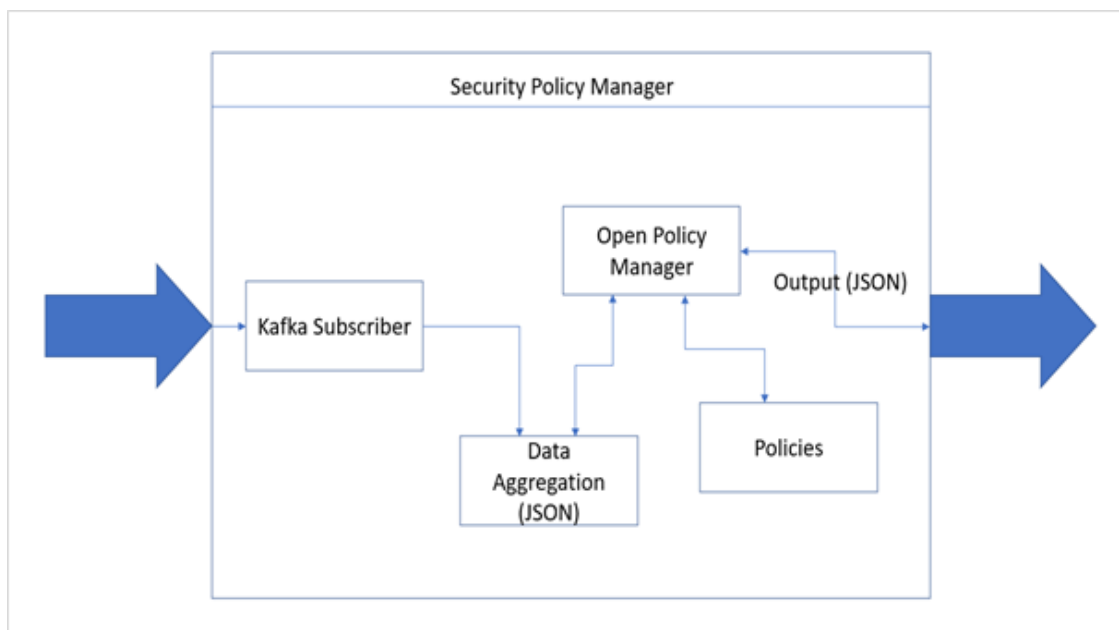


Figure 14 SSPM high level architecture

Figure 14 depicts a scheme of the architecture with the SSPM containing OPA as its main service, a module that acts as a data preparation on the input received that is used during policy evaluation, and the policies that will be evaluated depending on the input received.

Policies can also be created and updated, and once a policy decision has been made, a JSON object is generated and passed as output to the external services.

SSPM supports the logic for multiple kinds 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;

- Cyberattacks identifications;
- Evasion attacks detection.

4.1.1.5.3 Dependencies

- Major libraries – the input is a Kafka library, the output can be a REST API or a Kafka queue. This will be decided according to the necessity to keep history of the outputs.
- Environment - 2vCPU, 4GB RAM, 40GB disk space
- Components - Kafka library, OPA, rule’s storage, management GUI, output (REST API or Kafka)

4.1.1.5.4 Availability

There is a draft version available, a git has been created and in the next phases, a repository description will be provided, including the installation guidelines and the docker installation script. Further instructions will include how to deploy, the main commands and online available documentation.

4.1.1.5.5 Installation/Deployment guidelines

Docker is not already deployed but it will be done at the end of the tool’s development. This has been decided because both the SSPM component and the general STAR platform are still under construction.

4.1.1.5.6 Documentation

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

- API doc (e.g., swagger availability) -> not available
- 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 available yet

4.1.1.5.7 Test Cases

Test ID	SDC-01		
Title	Attack evaluation		
Pre-Requisite	<ul style="list-style-type: none"> • Kafka queue from RMS (through Data Bus) and AI Cyber Defence Module 		
Expected Outcome	<ul style="list-style-type: none"> • REST API or Kafka queue 		
Actions	Expected Result	Result	Comment
SSPM receives inputs from RMS and XAI cyber defence module and can validate the data from data provenance and traceability	Policy manager’s decision on a possible threat according to the	Policy manager’s decision on a possible threat according to the	

components. In the end, SSPM communicates the existence of a threat	policies present in the policies database	policies present in the policies database	
---	---	---	--

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 Pilot #2	Human supervised learning for visual quality inspections	<ul style="list-style-type: none"> Runtime Monitoring System Distributed Ledger Services for Data Reliability AI Cyber-Defence Strategies (ACDS) Risk Assessment and Mitigation Engine (RAME) Security Policies Manager (SPM) - Security Policies Repository (SPR) 	Based on UC2 - Flexibly Quality Control
IBER Pilot #4	Agile Production Management System Data Integrity and Reliability	<ul style="list-style-type: none"> Runtime Monitoring System Distributed Ledger Services for Data Reliability AI Cyber-Defence Strategies (ACDS) Risk Assessment and Mitigation Engine (RAME) Security Policies Manager (SPM) - Security Policies Repository (SPR) 	

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

Currently, the scope of the Simulated Reality component is to assist in the Automated Quality Inspection use-case of the project where it will generate simulated images to balance defect datasets that suffer from skewed class data. Its scope may be enhanced to cover other use-cases such as object recognition for robotic perception through training in simulation and shop-floor resource allocation simulation.

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 will interact 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.

4.2.1.1.3 Dependencies

- Major libraries: The component will be built in python, currently its major dependences include: python >= 3.7, tensorflow >= 2.6.4, keras >= 2.6.0, torch >= 1.11.0
- Environment: The SR 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. Computationally heavy tasks such as GAN training will require access to a GPU.
- Components: One or more batch jobs for training data generators/AI models in a simulated environment. A service (or batch job) to generate data or perform inference.

4.2.1.1.4 Availability

Not available yet. Code will be provided in a public/project specific repository. Docker images could also be stored in an artifact store (e.g. JFrog Artifactory).

4.2.1.1.5 Installation/Deployment guidelines

Not available yet.

4.2.1.1.6 Documentation

Long running batch jobs for ML training can be triggered at specific time intervals (e.g., once a day).

The data serving layer can be exposed through a REST API to which the caller will provide a specification for the synthetic data requested (e.g., number of images per class) and the component will return a location in the data lake where the results are stored, or, if possible, will directly return the results in the request. The details of the API still need to be finalized, as the component is under development.

4.2.1.1.7 Test Cases

The provided test case refers to the component’s current scope (Visual Quality Inspection). Additional test cases will be provided as it extends to other STAR use cases.

Test ID	SR-01
Title	Generation of Synthetic Images for Visual Quality

	Inspection		
Pre-Requisite	<ul style="list-style-type: none"> Access to original training images and potentially required pre-trained models 		
Expected Outcome	<ul style="list-style-type: none"> Generation of images that are visually similar to the originals (as judged through human perception, perceptual loss, FID etc.) 		
Actions	Expected Result	Result	Comment
Another STAR component requests synthetic data fitting a certain specification (e.g., #synthetic images per class)	The requested data is generated and saved in a specific location specified and made accessible to the requesting component		

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 do not provide any performance enhancement to the machine learning model.

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 will interact with the ML Algorithms of the STAR Machine Learning and Analytics Platform, with Human-AI interaction components such as the Natural Language processing module, and the Simulated Reality, to obtain synthetic data.

4.2.1.2.3 Dependencies

Major libraries: The AL component is built in python, currently its major dependencies include: python >= 3.7, modAL²⁸, scikit-learn >= 0.18.

Environment: The AL 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. Computationally heavy tasks may require access to a GPU, but only on training time. The Docker instance will be running some Linux distro.

²⁸ <https://modal-python.readthedocs.io/en/latest/>

Components: One or more batch jobs for training AI models.

4.2.1.2.4 Availability

Not available yet. Code will be provided in a public/project specific repository. Docker images could also be stored in an artifact store (e.g. JFrog Artifactory).

4.2.1.2.5 Installation/Deployment guidelines

No installation guidelines are provided for this deliverable, given the service was not implemented yet.

4.2.1.2.6 Documentation

No documentation was developed up to now. We expect to create a Swagger endpoint, to describe the REST API exposing the service.

4.2.1.2.7 Test Cases

Test ID	AL-01		
Title	Ask whether the data instance should be queried for a label.		
Pre-Requisite	<ul style="list-style-type: none"> The data instance with a specific ID exists. 		
Expected Outcome	<ul style="list-style-type: none"> Boolean value, indicating whether the data instance should be labelled or not. 		
Actions	Expected Result	Result	Comment
The user sends an HTTP GET request to MODEL_REPOSITORY_BASE_URL/active-learning/[strategy]/data/[id]/labels/.	The user receives an HTTP response with a 200 status code and a JSON with a boolean value indicating whether the data instance should be labeled or not.		

4.2.1.3 Production Processes Knowledge Base (PPKB)

4.2.1.3.1 Short Description

The production processes knowledge base module implements means to grow and enrich a knowledge base with meaningful data, to register knowledge known to workers but not documented in the system.

4.2.1.3.2 Relation with the Reference Architecture

PPKB aims to support components in the context of the Trusted Human-AI interactions pillar of the reference architecture. In its current form it will interact with the AL module to understand what piece of data could be valuable to the user, and with Human-AI interaction components such as the Natural Language processing module, and the Feedback module, to present relevant information and obtain inputs from the user.

4.2.1.3.3 Dependencies

Major libraries: The PPKB component is built in python, currently its major dependencies include: python >= 3.7, and owlready2²⁹.

Environment: The PPKBL 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.

Components: It provides batch jobs to perform reasoning, knowledge extraction and knowledge base enrichment. It requires a service to interact with the users based on question-answering, facilitated through a natural language processing interface, to retrieve new knowledge and serve existing knowledge to the users.

4.2.1.3.4 Availability

Not available yet. Code will be provided in a public/project specific repository. Docker images could also be stored in an artifact store (e.g., JFrog Artifactory).

4.2.1.3.5 Installation/Deployment guidelines

No installation guidelines are provided for this deliverable, given only a proof-of-concept service was implemented by now.

4.2.1.3.6 Documentation

No documentation was developed up to now. We expect to create a Swagger endpoint, to describe the REST API exposing the service, and a document describing the user interface exposed to the users.

4.2.1.3.7 Test Cases

Test ID	PPKB-01
Title	Add a new piece of knowledge.
Pre-Requisite	<ul style="list-style-type: none"> No prerequisites are required for this

²⁹ <https://owlready2.readthedocs.io/>

	case.		
Expected Outcome	<ul style="list-style-type: none"> The piece of knowledge is incorporated into the knowledge base. When incorporated, meaningful relationships may be created. 		
Actions	Expected Result	Result	Comment
The user sends an HTTP POST request to <code>MODEL_REPOSITORY_BASE_URL/knowledge-fragments/</code> .	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-02		
Title	Search for a piece of knowledge.		
Pre-Requisite	None.		
Expected Outcome	<ul style="list-style-type: none"> Any piece of knowledge that match the specified criteria are returned. 		
Actions	Expected Result	Result	Comment
<p>The user sends an HTTP GET request to <code>MODEL_REPOSITORY_BASE_URL/knowledge-fragments/</code> that contains the search criteria.</p> <p>The query parameters are specified following the HTTP query syntax.</p>	The user receives an HTTP response with a 200 (OK) status code and a JSON with knowledge fragments that match the specified search criteria in the body.		

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) and Text-to-Speech (TTS) technologies. 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 that we consider may be of interest to understand and give context to the interaction. For that reason, our module also supports sentiment analysis and polarity detection.

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 15) 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.

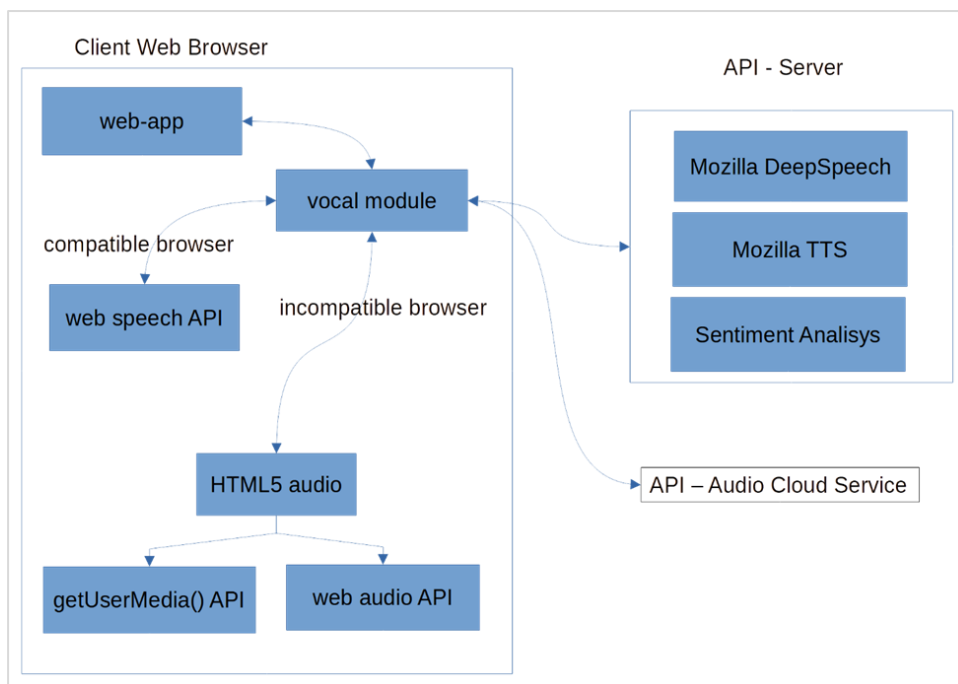


Figure 15 Architecture of the components for Nature Language Processing

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	<ul style="list-style-type: none"> • Microphone enable and not blocked in the browser 		
Expected Outcome	<ul style="list-style-type: none"> • 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"	The user receives the		

button and records the message	converted text.		
--------------------------------	-----------------	--	--

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.		

Test ID	NLP-03		
Title	Test Sentiment Analysis		
Pre-Requisite	A text box with a text		
Expected Outcome	The polarity of the text: Positive, negative, or neutral.		
Actions	Expected Result	Result	Comment
The user writes a text in a text box.	The system analyses the text and provides a classification based on the polarity of the output	Neutral, positive, negative	

4.2.1.5 Feedback Module

4.2.1.5.1 Short Description

The feedback module implements means to obtain relevant information from the user, whether these are labels for the AL module, feedback regarding the knowledge drawn from the PPKB module, feedback regarding values forecasted by the ML models, or any other kind of information relevant to the users.

4.2.1.5.2 Relation with the Reference Architecture

The Feedback module aims to support components in the context of the Trusted Human-AI interactions pillar of the reference architecture. In its current form, it will interact with the AL module, the PPKB module, and any kind of machine learning models developed as part of the STAR Machine Learning and Analytics Platform.

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 Flask.

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 intelligence on what to display, rank, or avoid displaying.

4.2.1.5.4 Availability

Not available yet. Code will be provided in a public/project specific repository. Docker images could also be stored in an artifact store (e.g., JFrog Artifactory).

4.2.1.5.5 Installation/Deployment guidelines

No installation guidelines are provided for this deliverable, given only a proof-of-concept service was implemented by now.

4.2.1.5.6 Documentation

No documentation was developed up to now. We expect to create a Swagger endpoint, to describe the REST API exposing the service.

4.2.1.5.7 Test Cases

Test ID	FM-01		
Title	Get a feedback request.		
Pre-Requisite	<ul style="list-style-type: none"> The feedback entity with the specified ID exists. 		
Expected Outcome	<ul style="list-style-type: none"> The feedback entity with the specified ID is displayed. 		
Actions	Expected Result	Result	Comment
The user sends an HTTP GET request to MODEL_REPOSITORY_BASE_URL/feedbacks/ [id].	The user receives an HTTP response with a 200 status code and a JSON body with details regarding the feedback entity.		

Test ID	FM-02		
Title	Add feedback outcome result.		
Pre-Requisite	A feedback entity with a given ID exists.		
Expected Outcome	<ul style="list-style-type: none"> The feedback outcome is persisted in the database, and linked to the corresponding feedback entity. 		
Actions	Expected Result	Result	Comment
The user sends an HTTP POST request to MODEL_REPOSITORY_BASE_URL/feedbacks/[id] responses.	The user receives an HTTP response with a 200 (OK) status code and the feedback outcome is persisted in the database and associated with the corresponding feedback.		

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
PCL Pilot #2	Human supervised learning for visual quality inspections	<ul style="list-style-type: none"> Simulated Reality (SR) Active Learning (AL) Production Processes Knowledge Base (PPKB) Feedback Module XAI Models and Library Natural Language Processing (NLP) 	Plan to use AL / FM to setup automated quality inspections quickly and easily within the Philips pilot. PHILIPS UC3 for fatigue monitoring Depending on environmental noise
IBER Pilot #3	Employee Training for Reduction of Human Errors	<ul style="list-style-type: none"> Simulated Reality (SR) Feedback Module Natural Language Processing (NLP) 	IBER UC3 for fatigue monitoring Depending on environmental noise

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 subsystem

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

The first component, described in D5.5, 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 D5.7, will access 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, will be integrated into the HDT system described in D5.1 and presented in the figure below (Figure 1). To support the Privacy by Design principle, the Safety Zones Detection System will publish 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 will access to the results of this module and consequently, they will adapt their behaviour.

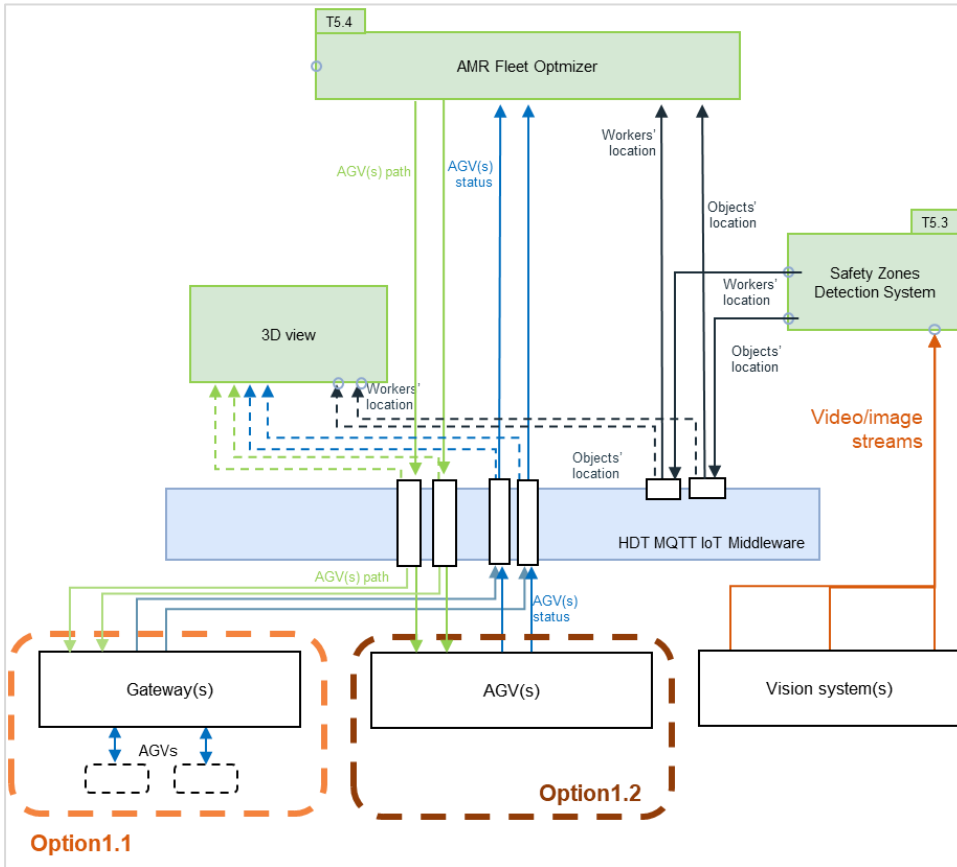


Figure 16 Safety Zone Detection & AMR Fleet Optimizer physical view

4.3.1.1.3 Dependencies

The Safety Zones Detection is composed of several Docker images and a Docker compose file will be 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: StructBased</p> <ul style="list-style-type: none"> – batteryStatus: NumberBased – unitOfMeasure: StringBased – factoryEntityID: StringBased <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: StructBased</p> <ul style="list-style-type: none"> – currentX: NumberBased – currentY: NumberBased – coordinateSystem: StringBased – factoryEntityID: StringBased <p>Message example</p> <p>TOPIC: HDT/<u>UUID1</u>/state/UUID4</p> <p>MESSAGE: 1652965322501#{"currentX": 12.3, "currentY": 27.5, "coordinateSystem":"ABC", "factoryEntityID": "UUID1"}</p>	AMR (Robotino)	AMR Fleet Optimizer
current speed (if may be sent together with	<p>State descriptor (UUID5) structure: StructBased</p>	AMR (Robotino)	AMR Fleet

<p>current position)</p>	<ul style="list-style-type: none"> – currentSpeed: NumberBased – unitOfMeasure: StringBased – factoryEntityID: StringBased <p>Message example</p> <p>TOPIC: HDT/UUID1/state/UUID5</p> <p>1652965322501#{ "currentSpeed": 2.9 "unitOfMeausure": "km/h", "factoryEntityID": "UUID0"}</p>	<p>)</p>	<p>Optimizer</p>
<p>current orientation (optional)</p>	<p>State descriptor (UUID6) structure: StructBased</p> <ul style="list-style-type: none"> – currentOrientation: NumberBased – factoryEntityID: StringBased <p>Message example</p> <p>TOPIC: HDT/UUID1/state/UUID6</p> <p>1652965322501#{ "currentOrientation": 12.4, "factoryEntityID": "UUID1"}</p>	<p>AMR (Robotino)</p>	<p>AMR Fleet Optimizer</p>
<p>GlobalOccupancy (a.k.a. heatmap)</p>	<p>State descriptor (UUID10) structure: StructBased</p> <ul style="list-style-type: none"> – GlobalOccupancy: ListBased – HumanOccupancy: ListBased <ul style="list-style-type: none"> – StructBased <ul style="list-style-type: none"> – cellIndex : NumberBased – occupancy : NumberBased – ObstacleOccupancy: StructBased <ul style="list-style-type: none"> – Type: StringBased – ListBased <ul style="list-style-type: none"> – cellIndex : NumberBased 	<p>Video Analytics (a.k.a. SafetyZone Detection)</p>	<p>AMR Fleet Optimizer</p>

	<p>Message example</p> <p>TOPIC: HDT/UUID2/state/UUID10</p> <pre>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"}</pre>		
--	--	--	--

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.

nextWaypoint	<p>State descriptor (UUID7) structure:</p> <p>StructBased</p> <ul style="list-style-type: none"> – nextX: NumberBased – nextY: NumberBased – factoryEntityID: StringBased <p>Message example</p> <p>TOPIC: HDT/UUID2/state/UUID7</p> <pre>1652965322501#{"nextX": 12.2, "nextY": 12.3, "factoryEntityID": "UUID2"}</pre>	AMR Fleet Optimizer	AMR (Robotino)
nextPath	<p>State descriptor (UUID8) structure:</p> <p>StructBased</p> <ul style="list-style-type: none"> – nextPath: ListBased 	AMR Fleet Optimizer	AMR (Robotino)

	<ul style="list-style-type: none"> – StructBased – x: NumberBased – y: NumberBased – factoryEntityID: StringBased <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>		
nextAction	<p>State descriptor (UUID9) structure:</p> <p>StructBased</p> <ul style="list-style-type: none"> – action: StringBased – factoryEntityID: StringBased <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>	AMR Fleet Optimizer	AMR (Robotino)

4.3.1.1.7 Test Cases

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

Actions	Expected Result	Result	Comment
Operator inspects the module(s)	<ul style="list-style-type: none"> Humans are detected People are well positioned in 3D real scene 	<p>Promising first tests on DFKI video</p> <p>Not yet executed in real-time DFKI factory</p>	

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"> Background model or a few minutes of the empty scene RTSP Cameras Cameras calibrations and global map of the factory 		
Expected Outcome	<ul style="list-style-type: none"> Moving object is well positioned 		
Actions	Expected Result	Result	Comment
Operator inspects the module(s)	<ul style="list-style-type: none"> Moving objects are detected Moving objects are well positioned in 3D real scene 	<p>Promising first tests on DFKI video</p> <p>Not yet executed in real-time DFKI factory</p>	

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"> Pretrained neural network parameters for object detection and classification Background model or a few minutes of the empty scene RTSP Cameras Cameras calibrations and global map of the factory 		
Expected Outcome	<ul style="list-style-type: none"> Moving object is well positioned 		
Actions	Expected Result	Result	Comment
Operator inspects	<ul style="list-style-type: none"> Objects of interest are detected and are well 	Not yet executed.	

the module(s)	positioned in 3D real scene	In progress	
---------------	-----------------------------	-------------	--

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

Test ID	AMR-06		
Title	AMR Fleet Optimization		
Pre-Requisite	<ul style="list-style-type: none"> • Empty map of the environment • Obstacle presence • Factory occupancy (a.k.a. heatmaps) • Robots status and positions • List of logistic tasks to be performed 		
Expected Outcome	<ul style="list-style-type: none"> • Each robot is attributed a task in such a way that all tasks are processed in an efficient manner 		
Actions	Expected Result	Result	Comment
Operator gives a list of logistic tasks	<ul style="list-style-type: none"> • Movements of all the fleet AMRs will be coherent in doing all the tasks allocated to the fleet (All tasks will be handled in a decent time with regards to the number of available AMR and the size of the list of tasks) 	Not yet executed. In progress	Simulation capabilities will allow to test in a wider scope than just the real environment

Test ID	AMR-07		
Title	AMR Fleet Commands to AMRs		
Pre-Requisite	<ul style="list-style-type: none"> • Empty map of the environment • Obstacle presence • Factory occupancy (a.k.a. heatmaps) • Robots status and positions • List of logistic tasks to be performed • AMR availability 		
Expected Outcome	<ul style="list-style-type: none"> • Each robot moves following paths displayed in the virtual environment 		
Actions	Expected Result	Result	Comment
Operator gives a list of logistic tasks	<ul style="list-style-type: none"> • Coherent AMR movements 	Not yet executed.	

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 17. The installation/deployment guidelines are available in the [project repository](#).

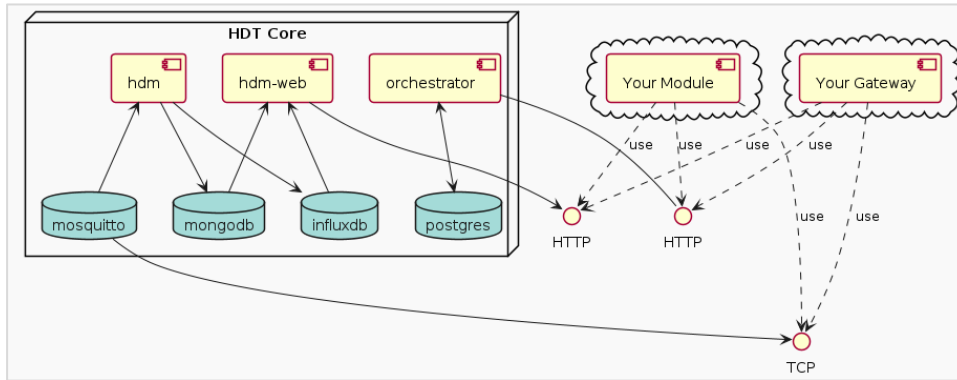


Figure 17 Deployment view of HDT components

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	<ul style="list-style-type: none"> The <i>orchestrator</i> service is up and running 		
Expected Outcome	<ul style="list-style-type: none"> The <i>orchestrator</i> service registers the new functional module. 		
Actions	Expected Result	Result	Comment
The client sends an HTTP POST request to ORCHESTRATOR_BASE_URL/api/v1/functional-module , passing a valid FunctionalModuleDto in the body.	The user receives an HTTP response with a 200 status code and a body containing a ResponseFunctionalModuleDto.	Test successful	DTOs are described in Swagger at ORCHESTRATOR_BASE_URL/swagger-ui/index.html

Test ID	HDT-02		
Title	Activate a functional module, by ID		
Pre-Requisite	<ul style="list-style-type: none"> The functional module has been already registered in the HDT. 		

Expected Outcome	<ul style="list-style-type: none"> 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 MODEL_REPOSITORY_BASE_URL /api/v1/functional-module/activate/functional-module/{functionalModuleId}.	The user receives an HTTP response with a 200 status code. The response body contains a ResponseFunctionalModuleActivationResponseDto, 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 ORCHESTRATOR_BASE_URL /swagger-ui/index.html

Test ID	HDT – 03 (FaMS - 01)		
Title	Communication between the HDT Core Infrastructure and the FaMS		
Pre-Requisite	<ul style="list-style-type: none"> HDT and FaMS configured, deployed and functioning. Gateway and wearables are active. 		
Expected Outcome	<ul style="list-style-type: none"> 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"> HDT Core Infr. deployed and functioning Robotino’s agent deployed and functioning. 		
Expected Outcome	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HDT Core Infr., Safety Zones Detection System and AMR Fleet Optimizer configured, deployed and functioning. 		
Expected Outcome	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> HDT and FaMS configured, deployed and functioning; Gateway and wearables active.
Expected Outcome	<ul style="list-style-type: none"> 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 will connect 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
 - 1st column: timestamp
 - 2-4th column: ACC 3-channels from left wrist sensor
 - 5-7th column: Gyroscope 3ch data from left wrist sensor
 - 8-10th column: Magnetometer 3ch data from left wrist sensor
 - 11th column: Capacitive sensor data from left wrist sensor
 - 12-14th column: ACC 3ch data from right wrist sensor
 - 15-17th column: Gyroscope 3ch data from right wrist sensor
 - 18-20th column: Magnetometer 3ch data from right wrist sensor
 - 21st 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"> • Pretrained neural network parameters for human activity recognition • Attached wearable sensors on worker’s body 		
Expected Outcome	<ul style="list-style-type: none"> • Human activity is classified 		
Actions	Expected Result	Result	Comment
Human inspects the module(s)	<ul style="list-style-type: none"> • The current activity is detected • Precision of the activity detection is returned • Possible next actions with their accuracies are returned 	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"> Human Centred Digital Twin Fatigue Monitoring System 	The use-case aims to detect and monitor workers' fatigue when performing manual labeling of images. In particular, we tackle the visual inspection use case (Philips UC2). Detecting workers' fatigue, attention, and/or mental stress during the labeling process can be helpful at least in two ways: understand whether the labeled 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"> Human Centred Digital Twin Workers Activity Recognition 	Plan to detect the human activities and predict their next actions. For this matter, DFKI created typical worker scenarios, happening during normal daily work.
DFKI Pilot #2	Robot Reconfiguration Based on the Dynamic Layout.	<ul style="list-style-type: none"> Human Centred Digital Twin AMR Safety 	Plan to dynamically update the navigation route of the mobile robot, by considering human and/or other (non-) moving objects in the environment by the ceiling cameras installed in the testbed. This use case will also enable easier reconfiguration of the robot in case the layout of the environment (including the production stations) changes.
DFKI Pilot #3	Dynamic Path Planning Using Both First and Second Use Cases	<ul style="list-style-type: none"> AMR Safety Human Centred Digital Twin Workers Activity Recognition 	The two use cases for DFKI Pilot #1 and #2 are going to be combined to have a safe environment for the workers and the hardware equipment.
IBER Pilot #3	Employee Training for Reduction of Human Errors	<ul style="list-style-type: none"> Fatigue Monitoring System 	

5 Testing, Validation, and Integration Roadmap

5.1 Lab Requirements, and Environment

In this deliverable as mentioned before, the focus is to provide a wide range of knowledge about the testing and validation, and integration of the STAR platform. To test, validate, and integrate different services and modules from other technology providers, DFKI is supposed to provide a server that can host and run all the components from WP3, WP4, and WP5 module providers. Access to the server could be provided for all the component owners. For this reason, we start with the list of assets and their minimum requirements to be run fluently on the STAR server. The list contains different fields explained below:

Component: The list of components that is related to the T6.2 and need to be validated, tested, and integrate into the DFKI server which is provided for the STAR project.

Component Owner: The technology provider(s) which provide(s) the mentioned modules/technologies.

Related Task: The work package’s task is related to the technology/component.

Dependency with other components (Input/output): If the component/ technology is related to other component(s)/technology(is) from other task(s). The relation could be providing the input for other technologies or utilizing the output from other technologies.

Component Version: The version of the currently available component.

Communication protocol: The protocol available to connect to the technology(is) (e.g., Kafka, TCP-IP, MQTT)

Availability (Code, Artifact): If there is code or artifact available from the asset(s).

Dockerization: if the code/ artifact is possible to be dockerized. It is essential to be able to utilize the docker service packaging the delivered software in the containers.

Hardware Requirements: The hardware requirements which is suitable to be used for the component/asset (e.g., CPU, GPU, HDD, Memory).

Possible Lab deployment date: The date that the deployment is possible.

Pilots to be applied (except the Lab environment): On which pilot and use case the component/asset is planned to be applied.

Status of the Components: If it is developed or is under development.

API Documentation: If there is any documentation or link to documentation available for the component/asset.

Test Availability (Yes/No): If there is any automated/code test available for the component/asset.

Link to Source code: The link to the code, if there is any open-source code available for the component/asset.

DFKI as the leader of the T6.2 is supposed to provide the server for the integrated platform. At the time of writing this deliverable, we are gathering the list of assets and related requirements. For the next phase, we will initiate a server and VM for each of the technology providers from WP3, WP4, and WP6. 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

Although for the T6.2 "Service Platform Integration and Lab Validation" we have both inter and between WPs' components validation, for this deliverable (D6.3), We just focused on the inter-WP components. For this specific task, we consider some supported scenarios that can be applied for the next step of this specific task.

The following, there are some of the potential scenarios explained.

Automation Tools:

For the ease of validation, testing, and integration, we start with the manual setup. During the update in every module/asset, the technology providers will test their artifacts manually to fulfil the task requirement. If there are any Automation CI/CD tools are needed we can utilize an integrated test service (e.g., Jenkin, GitLab CI/CD services)

Validation the Components:

For the validation of the components based on the KPIs, we start with simple scenarios. Each WP will fulfil the validation phase.

Evaluate the accuracy of the architectures:

The validation of each component is dedicated to component providers. They will validate their artifacts/components each time they update them.

The Process of Accessing the Data:

To access the data which is planned to be the input for the components/software from technology providers we planned to have saved samples for each data source for each scenario and then test the artifact on them. Maybe for the next action, we need to make a communication to the HWs which are providing the data for the artifacts (e.g., Cameras, sensors, actuators).

6 Conclusion

The STAR project consists of a wide range of functionalities and technologies for research and the industrial environment. The major target of the STAR project is to research, implement validate and demonstrate the trusted AI technologies that can be utilized in the production line in different scenarios. In this deliverable, we have discussed the activities related to testing, validation, and integration of the components into the STAR platform. The points and themes to be discussed are:

- The integration platform and its relation to the STAR Reference Architecture: We explained the reference architecture and the WP-level deployment/physical diagram to be used for lab validation.
- List of components: We provide the list of all target components from WP3, WP4, and WP5 that are utilized in T6.2. These components are the technology(es)/software(s) that technology providers test, evaluate, and integrate into the STAR platform. For each component, we have a short description, the relation to the STAR reference architecture, dependency of the components to other packages or components, installation/deployment guidelines, documentation, and test case. The test case is provided in separate tables for each component and presents the actions and expected results for the atomic inter-component and WP level communication.
- Test, validation, and integration roadmap: We afford the section about the preparation of the lab environment, and the components' requirements. This section is providing the steps that we are planning to do in the future for this task.

References

Reference	Name of document
[Kisller21]	E. Kisller, "What Is a Container Registry? And Why Do I Need One?," 26 March 2021. [Online]. Available: https://jfrog.com/knowledge-base/what-is-a-container-registry/ . [Accessed August 2021]
[Souza 18]	H. Souza, "How to dockerize any application", May 2018, available at: https://hackernoon.com/how-to-dockerize-any-application-b60ad00e76da , 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-D3.5]	STAR, "D3.5 – Security and Data Governance Infrastructure-Initial version", 2022-06-14

APPENDIX A

```

STAR AI Cyber Defense tool Docker compose file
version: '2.1'

services:
  zoo1:
    image: confluentinc/cp-zookeeper:7.1.1
    hostname: zoo1
    container_name: zoo1
    ports:
      - "2181:2181"
    networks:
      star_network:
        ipv4_address: 10.10.10.2
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_SERVER_ID: 1
      ZOOKEEPER_SERVERS: zoo1:2888:3888

  Kafka1:
    image: confluentinc/cp-Kafka:7.1.1
    hostname: Kafka1
    container_name: Kafka1
    ports:
      - "9092:9092"
      - "9999:9999"
    networks:
      star_network:
        ipv4_address: 10.10.10.3
    environment:
      KAFKA_ADVERTISED_LISTENERS:
LISTENER_DOCKER_INTERNAL://Kafka1:19092,LISTENER_DOCKER_EXTERNAL://{DOCKER_HOST_IP:-
127.0.0.1}:9092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_ZOOKEEPER_CONNECT: "zoo1:2181"
      KAFKA_BROKER_ID: 1
      KAFKA_LOG4J_LOGGERS:
"Kafka.controller=INFO,Kafka.producer.async.DefaultEventHandler=INFO,state.change.logger=INFO"
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_JMX_PORT: 9999
      KAFKA_JMX_HOSTNAME: ${DOCKER_HOST_IP:-127.0.0.1}
      KAFKA_AUTHORIZER_CLASS_NAME: Kafka.security.authorizer.AclAuthorizer
      KAFKA_ALLOW_EVERYONE_IF_NO_ACL_FOUND: "true"
      KAFKA_CFG_MAX_REQUEST_SIZE: 52428800
      KAFKA_MESSAGE_MAX_BYTES: 52428800
      KAFKA_SOCKET_REQUEST_MAX_BYTES: 52428800
    depends_on:
      - zoo1

```

```

producer:
  build: .
  command: python3 serverProducer.py
  ports:
    - "8080:8080"
  networks:
    star_network:
      ipv4_address: 10.10.10.4
  volumes:
    - ./star

consumer:
  build: consumer/
  command: bash -c "sleep 45; python3 serverConsumer.py"
  networks:
    star_network:
      ipv4_address: 10.10.10.5
  volumes:
    - ./star
  depends_on:
    - Kafka1

networks:
  star_network:
    driver: bridge
  ipam:
    config:
      - subnet: 10.10.10.0/24
      gateway: 10.10.10.1
  
```

```

STAR Risk Assessment and Mitigation Engine Docker compose file
version: '3.3'

services:

  postgres:
    container_name: "${PROJECT_NAME}_postgres"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    env_file:
      - .env
    environment:
      - POSTGRES_USER=${DATABASE_USERNAME}
      - POSTGRES_PASSWORD=${DATABASE_PASSWORD}
      - POSTGRES_DB=${DATABASE_NAME}
    ports:
      - ${DATABASE_PORT}:5432
    image: 'postgres:13.1'
    restart: unless-stopped

  pgadmin:
  
```

```

container_name: "${PROJECT_NAME}_pgadmin"
ports:
  - ${PGADMIN_PORT}:5454
env_file:
  - .env
environment:
  - PGADMIN_DEFAULT_EMAIL=${PGADMIN_DEFAULT_EMAIL}
  - PGADMIN_DEFAULT_PASSWORD=${PGADMIN_DEFAULT_PASSWORD}
  - PGADMIN_LISTEN_PORT=${PGADMIN_PORT}
image: 'dpage/pgadmin4:6.1'
depends_on:
  - postgres
links:
  - 'postgres:pgsql-server'
restart: unless-stopped

postgreskeycloak:
container_name: "${PROJECT_NAME}_keycloak_postgres"
ulimits:
  memlock:
    soft: -1
    hard: -1
env_file:
  - .env
# volumes:
#   - ./profiles/local/01-schema.sql:/docker-entrypoint-initdb.d/01-schema.sql
environment:
  - POSTGRES_USER=${KEYCLOAK_DATABASE_USERNAME}
  - POSTGRES_PASSWORD=${KEYCLOAK_DATABASE_PASSWORD}
  - POSTGRES_DB=${KEYCLOAK_DATABASE_NAME}
ports:
  - ${KEYCLOAK_DATABASE_PORT}:5432
image: 'postgres:13.1'
restart: unless-stopped

keycloak:
container_name: "${PROJECT_NAME}_keycloak"
image: 'quay.io/keycloak/keycloak:15.0.2'
environment:
  DB_VENDOR: POSTGRES
  DB_ADDR: postgreskeycloak
  DB_DATABASE: ${KEYCLOAK_DATABASE_NAME}
  DB_USER: ${KEYCLOAK_DATABASE_USERNAME}
  DB_PASSWORD: ${KEYCLOAK_DATABASE_PASSWORD}
  KEYCLOAK_USER: ${KEYCLOAK_ADMIN_USERNAME}
  KEYCLOAK_PASSWORD: ${KEYCLOAK_ADMIN_PASSWORD}
  KEYCLOAK_IMPORT: /tmp/realm.json
volumes:
  - ./realm/realm.json:/tmp/realm.json
ports:
  - ${KEYCLOAK_PORT}:8080
depends_on:
  - postgreskeycloak
restart: unless-stopped

```

```

jaeger:
  container_name: "${PROJECT_NAME}_jaeger"
  profiles: ["monitoring"]
  ports:
    - '5775:5775'
    - '6831:6831'
    - '6832:6832'
    - '5778:5778'
    - '16686:16686'
    - ${JAEGER_PORT}:14268
  image: 'jaegertracing/all-in-one:1.22'
  restart: unless-stopped

es01:
  image: 'docker.elastic.co/elasticsearch/elasticsearch:7.8.1'
  container_name: "${PROJECT_NAME}_es01"
  # restart: unless-stopped
  environment:
    - node.name=${PROJECT_NAME}_es01
    - cluster.name=${PROJECT_NAME}_es-cluster
    # - discovery.seed_hosts=es02
    #- cluster.initial_master_nodes=${PROJECT_NAME}_es01
    - bootstrap.memory_lock=true
    - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    - discovery.type=single-node
  logging:
    options:
      max-size: "10MB"
      max-file: "10"
  ulimits:
    memlock:
      soft: -1
      hard: -1
    nofile:
      soft: 65536
      hard: 65536
  cap_add:
    - IPC_LOCK
  # volumes:
  # - data01:/usr/share/elasticsearch/data
  ports:
    - ${ELASTIC_PORT}:9200
  networks:
    elastic:
      aliases:
        - ${PROJECT_NAME}_es01
        - elasticsearch

kib01:
  image: 'docker.elastic.co/kibana/kibana:7.8.1'
  container_name: "${PROJECT_NAME}_kib01"
  ports:
    - ${KIBANA_PORT}:5601
  environment:
    ELASTICSEARCH_URL: http://${PROJECT_NAME}_es01:${ELASTIC_PORT}

```

```

ELASTICSEARCH_HOSTS: http://${PROJECT_NAME}_es01:${ELASTIC_PORT}
networks:
- elastic
depends_on:
- es01
restart: unless-stopped

logstash:
image: 'docker.elastic.co/logstash/logstash:7.8.1'
container_name: "${PROJECT_NAME}_logstash"
restart: unless-stopped
profiles: ["monitoring"]
volumes:
- source: ./pipelines
  target: /usr/share/logstash/pipeline
  type: bind
ports:
- ${LOGSTASH_PORT}:12201/udp
- "5000:5000"
- "9600:9600"
networks:
- elastic
depends_on:
- es01

minio:
image: 'minio/minio:RELEASE.2021-06-09T18-51-39Z'
container_name: "${PROJECT_NAME}_minio"
volumes:
- miniodata1:/data1
# - data1-2:/data2
ports:
- ${MINIO_PORT}:9000
environment:
MINIO_ROOT_USER: ${MINIO_ROOT_USER}
MINIO_ROOT_PASSWORD: ${MINIO_ROOT_PASSWORD}
MINIO_ACCESS_KEY: ${MINIO_ROOT_USER}
MINIO_SECRET_KEY: ${MINIO_ROOT_PASSWORD}
command: server /data1
# command: server http://minio{1...3}/data{1...2}
# command: server http://minio{1...3}/data{1...2} http://minio{4...5}/data{1...4}
healthcheck:
test: ["CMD", "curl", "-f", "http://localhost:${MINIO_PORT}/minio/health/live"]
interval: 30s
timeout: 20s
retries: 3
networks:
- minionw
restart: unless-stopped

mongo:
image: "mongo:5.0.6"
ports:
- ${MONGO_PORT}:27017
# volumes:

```

```
# - /media/vfs/olistic/mongodb:/data/db  
restart: unless-stopped
```

```
volumes:  
  miniodata1:  
    name: ${PROJECT_NAME}_minio
```

```
networks:  
  elastic:  
    driver: bridge  
  minionw:  
    driver: bridge
```