Ref. Ares(2022)2819875 - 08/04/2022

IntellioT

Deliverable D5.2
Deployed & tested use case demonstrators (first version)

Deliverable release date	31.03.2022
Authors	SIEMENS: Andreas Zirkler, Andreas Ziller, Arne Broering, Vivek Kulkarni EURECOM: Jérôme Härri AAU: Beatriz Soret, Lam Nguyen UOULU: Sumudu Samarakoon, Mohamed Abdel-Aziz TTC: Martijn Rooker TSI: Andreas Brokalakis, Vassilis Amourgianos, Babis Savvakos PHILIPS: Nancy Irisarri Mendez, Anca Bucur SANL: Konstantinos Fysarakis, Ioannis Vezakis HSG: Simon Mayer, Jérémy Lemée HOLO: Carina Pamminger, Nour Fendri AVL: Holger Burkhardt, Wolfgang Hollerweger
Editor	Andreas Zirkler (SIEMENS)
Reviewer	Nour Fendri (HOLO), Martijn Rooker (TTC)
Approved by	PTC Members: (Vivek Kulkarni, Konstantinos Fysarakis, Sumudu Samarakoon, Beatriz Soret, Arne Bröring, Kristin Bauermeister) PCC Members: (Vivek Kulkarni, Jérôme Härri, Beatriz Soret, Mehdi Bennis, Martijn Rooker, Sotiris Ioannidis, Anca Bucur, Georgios Spanoudakis, Simon Mayer, Filippo Leddi, Harshitha Chandregowda, Maren Lesche, Fragkiskos Parthenakis)
Status of the Document	Final
Version	1.0
Dissemination level	Public

Document history 4
Acronyms and definitions 5
1 Introduction 7
2 Cross-use-case Integration tests 8
2.1 Web-based EUP IDE for HyperMAS and HyperMAS Infrastructure 8
2.2 5G Infrastructure 9
2.3 Distributed Ledger Technologies 9
2.4 Trust components 16
2.4.1 Security Assurance Platform 17
2.4.2 Trust IDs and MTD 23
2.4.3 Authentication, Authorization and Accounting (AAA) 24
3 UC1: Agriculture 25
3.1 Hardware deployment 25
3.1.1 Tractor Infrastructure 25
3.1.2 5G Infrastructure 27
3.1.3 VR Infrastructure 28
3.2 Software / Apps deployment 29
3.3 Integration Tests 30
3.3.1 Integration and Tests of Tractor Controller and Tractor 30
3.3.2 Integration and Tests of HyperMAS and Tractor 31
3.3.3 Integration and Tests of HIL Service 32
3.3.4 Integration and Test of HIL application 32
3.3.5 Integration and Test of AI and Tractor 32
3.3.6 Integration and Test of DLT and Tractor 33
3.3.7 Integration and Test of 5G Infrastructure 35
3.3.8 Integration and test of trust components 36
4 UC2: Healthcare 38
4.1 Hardware deployment 38
4.1.1 Smart Devices 38
4.2 Software / Apps deployment 39
4.3 Integration Tests 40
4.3.1 Integration and Tests of Global AI AND Local AI Components 40
4.3.2 Integration and Tests of Global AI and iakm 45
4.3.3 Integration and Tests of HyperMAS and HMI/OTHER 45
4.3.4 Integration and Tests of Trust components 46
5 UC3: Manufacturing 47

5.1 Hardware deployment 47
5.1.1 Edge Infrastructure 48
5.1.2 TSN Infrastructure 51
5.1.3 5G Infrastructure 52
5.1.4 AR Infrastructure 53
5.1.5 Robot 54
5.1.6 Laser Engraver machine 54
5.1.7 Milling machine 55
5.1.8 Workpiece storage 56
5.2 Software / Apps deployment 58
5.3 Integration Tests 59
5.3.1 Integration and Test of HyperMAS and Engraver machine 59
5.3.2 Integration and Test of HIL service 59
5.3.3 Integration and Test of HIL application 61
5.3.4 Integration and Test of Robot controller and AI 62
5.3.5 Integration and Test of TSN controller 62
5.3.6 Integration and Test of Communication resource manager 62
5.3.7 integration and Test of DLT 63
5.3.8 Integration and Test of SAP 66
6 Conclusion and future work 67

DOCUMENT HISTORY

Date	Author	Main Activities
27.10.2021	Andreas Zirkler	First draft, table of contents
12.01.2022	Andreas Zirkler	Document framework ready to be filled by partners
14.03.2022	Andreas Zirkler	Draft version for review
30.03.2022	Andreas Zirkler	Final version for final review

ACRONYMS AND DEFINITIONS

Acronym	Definition
5G NR	5th Generation New Radio
AAA	Authentication, Authorization, Accounting
AGV	Automated Guided Vehicle
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
BDI	Belief-Desire-Intention
BMS	Battery Management System
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CAN	Controller Area Network
CNC	Computerized Numerical Control
COVID-19	Corona Virus Disease of 2019
D2D	Device to Device
DLT	Distributed Ledger Technology
ECG	Electrocardiogram
EMS	Edge Management System
EUP IDE	End-User Programming Integrated Development Environment
GDPR	General Data Protection Regulation
GPS	Global Positioning System
HMI	Human Machine Interface
HyperMAS	Hypermedia Multi-agent System
IAKM	Infrastructure Assisted Knowledge Management
IED	Industrial Edge Device
IEM	Industrial Edge Manager
ID	Identification
IDS	Intrusion Detection System
IMU	Inertia Measurement Unit
IO	Input Output
IoT	Internet of Things
IPFS	Inter Planetary File System

IPSec	Internet Protocol Security
IT/OT	Information Technology/Operation Technology
JSON	JavaScript Object Notation
MES	Manufacturing Execution System
MTD	Moving Target Defences
NGIoT	Next Generation Internet of Things
OBC	On-board Charger
OEM	Original Equipment Manufacturer
PLC	Programmable Logic Controller
PROFInet	Process Field Network
QoS	Quality of Service
TLS	Transport Layer Security
TSN	Time Sensitive Networking
UC	Use Case
UE	User Equipment
UML	Unified Modeling Language
URLL	Ultra-Reliable and Low Latency
VDMA	Verband Deutscher Maschinen- und Anlagenbau
VPN	Virtual Private Network
VR	Virtual Reality
W3C WoT TD	World Wide Web Consortium Web of Things Things Description
WiFi	Wireless Fidelity
WP	Work Package
ZVEI	Zentralverband Elektrotechnik- und Elektronikindustrie

1 INTRODUCTION

This deliverable is the first version of Task T5.2 narrates deployment, testing and demonstration of the IntellIoT framework components and use case applications in the respective demo environments. Based on this deployment, the project members will conduct the demonstrations for the 3 use cases in real-life settings by incorporating, e.g., an actual tractor, real patients, and actual manufacturing devices and machinery.

The deployment of the integrated IntellIoT framework and implemented use case applications will be on top of a computation & communication infrastructure (specific for each use case) according to the requirements and use case specifications defined in Tasks 2.1 and 2.2. The IntellIoT mechanisms for infrastructure management (WP4) will facilitate the configuration of connectivity, scalability, and reliability. They will guarantee quality for the proposed real-life experimental activities of the use cases. The testing part of this task deals with the analysis of the correct behavior of each component (a) of the technical framework (WP3 and WP4) and (b) of the use case implementations (Task 5.1). The functionality of these components will be tested in context of the realization of the 3 use cases and their demonstration.

Task 5.2 delivers deliverables D5.2 and D5.5. These describes the integration and deployment, as well as the testing of the individual components in the different use cases, comparable to the "+1 view" described in D2.3, while D5.2 focusses on cycle 1, D5.5 adds learnings and additions from cycle 2. These deliverables build upon D5.1, which describes the development of components in silos.

2 CROSS-USE-CASE INTEGRATION TESTS

Deployment and demonstration of technologies and components developed in IntellIoT is done in use-case specific setups. Where possible, the use case contributors are using the same setup for integration tests and demonstration. Even being use-case specific, the setups naturally comprise cross-use case platform components, which will be used in several setups.

The setups are building up on components from work packages 3 and 4. All components have been pre-tested, i.e., they have been to component tests in component specific test setups, which have been performed by the responsible partners.

In this deliverable, the step-by-step integration and test of the various components is briefly described, where chapter 2 focusses on cross-use case components, chapters 3 to 5 on the three use cases. Issues are tracked and exchanged between partners using a GitLab repository hosted at EURECOM (https://gitlab.eurecom.fr/intelliot-project). The same repository also contains interface descriptions, test scripts etc. required for the tests.

For the test steps, the use case scenarios defined in D2.1 are followed. Selected test scenarios are used for demonstration. Please note that not all scenarios will be covered in this cycle 1 document.

As described in IntellIoT's architectural documents, interfaces between components from different partners are predominantly REST-based, which allows for clear and easily exchangeable interface descriptions in YAML or JSON. However, there will be exceptions, e.g., for time critical interfaces like robot control or interfaces transporting huge amounts of data like the cameras used in use cases 1 and 3.

This chapter will describe cross-use-case integration tests for components which are commonly used in multiple use cases. Thanks to IntellIoT's platform-oriented architecture, various components are used in multiple use cases, however they are applied in different use-case domains, which requires domain specific integration tests in some cases. Even for the cross-use-case integration, interfacing components from one or more use cases might be used as test bed. When these pre-tests are successfully performed, the cross-UC components will be used as test references for interfacing components in use-case specific tests, which will be described in chapters 3, 4 and 5.

2.1 Web-based EUP IDE for HyperMAS and HyperMAS Infrastructure

These two components, i.e. the web-based EUP IDE and the HyperMAS Infrastructure, (see Deliverable D3.1 for details) have been developed in an integrated approach from the beginning. At the moment, the HyperMAS Infrastructure is providing services for registering and retrieving W3C WoT Thing Descriptions (TDs) across workspaces ("uc1-agriculture", "uc2-healthcare", "uc3-manufacturing") within its "intelliot" environment. These TDs are retrieved by the Web-based End-user Programming IDE for HyperMAS (EUP IDE) and provided to agent programmers as building blocks. Subsequently, generated Agent programs can be run in a MAS that is in the process of being migrated to the HyperMAS Infrastructure as well. One research article that describes this process is currently being finalized and we plan to submit it in March 2022.

We have performed tests of these components with all TD-described components that we have at our disposal, in particular with those components from our laboratory (i.e., two small mobile robots towards UC1, a messaging service towards UC2, and a robotic arm towards UC3, as well as with building automation equipment such as light switches and blinds controllers). The tests aimed at verifying the successful registration of artifacts with the infrastructure using their TDs, the loading of these TDs in the EUP IDE, the programming of Agents on top of the registered artifacts (with a user study with end users who had never programmed before), and the deployment and execution of these Agent programs. These tests have been successful, and we have created several demonstrators and two demonstration videos on top of them.

The next step regarding the integration of the HyperMAS components is clearly the onboarding of devices and services that are not furnished by Hochschule St. Gallen (HSG). These are:

- For UC1, the actual tractor controller as well as a service that delivers waypoints given a specific field.
- For UC2, any sensors that should trigger agents to message doctors and/or patients.

- For UC3, the actual robot controller, engraver service, and the AI component.

We do not expect any issues with the technical integration of these components, since their APIs are required to be TD-compatible and since TD is sufficiently standardized.

2.2 5G Infrastructure

The 5G architecture to be integrated in IntellIoT shares features between UC1 and UC3. Indeed, from an abstract perspective, in both UCs, a private 5G infrastructure provides a 5G link supporting URLL to transmit video streams on one link and control commands (e.g., controlling a robot) on the other link. The main difference between the two UCs is that in UC1, the control command is on the 5G downlink, whereas it is in the 5G UL for UC3. If this will impact the 5G Communication Manager, it does not impact the 5G infrastructure itself. A second difference between UCs is that UC1 is primarily outdoor, whereas UC3 is indoor. If technically, outdoor scenarios need to follow a stricter spectrum protection procedure than indoor, a similar radio hardware will be employed.

Accordingly, a similar 5G architecture will be provided for the UC1 and UC3 integration. UC2 is a bit different due to the usage of a commercial 5G network in the selected hospital. Accordingly, the 5G infrastructure will be limited to an emulation of a 5G MEC.

2.3 Distributed Ledger Technologies

Distributed Ledger Technologies (DLTs) have received significant attention in recent years as an innovative method of storing and updating data within and between individuals and organizations. A distributed ledger is a digital ledger that is different from centralized networks and ledger systems in two ways. First, information is stored on a network of machines, with changes to the ledger reflected simultaneously for all holders of the ledger. Second, the information is authenticated by a cryptographic signature. Together, these systems provide a transparent and verifiable record of transactions.

In the scope of IntellIoT project, we develop a distributed and trusted framework based on the Blockchain and Smart Contracts. In specific, we develop two DLT components called DLT manager and DLT clients to connect the system components together on top of DLT infrastructure.

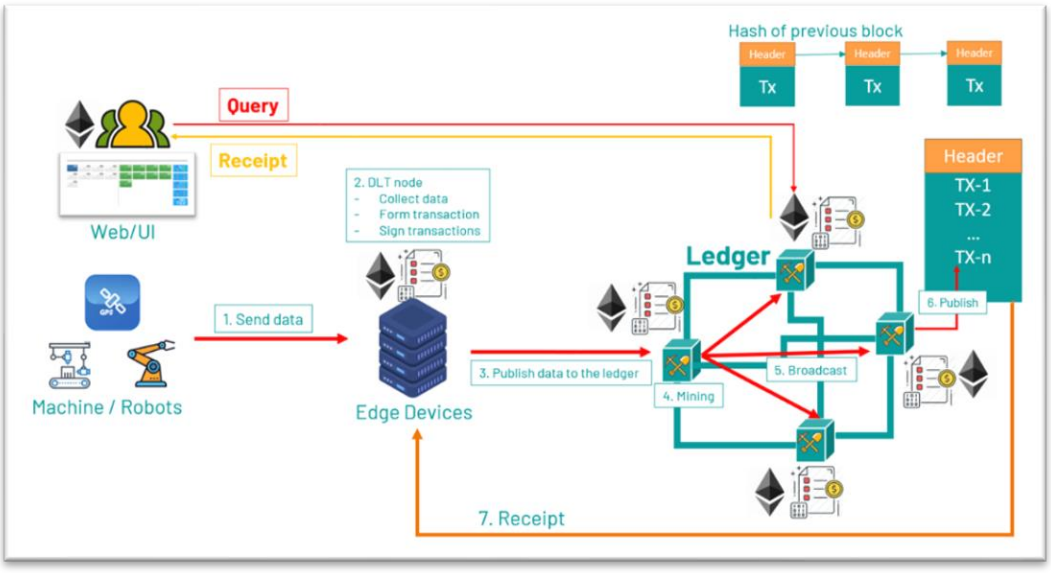


Figure 1. A proof of concept of DLT-based infrastructure for IoT networks

In specific, the DLT-based infrastructure includes several components as described below:

- **The distributed ledger**: The distributed ledger consists of sequence blocks that have been committed to the Ethereum Blockchain network in the history of the network. Each block contains a reference to the previous block, which helps us maintain an ordering over all blocks. We also provide an alternative solution for Ethereum is Quorum or Hyperledger Fabric, depending on the specific scenario and use case.
- **ETH**: The native cryptocurrency of the Ethereum network. Participants are assigned an unlimited amount of ETH to have their code execution requests fulfilled.
- **EVM:** The Ethereum Virtual Machine is a virtual computer whose state every participant on the Ethereum network stores and agrees on. Any participant can request the execution of arbitrary code on the EVM.
- **Accounts:** Where ether is stored. Users can initialize accounts, deposit ether into the accounts, and transfer ether from their accounts to other users. Accounts and account balances are stored in a big table in the EVM, and they are a part of the overall EVM state.
- **Transactions**: A "transaction request" is the formal term for a request for code execution on the EVM, and a "transaction" is a fulfilled transaction request and the associated change in the EVM state. Any user can broadcast a transaction request to the network from a node. For the transaction request to affect the agreed-upon EVM state, it must be validated, executed, and "committed to the network" by another node. Execution of any code causes a state change in the EVM; upon commitment, this state change is broadcast to all nodes in the network.
- **Smart contracts:** A reusable snippet of code (a program) which a developer publishes into EVM state. Anyone can request that the smart contract code be executed by making a transaction request. Because developers can write arbitrary executable applications into the EVM (games, marketplaces, financial instruments, etc.).

```
1  pragma solidity ^0.5.2;
2
3  contract SimpleStorage {
4      uint storedData;
5
6      function set(uint x) public {
7          storedData = x;
8      }
9
10     function get() public view returns (uint) {
11         return storedData;
12     }
13 }
14
15
```

Figure 2. Example of Smart Contract to record the Data to DLT

In order to implement a DLT manager, we develop the Golang-based implementation of the Ethereum client named Geth. Go Ethereum is one of the three original implementations (along with C++ and Python) of the Ethereum protocol. It is written in Go, fully open source and licensed under the GNU LGPL v3.

In general, we build a Proof-of-Concept DLT-based infrastructure for different use cases.

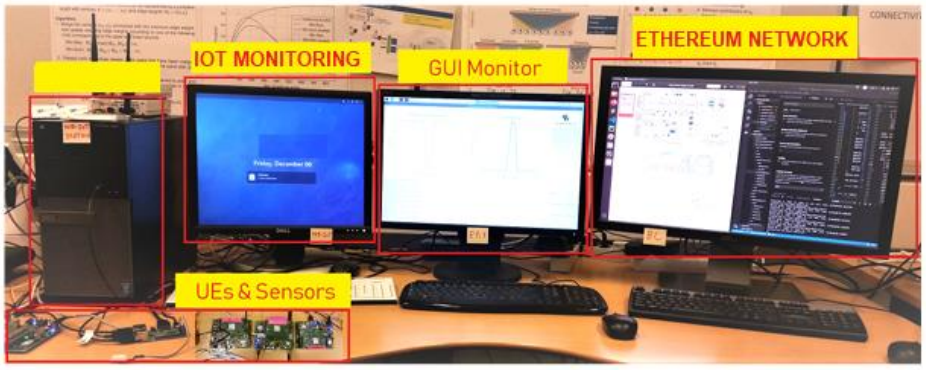


Figure 3. DLT testbed setup based on Ethereum Blockchain platform.

The proof of concept is built on top of Ethereum network. In practice, the simple model includes following steps as below:

- **Step 1:** The IoT data is collected via IoT devices and sensors. The IoT data could be the sensing data, progress update, or control data. For example, in agriculture use case, we collect the GPS update from tractor controller to analyse on the distributed ledger.
- **Step 2:** Each IoT device acts as an DLT client in the DLT-based network. We assume that the IoT device have enough computation and storage capacity to host a running DLT client program. Via Web3 library, we develop a DLT client running on these IoT devices. The DLT clients could be written in Python, Java, or C/C++ depending on the integration with different IntellIoT partners. The DLT client formats the collected IoT data into DLT transactions with its signature, headers. The client needs to register with the DLT about the identity to sign the transactions. Only recognized DLT clients can publish the data to the distributed ledger. Then, the transaction is sent to the distributed ledger.

```
ganache          | Available Accounts
ganache          | ==================
ganache          | (0) 0x2E6d29B17a526334843141F8BEf0f0424760199e (~100000000 ETH)
ganache          | (1) 0xC4D62b4DC0749127b63FB070966c25Ac381c1944 (~100000000 ETH)
ganache          | (2) 0xDF1Db5C73747dB8deD48A3a25C17Af71FBdF1E89 (~100000000 ETH)
ganache          | (3) 0x4988518828cA3aE1E2a3E4dbcce623A40B249102 (~100000000 ETH)
ganache          | (4) 0x936cc17ac0766aE0b5f02Fa1F6BacAE30591313b (~100000000 ETH)
ganache          | (5) 0x5a6571a19e0e444411afb29E8566ba93417C7Ee3 (~100000000 ETH)
ganache          | (6) 0xaA445Bbe0970D210615A37109C734dffBc15f867 (~100000000 ETH)
ganache          | (7) 0xEFAA828E2d2Ca6317fE56a31E44C72390F854006 (~100000000 ETH)
ganache          | (8) 0x53866B55bDc66Ce7a30c20C39F105290C0559aCF (~100000000 ETH)
ganache          | (9) 0x2e94d4AA531df3943Bc09e3eEAa80358a02ad663 (~100000000 ETH)
ganache          |
ganache          | Private Keys
ganache          | ==================
ganache          | (0) 0xf90c16e1628661bc15769b76c8266343ec35db482239904b0ca1bd90c8fbca01
ganache          | (1) 0x7968992c32f33408f0abc9ab42aa95d77fb52524c2e0c62d1adef42840d3d502
ganache          | (2) 0xb492c13c93edc695916522966262ad4dc1c7f64a22b138504c5b250a1c484203
ganache          | (3) 0xb22df97c876895cec01b8ad70b00482e2201dbecc61482a9c1e44544436b1b04
ganache          | (4) 0x69ce909787a29af568092a6d1cb693a3b4510c6d4eebd7d01762aa5f4d824105
ganache          | (5) 0x867f938edd9a5598b40bcf2c68a3106c0fe7486c1a6d50ba9a5171dc865b5306
ganache          | (6) 0xe1921a270bd9751ccc2d04896c7f62cbb5d9cd5dc3195de3fc2d29c483f38507
ganache          | (7) 0x03e88e467120938d05ca0a2677a9a25080e1a16093655bdd3c8fd8c43bd4ba08
ganache          | (8) 0x3b270a195f592a505ce94186793ababb27a33a65a572bc9e7b06766fb6626b09
ganache          | (9) 0x6418670e00dbdb0a6e9bd99f1521f7294f58af0995c7a0534c2021fcd486e710
```

Figure 4. Example of available accounts can be used for identifying and signing transactions.

The transaction is sent via an Ethereum API which can be showed via Swagger as below:

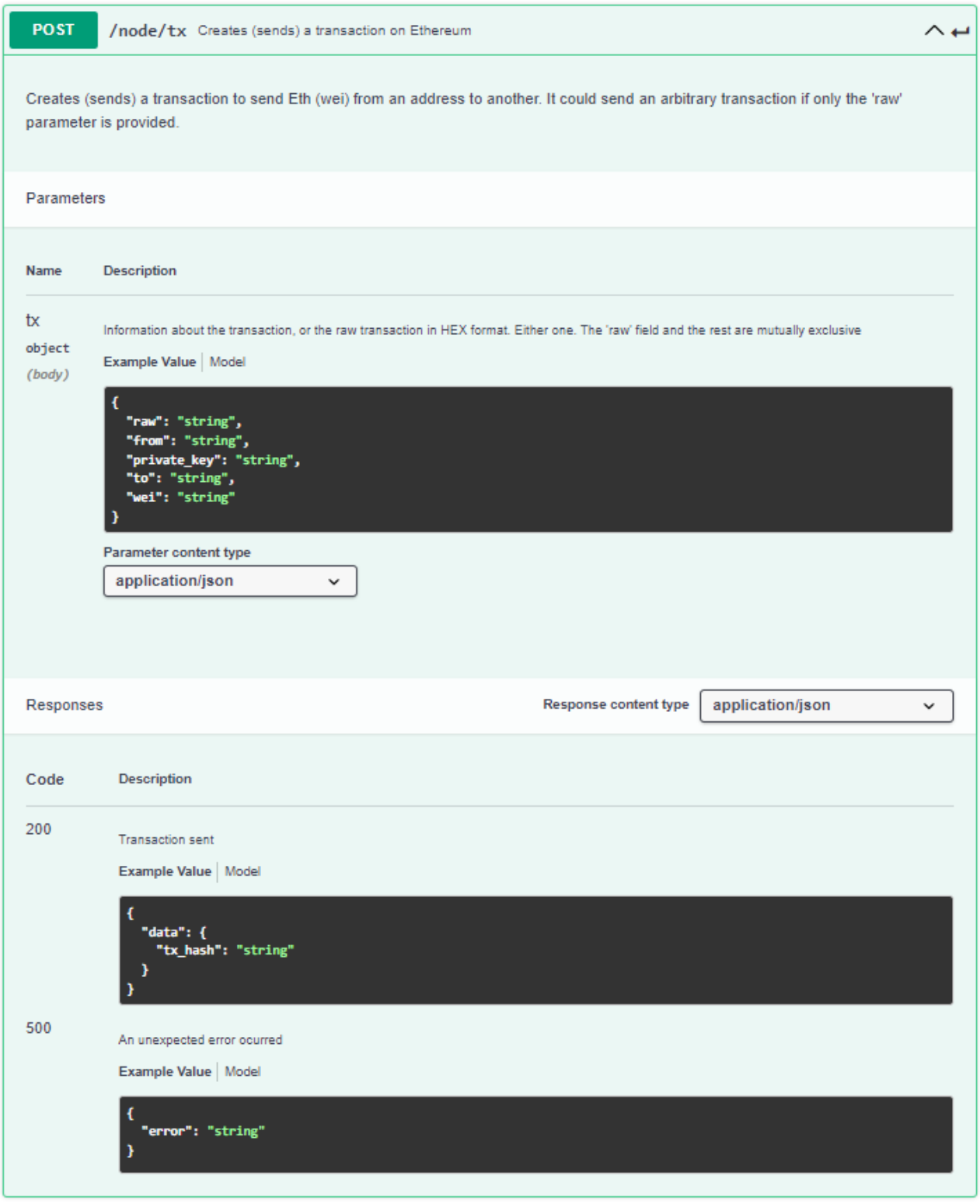


Figure 5. Creates (sends) a transaction to send Eth (wei) from an address to another. It could send an arbitrary transaction if only the 'raw' parameter is provided.

Besides, the clients can communicate with the smart contracts for specific purposes via an API. Note that, each smart contract has a unique address to communicate with. The client program needs to specify the detail of digit address of smart contract and interact with proper features defined in the smart contract. Then, the smart contract will be executed autonomously to complete agreements among involved DLT participants.

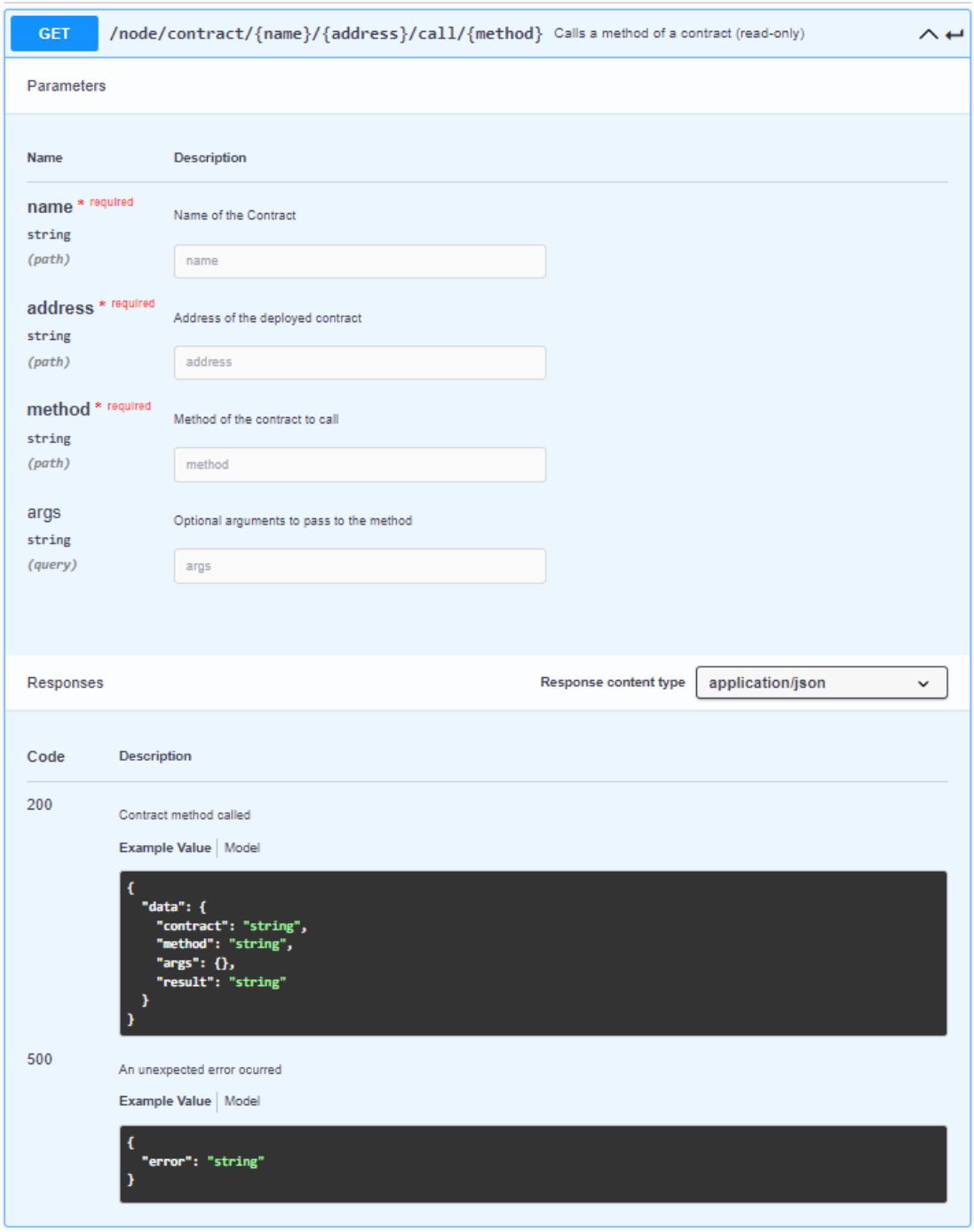


Figure 6. An API to call a smart contract showed in Swagger

- **Step 3:** The transactions are processed and consensus via Proof-of-Work algorithm running on DLT nodes. The nodes sacrifice their computation power to compete for a proof-of-work outcome and validate the transactions from the clients.
- **Step 4**: Depending on the purpose of the transactions, the administrator needs to implement the corresponding smart contract for each application. The smart contract will be deployed in a decentralized manner in DLT network. An example of smart contract deployment is showed as below:

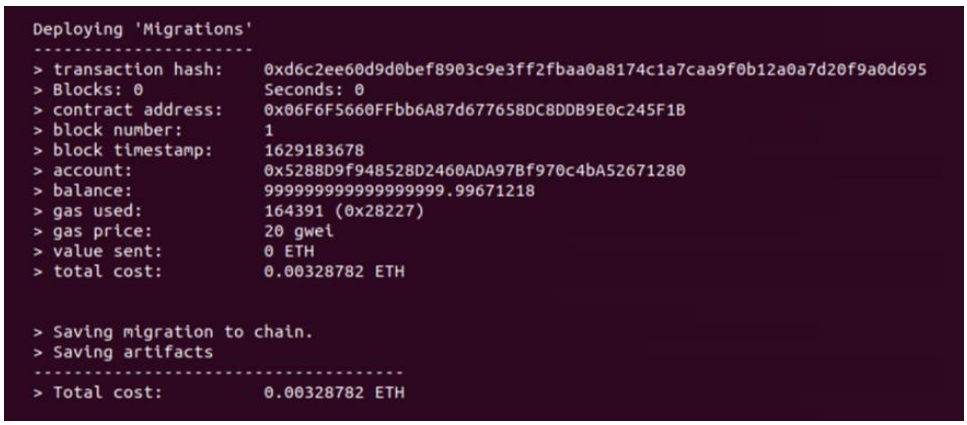


Figure 7. The deployment of a Smart Contract

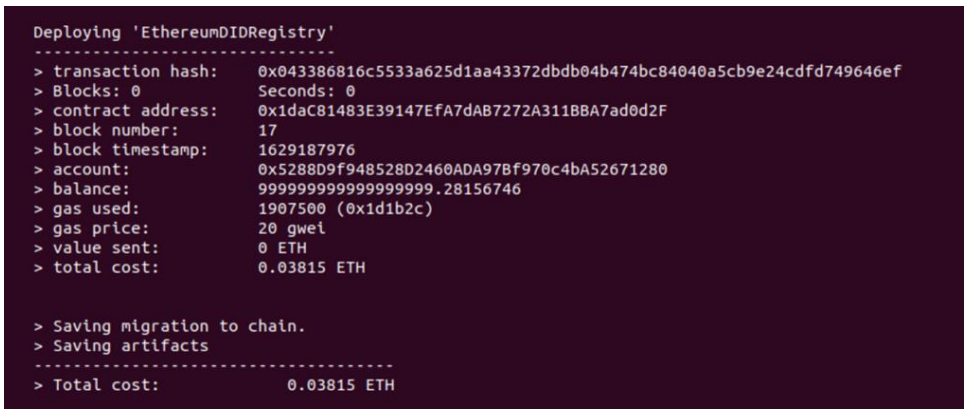


Figure 8. The deployment of Digital Identity Smart Contract

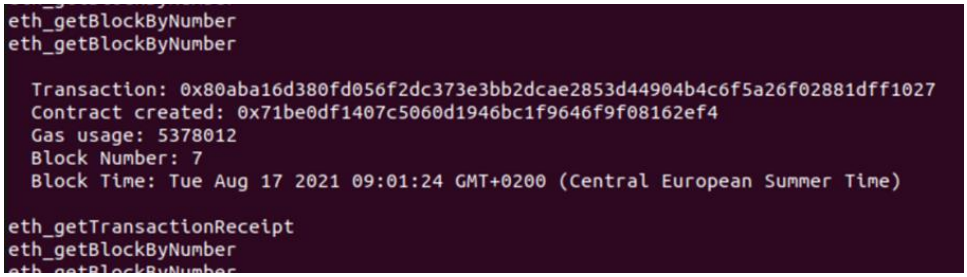


Figure 9. The transaction detail which shows the deployment of smart contract

After all, the data will be recorded transparently and immutability in the distributed ledger. The clients and users can access the data over provided API and client programs. One of the popular and user-friendly ways to interact with Ethereum Blockchain is Metamask wallet, which can be easily installed in web browsers. After registering an account and connect with the DLT via specific port (default 8338 in our setting), the user can query and get the information which is published to the ledger.

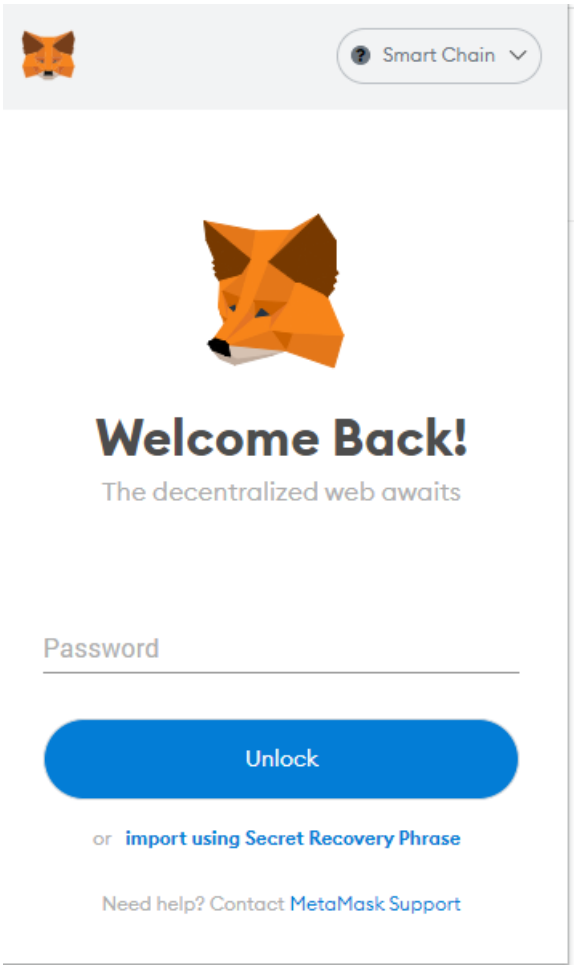


Figure 10. Metamask User Interface

2.4 Trust components

As detailed in Section 6 of D4.4 - "Trust mechanisms (first version)", and subsection 2.4 of D5.1 - "Integrated IntellIoT framework & use case implementations (first version)", a key characteristic that maximises the efficacy of IntellIoT's Trust Enablers is their tight integration, through a dedicated communication bus and the associated interfaces. A high-level overview of this integration is provided in Figure 11, while an abstract deployment-level diagram is provided in Figure 12.

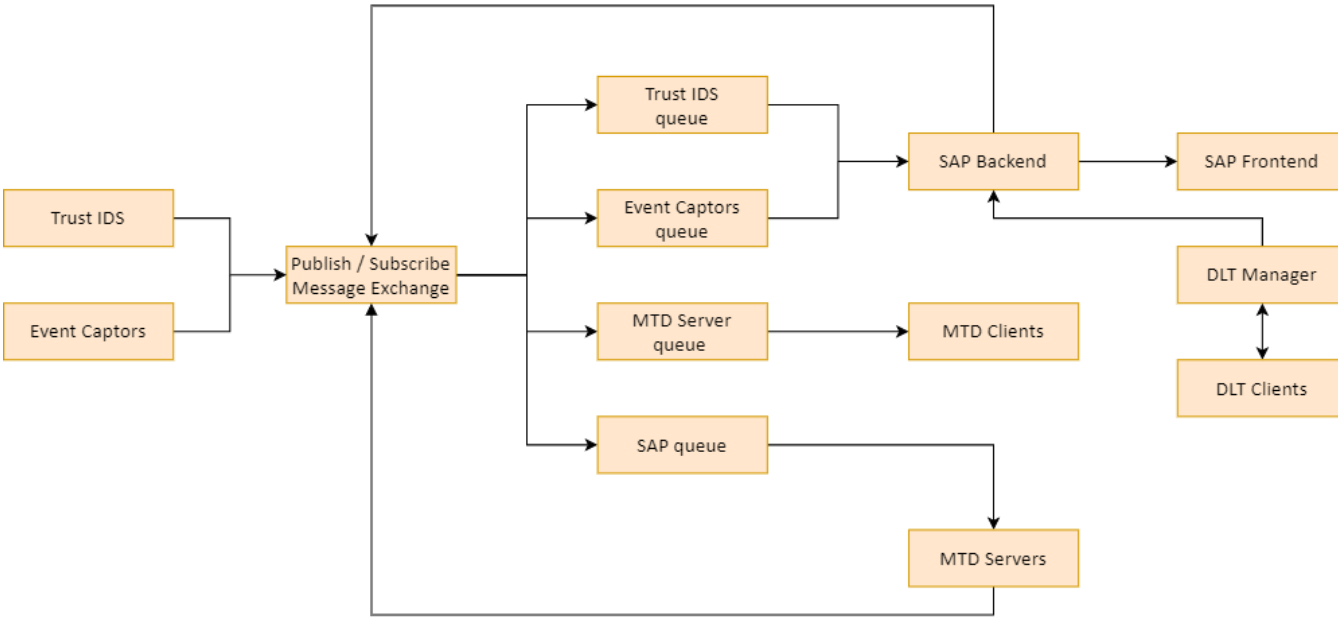


Figure 11. Overview of Trust Enablers' integration via shared Trust Broker.

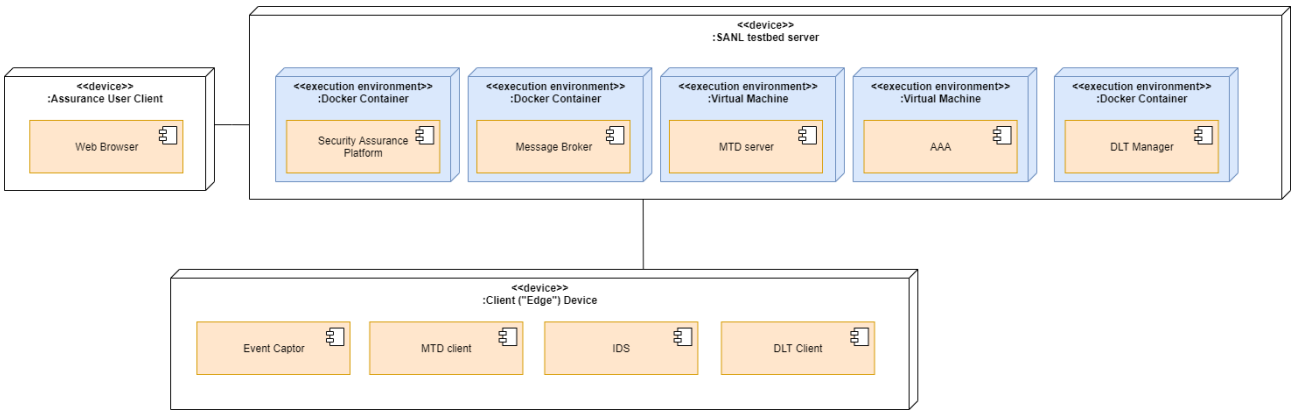


Figure 12. Deployment view of Trust Enablers' integration.

As such, the deployment and integration between the trust components has elements that are testbed-agnostic, while, of course, use case-specific elements (e.g., specific features, configuration aspects) are also needed.

Herein, we provide details on the deployment process that is shared between the different testbeds (i.e., cross- use case), while individual, use case / testbed-specific elements related to deployment and testing will be documented in the corresponding use cases sections that follow (i.e., Sections 3, 4, and 5, for UC1, UC2, and UC3, respectively).

2.4.1 SECURITY ASSURANCE PLATFORM

The Security Assurance Platform (SAP) provides a key integration point for the rest of the trust enablers, as well as the main means of interaction for operators intending to monitor the security assurance posture of IntellIoT's deployment. It also integrates the multi-layer monitoring capabilities of IntellIoT, through ingestion of the raw monitoring evidence of the various event captors; thus, deployment of the multi-layer monitoring enablers takes place concurrently with the deployment of the rest of the SAP components. More details on the capabilities of SAP are provided in Section 2 of D4.4.

The various subcomponents comprising SAP are deployed in dockerized form, facilitating the deployment in the various use case testbeds in a uniform manner. Herein details are provided on the deployment process, as well as the ensuing setup and configuration steps that must take place before it becomes fully operational.

2.4.1.1 DEPLOYMENT, SETUP & CONFIGURATION OF CORE SAP COMPONENTS

As mentioned, each of its tools resides inside their own docker containers. In more detail, their images are pre-built and are downloaded from Sphynx's GitLab container repository during the setup. All of the platform's service dependencies and configuration are documented in a single docker-compose.yml file, which when run, creates the following containers:

- ***Assurance Database:*** This contains a PostgreSQL database with pre-configured schemas for both core platform functionalities as well as the authentication server.
- ***pgAdmin:*** This PostgreSQL administration platform is used for configuration purposes, manually overriding database settings where necessary, and checking the validity of the deployed database.
- ***InfluxDB:*** This is a time-series database used for storing and querying metrics data displayed on the Assurance Platform GUI, such as number of newly detected vulnerabilities grouped by date.
- ***AssuranceModelEditor:*** Also referred to as AssetLoader, this is the core web application of the Security Assurance Platform providing functionality for CRUD functions on the asset model.
- ***Nginx:*** This is the web server serving the Assurance GUI.
- ***AssuranceGUI:*** The Assurance Platform's frontend.
- ***VulnerabilitiesLoader:*** This software is responsible for creating Vulnerability Assessments and producing SSEs consumed by the AssuranceGUI.
- ***CVEFetcher:*** This component fetches CVEs from the National Vulnerability Database (NVD).
- ***IdentityServer:*** This is an Access and Identity Management solution facilitating signing in and securing the platform's APIs.
- ***ApplicationManager:*** This is a solution allowing the publishing and managing of the Security Assurance Platform's APIs. It acts as the mediator between the Assurance GUI and the backend, restricting or throttling access to unauthorized users or specific user roles.
- ***UserManager:*** Is an in-house software developed to manage users and customize the Json Web Tokens (JWT) provided handed by the IdentityServer.
- ***PenTester:*** This software enables the dynamic testing functionalities of the Assurance Platform, discovering new assets and vulnerabilities.

In essence deploying the Security Assurance Platform can be easily done through the use of the docker-compose.yml file. However, upon completion, further configuration is necessary in order to login and start using it.

The first step involves adding existing APIs to the WSO2 Application Manager (APIM)[1]. This is done by creating a new API and pointing it to each respective tool's OpenAPI (Swagger)[2] URL. In this way, the Application Manager acts as the mediator, receiving requests from the Assurance GUI and forwarding them to the backend. No ports except the APIM's default (8245) are forwarded. As such, one cannot directly send a request to the backend, having to send it through the APIM instead.

In the second step, it is important to properly configure the WSO2 Identity Server[3] (see Figure 13and Figure 14). More specifically, in order for the JWT to be properly generated the groups and roles claims must be manually enabled. In addition, entitlement policies may be created in order to define the endpoints each user role is allowed to use. These mediation policies are in the form of XACML files and contain, among others, the endpoint URLs along with

[1] https://wso2.com/products/app-manager/
[2] https://swagger.io/specification/
[3] https://wso2.com/identity-server/

permit/deny options for each role (see Figure 15), allowing for fine-grained control of user access to the different APIs.

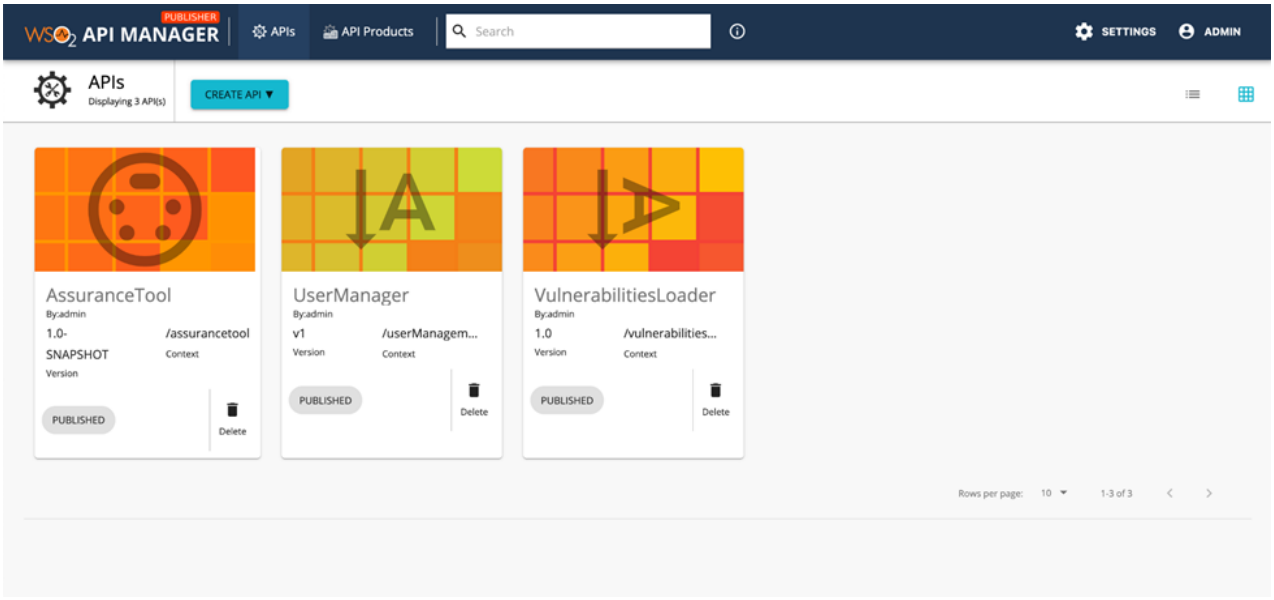


Figure 13. The API Manager interface, with three added APIs

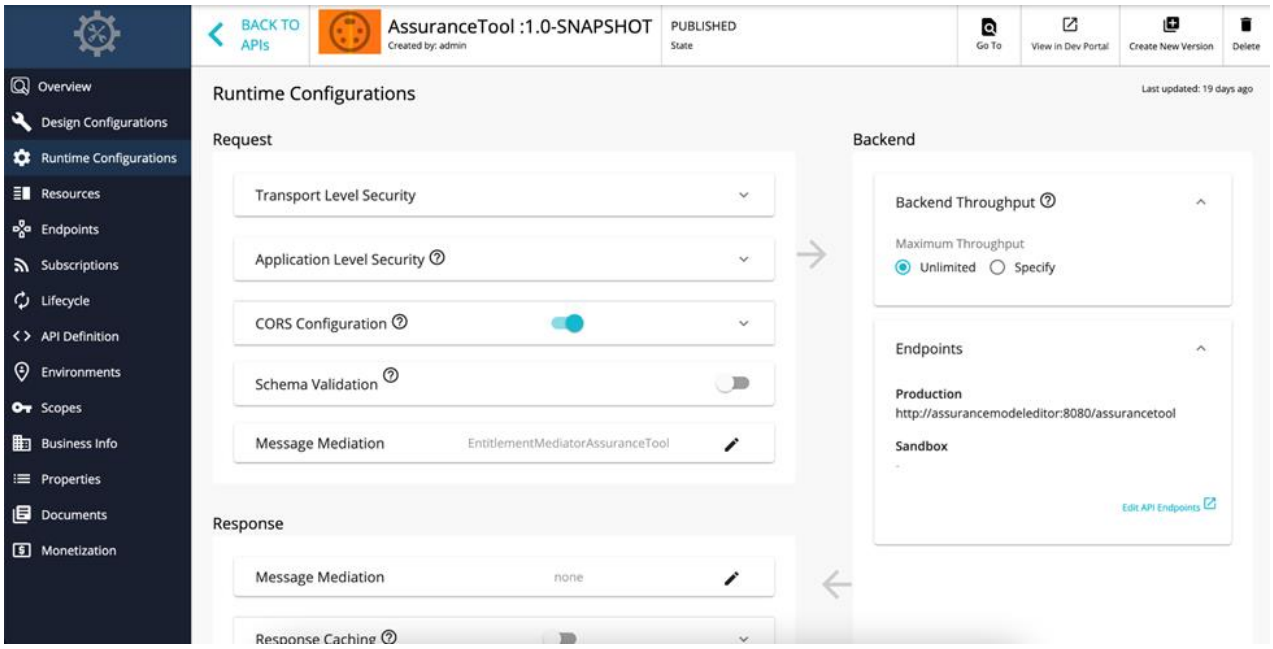


Figure 14. Configuration page for a single API with different options on how to handle it.

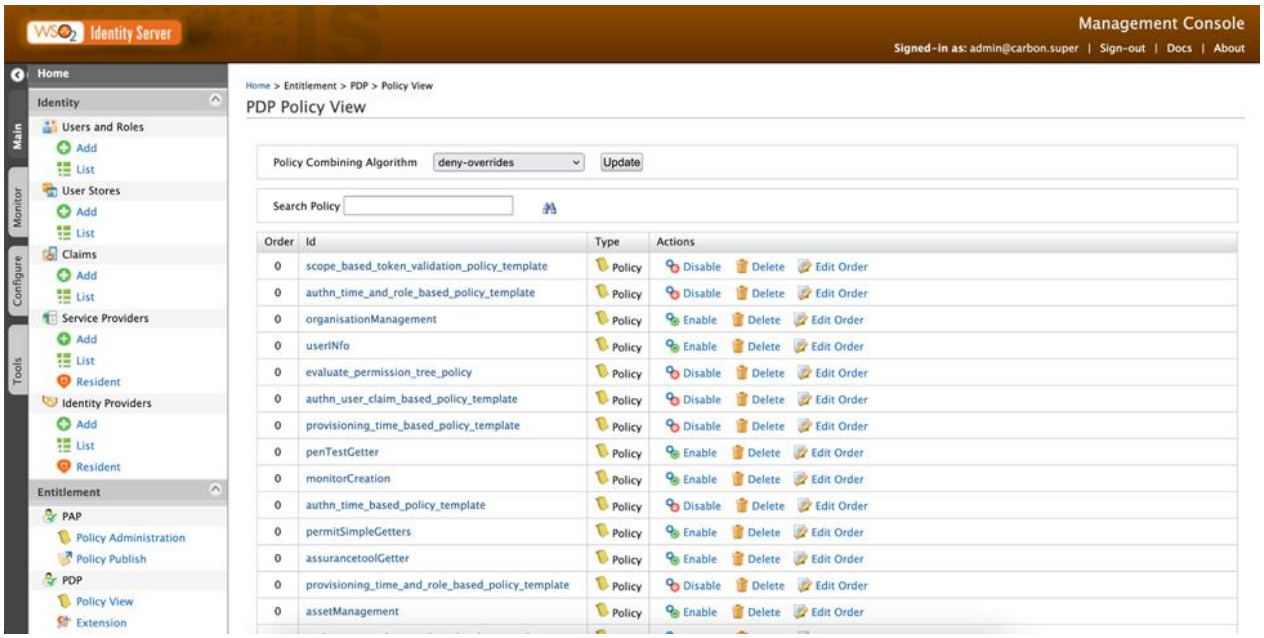


Figure 15. Policies defined in the WS02 Identity Server.

2.4.1.2 DEPLOYMENT, SETUP & CONFIGURATION OF MONITORING COMPONENTS

The monitoring ecosystem that comes alongside SAP and will provide the core multi-layer capabilities of IntellIoT is also prepared to be deployed in a Docker ecosystem. Installation can be performed easily by downloading the relative setup from Sphynx's GitLab repository.

MONITORING ASSESSMENT TOOL

The Monitor Instance and orchestrator container may come up by simple deploying them using the docker-compose tool (see Figure 16). By default, the Docker will expose port 9999 for the orchestrator so make sure this port is not in use. Every time a new assessment is requested, a new monitoring instance docker will be created. Orchestrator, using docker-in-docker techniques will configure and deploy a docker container to await future assessment requests.

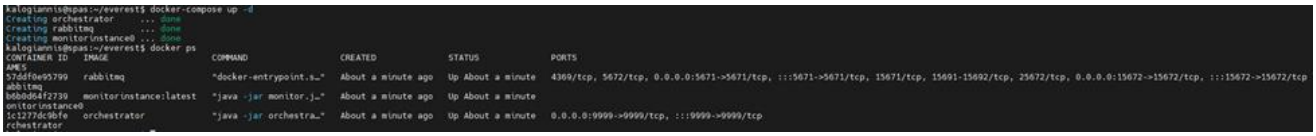


Figure 16. The process of bringing up the containers of monitor using the docker-compose utility.

This will create the orchestrator image and pull in the necessary dependencies. Also, it will start up the first monitoring instance docker that will be used for the first assessment. There are no ports exposed for these instances as the name resolution is enough for the docker containers to communicate with each other. So, the only port needed is port 9999 for the orchestrator docker to expose the API for the assessment requests.

ELK STACK

Elasticsearch-Logstash-Kibana (ELK)[4] comes in a dockerized environment containing each service as a separate container. Installation can be performed easily by downloading the automated setup from Sphynx's GitLab repository. The installation process is fully automated using Makefile scripts. The installation process starts by setting a series of environmental variable in the respectively environmental file (.env), such as the version of the stack

(currently set to 7.17), the ports of each service that will be exposed, user access credentials, container names etc., as shown in Figure 17.

```
COMPOSE_PROJECT_NAME=elastic
ELK_VERSION=7.17.0

#----------- Resources -------------------------#
ELASTICSEARCH_HEAP=2048m
LOGSTASH_HEAP=512m

#----------- Hosts and Ports -------------------#
# To be able to further "de-compose" the compose files, get hostnames from environment variables instead.

LOCAL_HOST=localhost
ELASTICSEARCH_HOST=elasticsearch
ELASTICSEARCH_PORT=9200

KIBANA_HOST=kibana
KIBANA_PORT=5601

LOGSTASH_HOST=logstash
LOGSTASH_PORT=8080

#----------- Credientals -----------------------#
# Username & Password for Admin Elasticsearch cluster.
# This is used to set the password at setup, and used by others to connect to Elasticsearch at runtime.
# USERNAME cannot be changed! It is set here for parmeterization only.
ELASTIC_USERNAME=elastic
ELASTIC_PASSWORD=elastic
#AWS_ACCESS_KEY_ID=notherealid
#AWS_SECRET_ACCESS_KEY=notherealsecret

#----------- Cluster ---------------------------#
ELASTIC_CLUSTER_NAME=elastdocker-cluster
ELASTIC_INIT_MASTER_NODE=elastdocker-node-0
ELASTIC_NODE_NAME=elastdocker-node-0

# Hostnames of master eligble elasticsearch instances. (matches compose generated host name)
ELASTIC_DISCOVERY_SEEDS=elasticsearch

#----------- For Multinode Cluster -------------#
# Other nodes
#ELASTIC_NODE_NAME_1=elastdocker-node-1
#ELASTIC_NODE_NAME_2=elastdocker-node-2
```

Figure 17. Snapshot of the .env file where all relative variables of ELK stack deployment are set

Considering the secure communication channels that are used between the components, the necessary certificates must also be created, by executing the corresponding scripts (see Figure 18). Furthermore, in order to bring up the three services (Elasticsearch-Logstash-Kibana) the user executes the make script (Figure 19). The makefile scripts includes the functionality to automatically create the necessary configuration file (beat.yml) that is needed when deploying any beat system[5].

[4] https://www.elastic.co/what-is/elk-stack
[5] https://www.elastic.co/beats/

```
kalogiannis@spas:~/docker-elk$ make certs keystore p12pem
docker-compose -f docker-compose.setup.yml run --rm certs
WARNING: The AWS_ACCESS_KEY_ID variable is not set. Defaulting to a blank string.
WARNING: The AWS_SECRET_ACCESS_KEY variable is not set. Defaulting to a blank string.
Building certs
Step 1/4 : ARG ELK_VERSION
Step 2/4 : FROM docker.elastic.co/elasticsearch/elasticsearch:${ELK_VERSION}
 ---> 6fe993d6e7ed
Step 3/4 : COPY scripts/docker-healthcheck .
 ---> Using cache
 ---> aa615d825830
Step 4/4 : HEALTHCHECK CMD sh ./docker-healthcheck
 ---> Using cache
 ---> 63bddbfd9964

Successfully built 63bddbfd9964
Successfully tagged elastic_certs:latest
WARNING: Image for service certs was built because it did not already exist. To rebuild this image you must use `docker-compose build` or `docker-compose
 up --build`.
Creating elastic_certs_run ... done
======= Generating Elastic Stack Certificates =======
================================================
Clearing Old Certificates if exits...
Generating...
Unzipping Certifications...
Applying Permissions...
================================================
SSL Certifications generation completed successfully.
================================================
docker-compose -f docker-compose.setup.yml run --rm keystore
WARNING: The AWS_ACCESS_KEY_ID variable is not set. Defaulting to a blank string.
WARNING: The AWS_SECRET_ACCESS_KEY variable is not set. Defaulting to a blank string.
Building keystore
Step 1/4 : ARG ELK_VERSION
Step 2/4 : FROM docker.elastic.co/elasticsearch/elasticsearch:${ELK_VERSION}
 ---> 6fe993d6e7ed
Step 3/4 : COPY scripts/docker-healthcheck .
 ---> Using cache
 ---> aa615d825830
Step 4/4 : HEALTHCHECK CMD sh ./docker-healthcheck
 ---> Using cache
 ---> 63bddbfd9964

Successfully built 63bddbfd9964
Successfully tagged elastic_keystore:latest
WARNING: Image for service keystore was built because it did not already exist. To rebuild this image you must use `docker-compose build` or `docker-comp
ose up --build`.
Creating elastic_keystore_run ... done
========== Creating Elasticsearch Keystore ==========
================================================
Elastic password is: elastic
Setting bootstrap.password...
Saving new elasticsearch.keystore
======= Keystore setup completed successfully =======
================================================
Remember to restart the stack.
Your 'elastic' user password is: elastic
================================================
docker-compose -f docker-compose.setup.yml run --rm p12pem
WARNING: The AWS_ACCESS_KEY_ID variable is not set. Defaulting to a blank string.
```

Figure 18. The process of creating the necessary certificates for TLS communication using the make file.

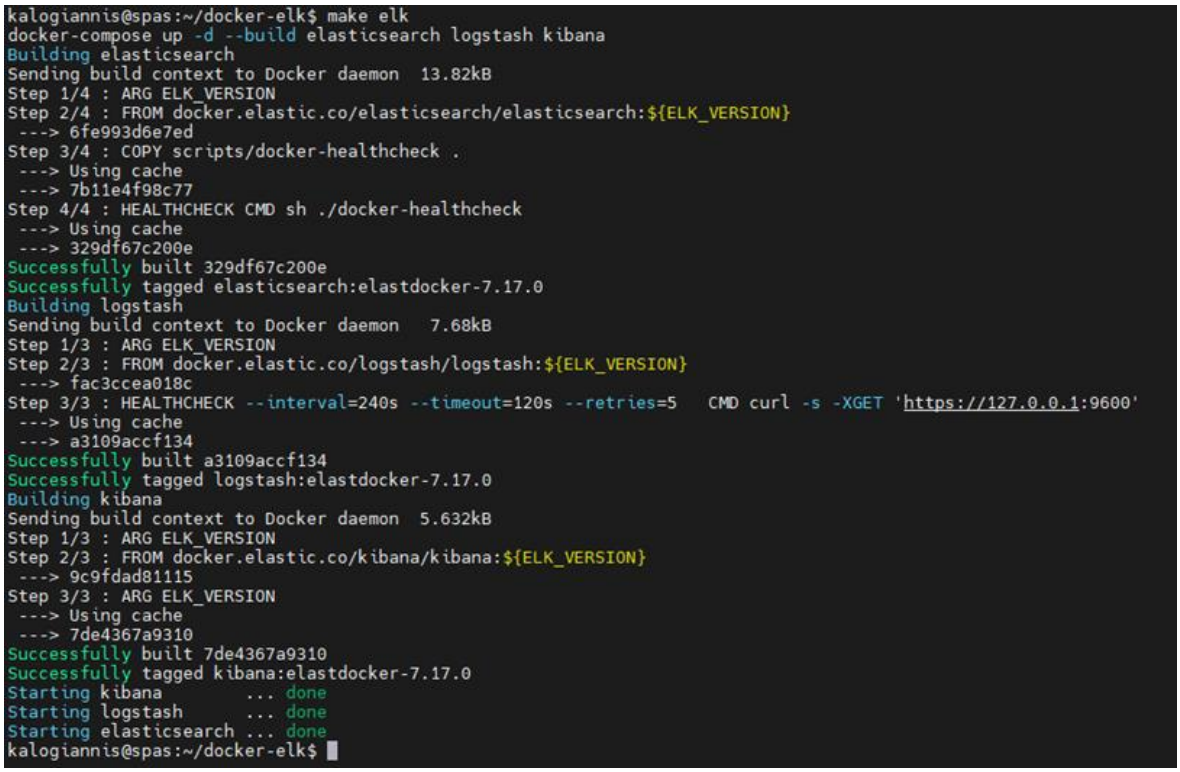


Figure 19. Bringing up the ELK stack as three separate containers

EVENT CAPTORS MODULE

The Event Captors module can be easily deployed since it is fully dockerized. By downloading it for the Sphynx's GitLab repository, the bash script go.sh will compile and deploy the Event Captors module as a separate container (namely ecorchestrator; see Figure 20). Further changes can be made by editing the respectively docker-compose.yaml file.



Figure 20. Bringing up the event captors module container

2.4.2 TRUST IDS AND MTD

Similarly to the SAP components, the Trust IDS and MTD (Client and Server) are dockerized to facilitate easy deployment along with the rest of the Trust Enablers at Sphynx's testbed. Trust IDS, MTD Client and MTD Server components communicate through the common dedicated communication bus.

We provide two separate docker images:

1. a bundle of Trust IDS and MTD Client components image
2. the MTD Server component image

These images are based on a minimal Alpine Linux (https://alpinelinux.org) image with their respective component binaries and the libraries they depend on. Both of them can mount external files over their default configuration for easier setup per deployment.

Integration tests have been performed on Sphynx's testbed validating the proper communication of the trust components.

Further details about the components operation and communication can be found at sections 4, 5 of "Deliverable D4.4Trust mechanisms (first version)" and sections 2.4.3, 2.4.4 of "Deliverable D5.1 Integrated IntellIoT framework & use case implementations (first version)"

2.4.3 AUTHENTICATION, AUTHORIZATION AND ACCOUNTING (AAA)

The Authentication, Authorization and Accounting (AAA) component is deployed using docker compose and it contains the following images:

1. Keycloak, the selected AAA solution
2. MySQL, acting as the Keycloak database
3. OpenResty, acting as a Nginx reverse proxy with Lua (https://www.lua.org/about.html) plugins

MySQL container has no external network connectivity; it is only accessible by the Keycloak container. Keycloak can be either accessed directly or through the reverse proxy. The reverse proxy acts both as a Keycloak client and a TLS termination proxy. This allows for internal REST services deployed by other IntellIoT partners and external contributors to remain simple, as they need to know nothing about authorization, authentication and encrypted connections. All security aspects are handled by Keycloak and OpenResty.

Configuring Keycloak and OpenResty is performed by mounting external configuration files. We can also prepare the configuration, if needed, for each Use Case demo in order to contain all the Use Case related information beforehand thus minimizing the setup time.

AAA integration will be available for each use case testbed, as they are created. Currently, we have a general demo that verifies the proper AAA operation for all the supported cases:

- direct authentication/authorization using Keycloak
- indirect authentication/authorization by Keycloak through a reverse proxy using a web login form
- indirect authentication/authorization by Keycloak through a reverse proxy using tokens

Further details about the AAA operation can be found at section 3 of "Deliverable D4.4 Trust mechanisms (first version)" and sections 2.4.5 of "Deliverable D5.1 Integrated IntellIoT framework & use case implementations (first version)"

3 UC1: AGRICULTURE

In this chapter, the physical hardware (Section 3.1) and software (Section3.3) deployment for the agriculture use case will be described. It will provide an overview of how the hardware will be deployed in the final demonstrator and how the different apps developed in the project for the agriculture use case will be on the available hardware. Section 3.3 will provide an overview of the integration and tests of the different hard- and software in the overall use case demonstrator

3.1 Hardware deployment

In this chapter, an overview of all the hardware that will be deployed in the demonstrator of the agriculture use case will be introduced and described.

At the moment of writing this deliverable, much of the hardware is not deployed or integrated yet or some parts are not even available yet. Nevertheless, the hardware components have all been identified and have either been ordered or available and are prepared for integration. Figure 21 provides an overview of the different infrastructures inside the agriculture use case, including the different hardware components that are being deployed inside the individual architectures. As clear from the picture, this figure is not true to scale.

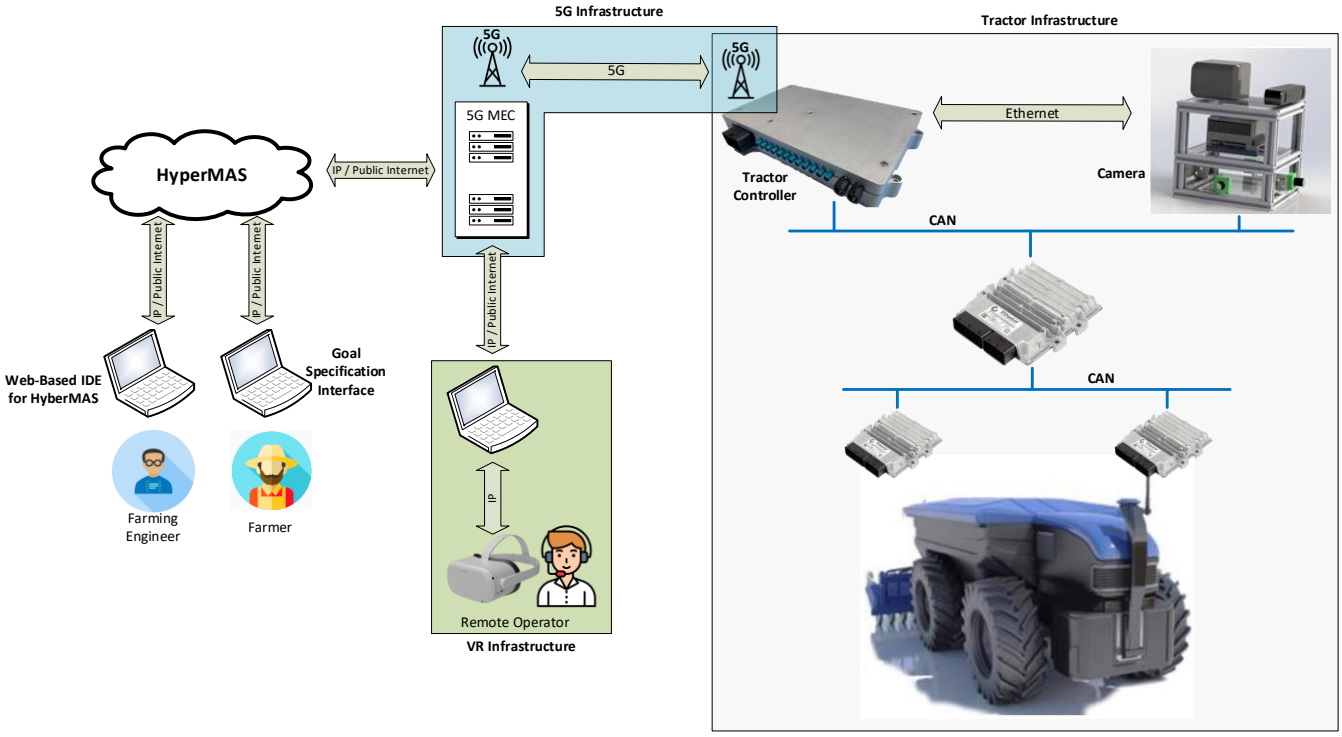


Figure 21: Hardware Deployment in UC1 - Agriculture

It provides a rough overview of the hardware deployed at the tractor and how they are connected (*Tractor Infrastructure*), the external communication architecture (5G Infrastructure) and the solutions for the human operator (VR Infrastructure). In the following subsections, these individual architectures will be described in more detail together with the individual hardware components. Software components will be deployed to the hardware components. A detailed description of the software components is provided in deliverable D5.1. Section 0 gives an overview of the deployment of the different software components inside the use case.

3.1.1 TRACTOR INFRASTRUCTURE

The tractor infrastructure (right part of Figure 21) consists of multiple parts, namely (i) the tractor platform itself, (ii) the camera system, and (iii) the tractor controller hardware.

The *tractor platform* is an E-tractor prototype developed by AVL. It is 3535 mm long, 1900 mm wide and weighs 1750 kg. The velocity range of the vehicle is 10 kph in forward and reverse direction. Each of the two axes has an electric motor to propel and - by recuperation - brake the tractor. A third electric motor drives a pump that is providing hydraulic power for steering and braking. All wheels are equipped with disc brakes and both axes can be steered allowing for a small turning radius. A battery management system (BMS) can be loaded by an on-board charger (OBC) with 400V and supplies the high-voltage (800 V) electric power for the three electric motors and a DCDC converter that provides 12 V for the on-board systems like control units, sensors and actuators. All those embedded systems described before have a control unit and those are connected in a network of four CAN busses. An additional vehicle control unit (VCU) commands all the subsystems as a master and communicates with the *camera system* executing its driving commands and reporting the tractor's state.

The camera system consists of 4 Stereo cameras, an IMU and a GPS sensor connected to a NVIDIA Jetson Xavier computer. This is connected to the tractor VCU on one side and to the tractor controller hardware on the other side. It observes the environment of the tractor and autonomously navigates it to the waypoints received. When this is not possible this is reported and the HIL (Human In the Loop) service can take over. The collected data is streamed to the *tractor controller* that forwards the information to the HIL application, as well as the AI so it can learn new behavior.

The *tractor controller hardware* is a high-performance computing platform prototype, specially developed for the off-highway domain, making it ruggedized to stand the different environmental conditions. It supports multiple interfaces (e.g., Ethernet, CAN, USB, etc.) enabling different data sources to provide the required data to the different apps on the controller. The controller will host most of the apps developed for the agriculture use case and will be the component that connects the external world with the vehicle, via the 5G infrastructure. The tractor controller will be mounted on the tractor platform and be connected with the rest of the tractor via either Ethernet or CAN, dependent on the data that needs to be transferred.



Figure 22: Tractor Controller hardware (left component) in test setup

3.1.2 5G INFRASTRUCTURE

The 5G infrastructure consists of an OAI 5G gNB with an OAI 5G Core and Mosaic5G FlexRIC and FlexCN dynamic 5G management functions. The HW depicted on Figure 23 will be installed indoor and connected to the IntelllioT infrastructure, whereas the HW depicted on Figure 24 will be installed outdoor on a metal pole. The Intelllot infrastructure is connected to the 5G infrastructure via the 5GCore over an ethernet connection. The Mosaic5G FlexRIC and FlexCN will provide APIs to the 5G Communication Manager located on the edge architecture.



Figure 23: 5G HW/SW for the 5G Core (left) and 5G gNB (right) respectively

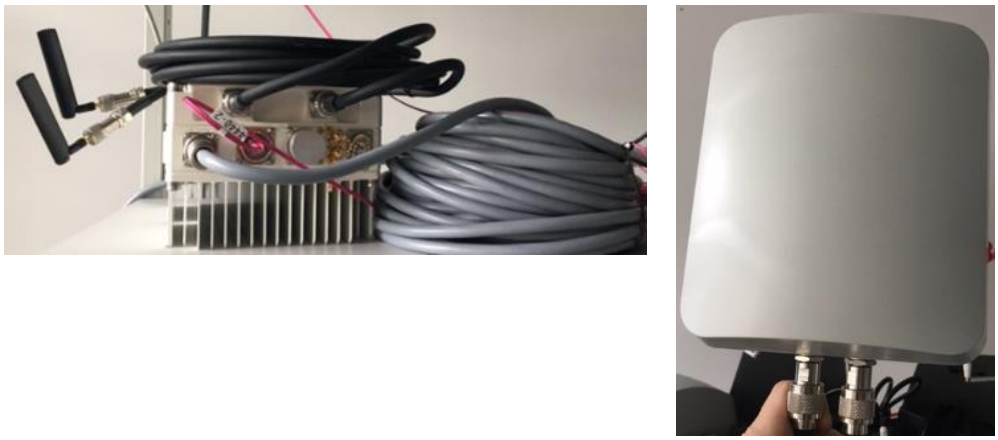


Figure 24: 5G radio HW, including the Radio Head Unit (RRU, left) and the 5G directional dual antenna (right)

Finally, the 5G HW architecture will be completed by a 5G UE (Figure 25) installed on the tractor to connect to the 5G infrastructure. The 5G UE dongle has a USB3 connector, which is connected to the tractor controller.



Figure 25: 5G User Equipment to be installed on the Tractor

3.1.3 VR INFRASTRUCTURE

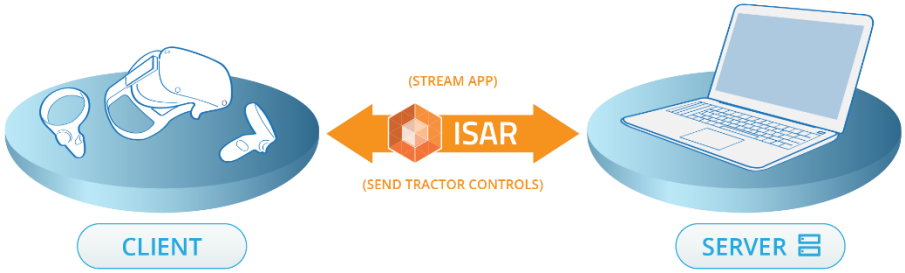


Figure 26. VR Infrastructure

The VR Infrastructure is composed of two parties, the Client side (left side) and the Server side (right side) as demonstrated in Figure 26.

The Client is a combination of a VR Head-Mounted Device (HMD) which is a 128 GB Oculus Quest 2 and battery-based (2AA-Batteries) Controllers[6] (Figure 27) . The controllers send movement control data to the HMD via a Bluetooth connection.

[6] More HW specifications can be found here



Figure 27.: Oculus Quest & Controllers[7]

The Server side is a powerful laptop hosting the HIL Application with the following HW specifications:

- Operating System (OS): Windows 10 Pro 64-bit (10.0, Build 19042)
- Memory (RAM): 16348MB (16 GB)
- Processor (CPU): Intel(R) Core(TM) i7-11800H @2.30GHz (16 CPUs)
- Graphics Card (GPU): GeForce RTX 3070 Laptop GPU

The Client and Server are linked to each other through ISAR (Interactive Streaming for Augmented Reality). ISAR is a plug-in added to Unity3D that allows applications to stream their contents to HMDs and receive arbitrary data back from it. In this context, it will stream the data from the HIL Application to the Oculus Quest 2 and display it to the Operator while receiving tractor control data from the controllers.

3.2 Software / Apps deployment

Figure 28 shows the deployment of the different edge apps on the hardware infrastructures presented in the previous chapter. Please refer to WP3 and WP4 deliverables for a detailed description of the individual software components. The different software components are depicted as square boxes and located at the hardware, where they will be deployed. The only software component that is not deployed on a specific hardware, is the agent, which is deployed in the HyperMAS, which will be deployed in a cloud solution. Some apps are in principle tractor specific, and are thus deployed on the vehicle, which are directly interaction with the data available at the tractor. These can be seen as edge apps, where the tractor controller is deployed as an edge device on the tractor. These apps will receive the data from the tractor either directly via the CAN network or via the Ethernet network through which the tractor controller is connected to the camera system and its processor.

Some software components are deployed to the base station, which is the 5G MEC, that will be located next to the fields, where the tractor will be deployed. These apps function as server and will collect potentially data from multiple vehicles, demonstrating the scalability of the solutions.

[7] Image from Oculus Quest 2 Official Website

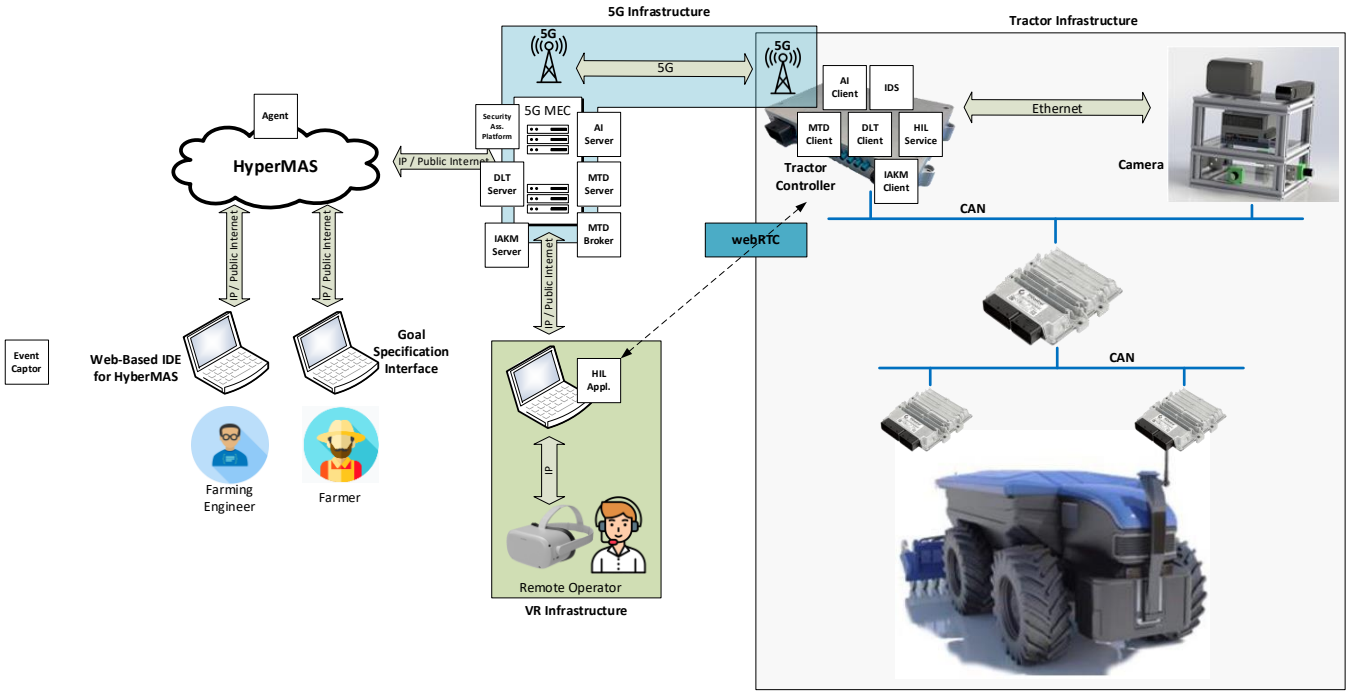


Figure 28: Software deployment of the different apps to the identified hardware infrastructures for UC1

Additionally, Figure 28 shows a dotted arrow between the HIL application and the Tractor Controller, with a software component marked in Blue. This arrow and component states that there is a virtual connection between the HiL application and the tractor using a webRTC interface, used by the human operator for controlling the vehicle. This component will not be described in the following chapters, but is mentioned where necessary.

3.3 Integration Tests

This chapter describes the iterative integration and tests of the hard- and software components deployed in the agriculture use case. The integration steps that are described in the following sections

3.3.1 INTEGRATION AND TESTS OF TRACTOR CONTROLLER AND TRACTOR

As mentioned before, the tractor controller is a high-performance computing platform for vehicles in the off-highway domain. The tractor controller is equipped with a Linux Operating system running ROS and capable of hosting multiple virtual machines, so dockerized applications can be hosted on the controller. Before the actual integration of the tractor controller on the tractor, the functionality of the tractor controller is being tested in a lab environment (see Figure 22).

First lab tests with the tractor controller have been performed with simple dockerized applications (e.g., "Hello World" applications) being hosted on the controller and validating if the applications are providing the expected output. The actual testing of the other functionalities is more related to the testing of the different applications that will be hosted on the tractor respectively the tractor controller. These tests will be performed in cooperation with the functionality testing of the different components but will either be performed in the lab environment or, ideally, after the integration of the controller on the tractor. An important aspect that needs to be tested is the capability of hosting all the different applications on the controller or if other locations are required for executing the applications (e.g., on the 5G MEC).

The actual integration of the tractor controller onto the tractor will require more than software being able to communicate with each other. The tractor controller needs to be mounted on the tractor, securing the controller in place, so it won't get loose during operation of the tractor. During the integration, the following steps will be considered:

- Mounting and securing the controller hardware onto the tractor platform.

- Connecting the controller to the power source on the tractor, so it can run without external power supplies to the tractor.
- Integration into the internal network of the tractor (using the already available CAN and Ethernet network on the tractor)
- Registration of the tractor controller into the operating system of the tractor, which is based on ROS. ROS-based messages and packages are exchanged between the tractor controller and the software controlling the tractor movements.
- For functionality tests, the tractor controller will send draft messages to the tractor (based on ROS interfaces and messages), so it can be validated that the messages are being received by the software running on the tractor. Vice versa, the tractor itself will send ROS data to the tractor controller, to validate that a bi-directional communication between the tractor and the controller has been correctly established.
- If these integration and test steps are successful, the different applications will be hosted on the tractor (e.g., DLT, AI, HiL Service, 5G functionality, etc.) and it will be tested if data collected by the tractor (e.g., GPS data, camera streams) can be forwarded through the tractor controller to the respective applications.

As the tractor is in a different location as where the lab testing environment for the tractor controller is, the tractor controller will de detached from the tractor again after the test sessions and be returned to the lab environment, where the continuous integration of the applications to be hosted on the controller will take place. This is also required, as the tractor is not continuously connected to the network and thus not reachable from remote. This will allow the use case team to continue integration, testing and validating the functionalities of the different applications to be running on the tractor.

3.3.2 INTEGRATION AND TESTS OF HYPERMAS AND TRACTOR

As described in Section 5.1, we have already successfully performed tests with small mobile robots in a laboratory environment. These robots have been modelled as "Tractorbots" and feature HTTP APIs that we expect to be similar to those of the autonomous tractors. Their APIs are TD-described and have been successfully deployed at the HyperMAS Infrastructure, loaded into the EUP IDE (see Section 5.1) and used by agents that have been programmed by members of our team as well as by individuals who had never programmed before (in the context of a user study) - one of the first functional deployments can be seen in a demonstrator video[8] and we are currently producing another demonstration video of a more advanced prototype. One research article that describes this process and is specific to IntelliIoT's UC1 has recently been submitted to the 7th IFAC Conference on Sensing, Control and Automation for Agriculture.

At the moment, as the controller software for the autonomous tractor is being written, we are defining the concrete HTTP API to the tractor controller jointly with HSG, AVL, and TTC. In addition, a further component is required that will supply Agents with waypoints that they then pass on to the tractor controller – this will also feature an HTTP API. The resulting APIs will then be described with TD and the respective TDs will be integrated with the HyperMAS Infrastructure to use it within the EUP IDE, i.e., to enable domain experts without programming experience to configure Agents that control a tractor given an end-user goal (e.g., "Fertilize field 5").

Based on these TDs, we will be able to perform all required integration tests with the tractor:

1. Registration of the TDs for the tractor controller and the waypoint service with the HyperMAS Infrastructure. The TDs shall be successfully registered with the HyperMAS Infrastructure, and it shall be possible to retrieve them given an entry point to an environment or workspace on the infrastructure.
2. Loading of both TDs from the HyperMAS Infrastructure to the EUP IDE. The EUP IDE shall successfully retrieve the TDs from the HyperMAS Infrastructure and render them as composable blocks within its interface.
3. Configuration of an Agent that uses both artifacts, using the EUP IDE. The user shall be able to configure a meaningful program for an Agent that retrieves waypoints from the waypoint service and assigns them to the

[8] https://www.youtube.com/watch?v=KIR3kknzTaE

tractor controller. It shall be possible to extend this Agent with functionality from other registered TDs (also across use cases, e.g., with the messaging service from UC2)

4. Deployment and running of this Agent. The Agent shall be successfully deployed in an Agent runtime and execute its program. In this process, the waypoint service shall respond with correct waypoints and the tractor controller shall move the tractor to assigned waypoints.

Because of the standardization of TD and our recent interactions about the HTTP API of the tractor controller, we do not foresee any issues with these tests. After these tests, the next step will be to integrate further services from other partners across IntellIoT.

3.3.3 INTEGRATION AND TESTS OF HIL SERVICE

The HIL service in the Agriculture use case has very few functionalities. It is sort of the interface between the tractor and the Human Operator. As soon as the AI doesn't find a confident solution to overcome an unknown situation, the HIL service will be triggered and will contact the Human Operator to take over the control of the tractor. The HIL service will thus inform the human operator by sending a signal from the tractor to the HIL application, so that a connection can be established.

The integration of the HIL service in the tractor controller will be performed during the integration of the different applications on the tractor controller. For the integration, it needs to be known what kind of threshold the AI will set to trigger the HIL service and how this data from the AI will be acquired. In cooperation with the AI developers, the interaction between the AI and the HIL service will be tested. Additionally, the triggering of the human operator by the HIL service needs to be implemented, integrated, and tested. Before this can be tested, a (remote and wireless) connection between the tractor controller and the human operator needs to be established, which will be based on the 5G infrastructure from partner EUR.

For testing purposes, the HIL service will receive a simulated signal, pretend to originate from the AI client (representing a non-confident solution). The HIL service will interpret this accordingly and will send out a trigger to the HIL application via the 5G infrastructure. As long as the 5G infrastructure is not setup, the HIL service will provide a simple out (e.g., console output) to validate that the triggering is activated. The testing and validation of the sending of the HIL application triggering is closely related to the testing and validation of the 5G infrastructure.

3.3.4 INTEGRATION AND TEST OF HIL APPLICATION

In the context of the Agriculture use case, the HIL Application 's main functionality is assisting the Tractor AI when it encounters an obstacle it cannot bypass.

The Operators are on stand-by, waiting for a help event from the HIL Service. The HIL Service informs the human operator, who connects to an application on Oculus Quest 2, which not only triggers an event toast message, but also allows the acceptance of said request. Once it was decided which Operator will handle the situation, they will run the ISAR Client Application and establish a connection with the HIL Application. Once the peer-to-peer connect was established, the Server application's content will be streamed and displayed on the Oculus Quest 2. This data stream will enable the Operator to solve the problem.

The HIL Application will connect to the tractor controller and receive 2 main data streams via a webRTC interface: Video frames from the cameras mounted on top of the Tractor; as well as a bi-directional data channel for the tractor controls. Furthermore, the data channel will be used for receiving metadata describing the state of the Tractor.

The data stream itself will be tested by connecting the HIL Application via webRTC to a data channel that spoofs tractor and video controls. Furthermore, on-site visits will be planned where the functionality will be tested on the tractor itself. If necessary, we can also setup a VPN allowing us the connection to our partners framework. This step, however, has not been discussed yet. All functionality that does not require a live data stream will be tested through our QA department in-house that will ensure the functionality.

3.3.5 INTEGRATION AND TEST OF AI AND TRACTOR

Prior to the integration of the AI component on the tractor, the AI will be trained offline in the lab using a set of labelled training data. The training data is composed of sensory data collected by the tractor (image stream of the camera mounted on the tractor) associated with labels which are the control decisions (angular and linear velocities) taken by the human operator while maneuvering various obstacles. After achieving a certain performance level (in terms of accuracy) via offline training, the pretrained AI component will be integrated on the tractor.

For testing purposes, an obstacle will be placed in the path of the tractor. The tractor should start its autonomous driving. When the obstacle is detected by the tractor's obstacle detection system, a trigger should be sent to the AI component to provide control decisions to bypass the detected obstacle. The AI component takes the image stream generated by the cameras mounted on the tractor along with information about the detected obstacle as the input. The output of the AI component will be the linear and angular velocities along with the confidence of that decision. If the confidence is above a predefined threshold, the values of the linear and angular velocities will be sent to the tractor's controller. This procedure should continue till either the obstacle is bypassed successfully, or till the confidence in the control decisions inferred by the AI is below the predefined threshold. In situations where the confidence is below the threshold, the tractor should brake, and request human aid as per the Human-in-the-loop scenario discussed under UC1 in deliverable D2.1.

3.3.6 INTEGRATION AND TEST OF DLT AND TRACTOR

Vehicle automation is one of the main applications of industry 4.0 especially in smart agriculture where the vehicle e.g., tractor, operates in an autonomous manner. it is a challenge to maintain the trust between customers and administrators about the activities, productivity as well as end-to-end process. Most of current solutions are based on centralized architecture such as cloud computing where the data is prone to various security threats.

The emerging of DLT provides a transparency, immutability and trusted infrastructure for recording the IoT sensing data, then the users and administrators can query and manage the data access. We develop a distributed and trusted solution based on DLT to decentralize the current cloud-based solution and guarantee about the trust of recorded sensing data. In specific, the location and productivity data from agriculture tractor are collected and published to distributed ledger via DLT-based smart contracts, which have responsibilities to handle the data from DLT clients and do an accounting for the data to guarantee that the activities of tractors are correctly matched with the agreement plan of administrators as well as customers.

Practically, we develop the solution based on the Blockchain Ethereum network, and solidity smart contracts. The general system architecture is described as below:

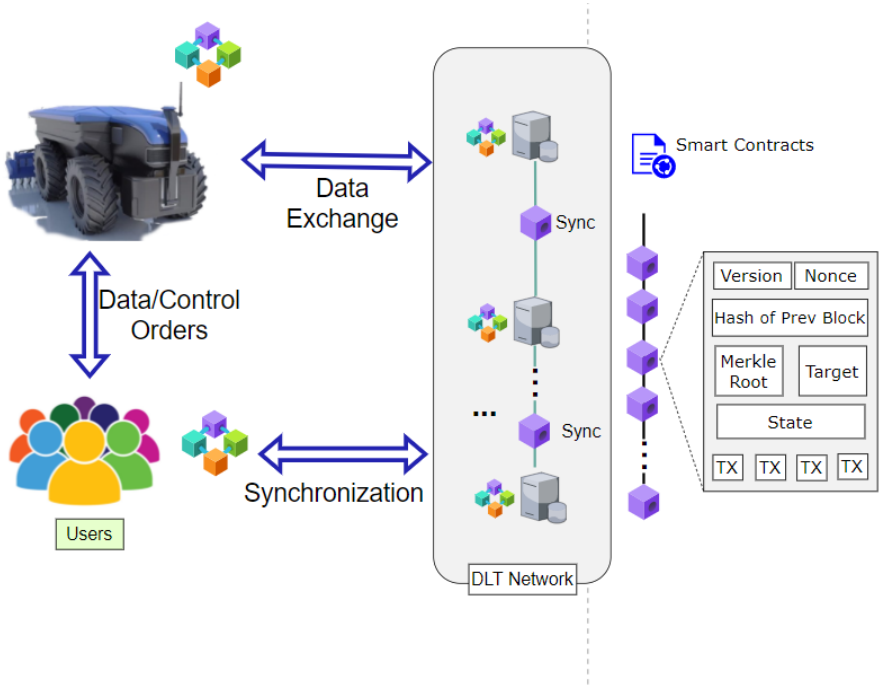


Figure 29: The general system design of DLT-based tractor monitoring

The Ethereum Blockchain tends to be implemented in 2 separated components. First, we implement a DLT client in the tractor controller to work with the GPS coordinator of tractor, then the DLT client form data in transaction from GPS receiver @AVLeTractor/AVLsensorBox: Navilock Products 61840 Navilock NL-602U USB 2.0 GPS Receiver u-blox 6 1.5 m Its integrated with the KI-gestützte autonome Maschinen im richtigen Umfang | NVIDIA Jetson AGX Xavier Running Linux Ubuntu 18.0; and publish to distributed ledger via Smart Contracts.

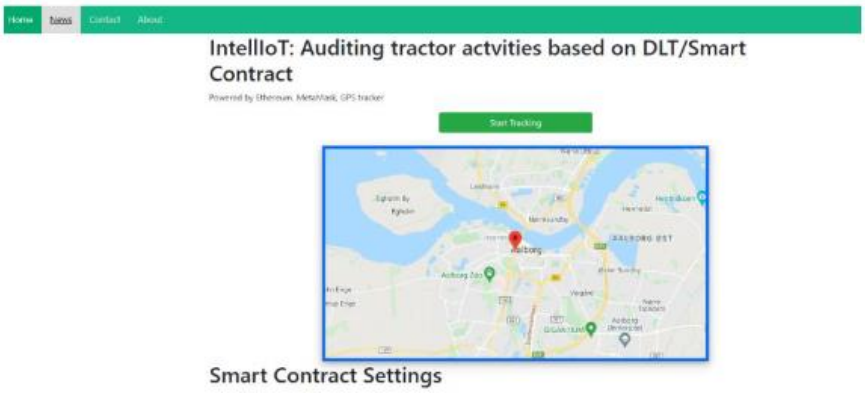


Figure 30: The Graphic User Interface (GUI) of Blockchain-based tractor monitoring scenario.

In order to integrate the Blockchain with the tractor controller, we are going to setup a Ethereum node on the tractor controller which running Linux operation system. We develop smart contract written in Solidity program to collect the data from sensor in tractor and process the gathered data.

The workflow of the integration system can be described as below:

- Step 1: The GPS coordinator collect the GPS data of the tractor, and send to the tractor controller.
- Step 2: The tractor controller, is installed with DLT-based client using Geth library, format the data into DLT transactions and forward to distributed ledger via specific smart contract address

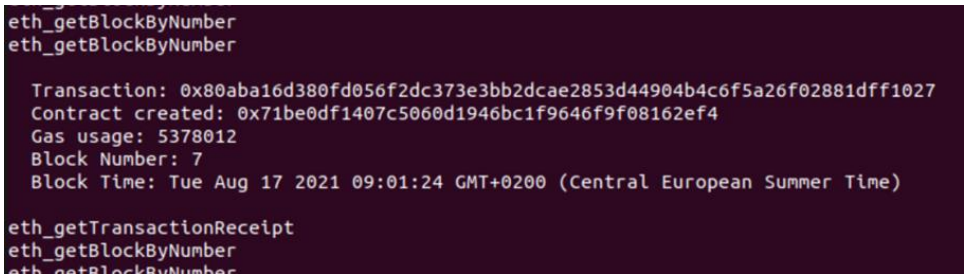


Figure 31: Unique address of the specific smart contract.

- Step 3: The GPS data of tractor is recorded and used by smart contract to check the realistic activity of tractor and accounting with the predefined plan of administrator.

3.3.7 INTEGRATION AND TEST OF 5G INFRASTRUCTURE

The 5G infrastructure is integrated with the IntellIoT infrastructure through the Edge orchestrator and the 5G Communication manager on the infrastructure side, and through the tractor controller on the 5G UE side. Data plane and management planes need to be integrated.

The 5G infrastructure is integrated to the IntellIoT data network by exposing its UPF internal component, as well as by connecting the 5G UE to the tractor component. First, connecting to the 5G UPF is performed as follows

- Step 1: the infrastructure machine generating traffic to the tractor needs to route traffic through the 5G UPF internal component; on the infrastructure machine, do:
  - ip route add TRACTOR COMPONENT IP ADDRESS via UPF IP ADDRESS dev eth0; sleep infinity;
- Step 2: the traffic will reach the tractor controller sub-network; a forwarding rule must be added to the tractor controller to reach the other components. On the tractor controller, do:
  - sysctl net.ipv4.conf.all.forwarding=1
  - iptables -P FORWARD ACCEPT

Traffic from the IntellIoT data network can reach the tractor controller components using the IP address of the specific tractor component.

NOTE: If the 5G UE gets disconnected and reconnects, it will receive a new IP address; the procedure will need to be repeated;

The 5G UE is integrated to the 5G tractor controller network by getting plugged via a USB3 cable. The 5G UE should be reachable via /dev/cdc-wdm0. A qmicli client tool is required to configure the 5G UE. The integration is done as follows:

- Step 1: turning the 5G UE on:
  - sudo qmicli -d /dev/cdc-wdm0 --dms-set-operating-mode="online"
- Step 2: configuring the 5G profile:
  - sudo qmicli -v -d /dev/cdc-wdm0 --wds-create-profile=3gpp,name=oai,apn=oai.ipv4,pdp-type=IP,roaming-disabled=yes
- Step 3: start the device
  - sudo qmi-network /dev/cdc-wdm0 start
- Step 4: mounting the 5G interface (wwan0)
  - sudo udhcpc -q -f -n -i wwan0

Traffic from the tractor controller can be sent to the 5G infrastructure via the wwan0 interface.

The management plane of the 5G infrastructure is integrated to the intelllot infrastructure through HTTP API exposed by the FlexRIC and FlexCN components. They will operate as docker containers on the same machine as the 5G Core under the same sub-network. Accordingly, integration of the 5G communication manager and edge orchestrator is done by adding ip routes to the 5G core machine. On the intelllot machine/component (5G Communication Manager or Edge orchestrator), do:

- Step 1: adding the ip route to use the right network interface
  - sudo ip route add 5GCORESUBNET via 5GCOREIP dev INTELLIOTCOMP_eth

Traffic or HTTP calls using the FlexRIC or FlexCN IP address is reachable by the IntellIoT component.

integration for both UC1 and UC3 will consist of two phases: a 5G Connection and Monitoring test, and a 5G slice configuration request phase.

The 5G connection & monitoring integration consists of the following phases:

- Step 1: programming of the spectrum parameters in the USIM and 5G Core
- Step 2: connection of the 5G UE to the 5G gNB and 5G Core and configuration of an IP address.
- Step 3: the 5G monitoring component will monitor and provide via its open API RAN statistics.
- Step 4: the Edge orchestrator and the 5G Communication Manager can query these RAN statistics via the Monitoring HTTP API.

The 5G slice configuration request consists of the following phases:

- Step 1: a radio resource configuration is received at the 5G Communication Manager requiring to open and configure a 5G slice with a given set of parameters.
- Step2: the 5G Communication Manager triggers the RAN configuration by sending instructions to the FlexRIC component and if required, the FlexCN component.
- Step 3: the FlexRIC component sends instructions to the OAI gNB to configure the radio resources for the specific slide and opens a radio bearer for the corresponding slice
- Step 4: the Monitoring component monitors the analytics of the 5G slice via the FlexRIC and FlexCN components;
- Step 5: the 5G communication manager and the edge orchestrator can obtain these analytics through the FlexRIC HTTP API.
- Step 6: data can be sent from the 5G UE to the 5G infrastructure on the configured slice.

3.3.8 INTEGRATION AND TEST OF TRUST COMPONENTS

In the agricultural use case testbed, the generic process of deployment and setup / configuration of SAP is followed, as detailed in subsection 2.4 above.

In addition to testing the standalone features of SAP, setup and testing of the integration between SAP and the rest of the trust components will also take place, per the process documented in deliverable D5.1, subsection 2.4.

4 UC2: HEALTHCARE

4.1 Hardware deployment

In UC2, the hardware deployment is composed of 3 main components (Figure 32):

- **Edge device**: a mobile phone Samsung Galaxy S21 5G with 128 GB of storage. There will be 4 units in the lab setup used by Philips employees. An additional 20 units will be provided to the patients enrolled by PAGNI.
- **5G MEC**: it is described in chapter 2.2 and will be used also as the Server in the lab setup. The simulated 5G MEC is a virtual machine at EURECOM with the following specifications:
  - Operating system Ubuntu 18.04.5 LTS.
  - RAM of 16 GB, which can be increased if necessary.
  - Storage disk of 80 GB.
  - Access to the Internet through IP address.
- **Server**: a server machine of which there will be one unit. As mentioned, in the lab setup the same machine that simulates the 5G MEC is used. Meanwhile in the production environment, the Server will be located at PAGNI.

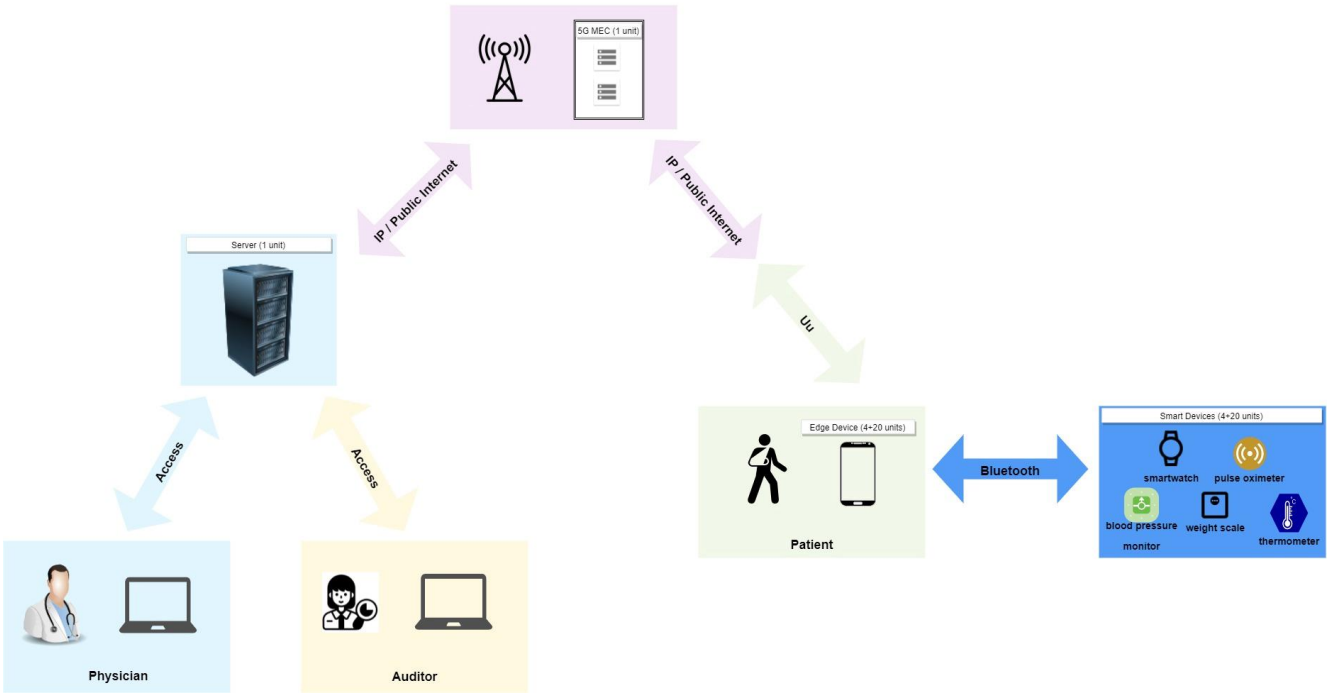


Figure 32: Hardware Deployment in UC2 - Healthcare

4.1.1 SMART DEVICES

As described in previous deliverables, the Sensors in UC2 are a smartwatch (Samsung Galaxy Watch4), a blood pressure monitor (Omron M7), a thermometer (Beurer FT95), a pulse oximeter (Beurer PO60), and a weight scale (Beurer BF600). The measurements from these devices will sync to the edge device, a smartphone Samsung Galaxy S21 5G with 128 GB of storage. The Local AI Component will be deployed on the smartphone to collect the data from the smart devices.



Figure 33: Sensors of UC2 that have been purchased at the time of writing.

4.2 Software / Apps deployment

In UC2, the software components will be deployed on the hardware as follows (Figure 34):

- On the edge device:
  - DLT Client.
  - Healthcare AI Models.
  - IAKM Agent.
  - IAKM Client.
  - Local AI.
- On the 5G MEC:
  - Global AI.
  - IAKM Edge.
- On the server:
  - DLT Manager.
  - Event Captors.
  - HyperMAS IDE.
  - HyperMAS Infra.
  - IAKM Server.
  - MTD Server.
  - Patients Data Repository.

- Security Assurance Platform.

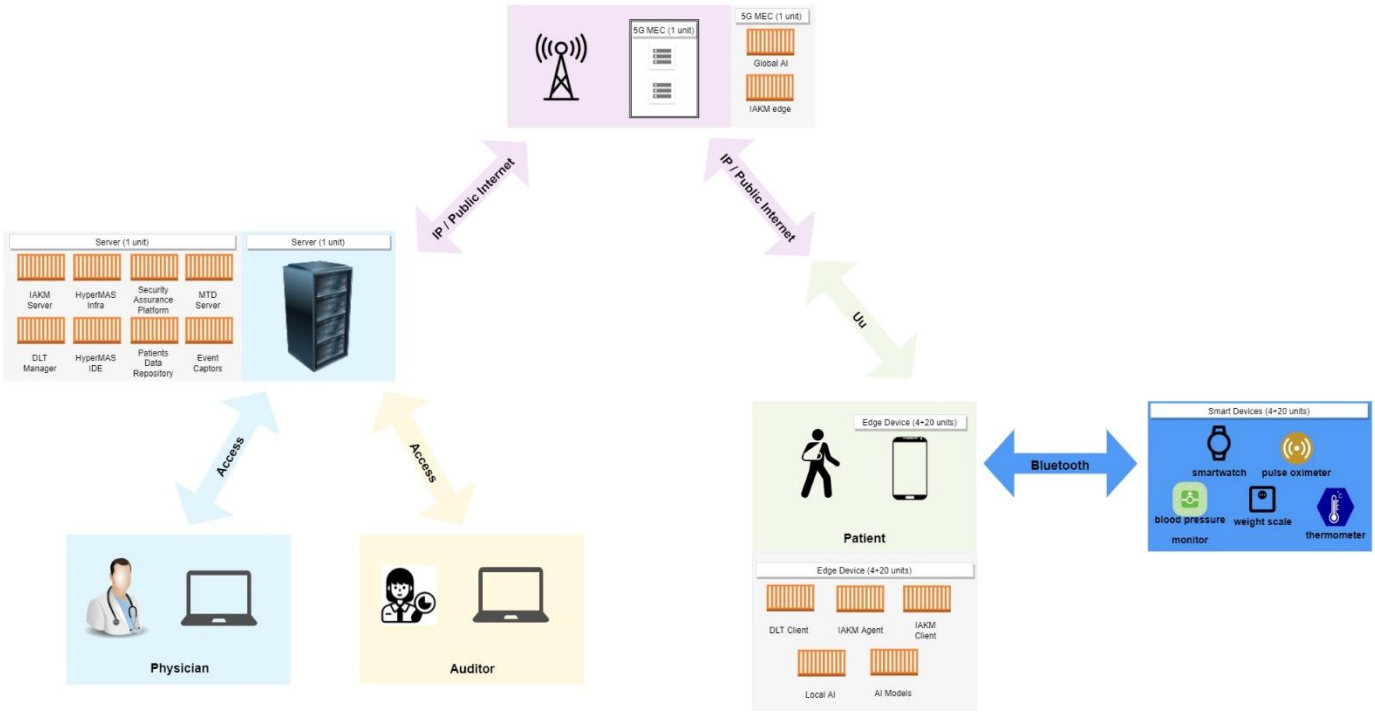


Figure 34: Hardware and Software Deployment in UC2 - Healthcare

4.3 Integration Tests

We will describe the integration tests of components that will be showcased in cycle 1 and the mid-term review, as discussed in the integration workshop.

4.3.1 INTEGRATION AND TESTS OF GLOBAL AI AND LOCAL AI COMPONENTS

As mentioned in previous deliverables, the Global AI Component exposes a secure REST API that is accessible by the Local AI Component. The API endpoints Swagger specifications are shown in Figure 35, Figure 36, Figure 37 and Figure 38. The endpoints expose the following functionality:

- Endpoint **/edge/models**, HTTP verb GET: download a Healthcare AI Model Component, passing its model and version identification numbers as HTTP request arguments.
- Endpoint **/edge/checkpoints**, HTTP verb GET: download the latest state (i.e., model weights) of a Healthcare AI Model Component, passing its model and version identification numbers as HTTP request arguments.
- Endpoint **/edge/ checkpoints/{task_id}**, HTTP verb POST: upload the state (i.e., model weights) of a Healthcare AI Model Component for a given training task (i.e., *task_id* which corresponds to a combination of specific AI Model, training round, and Local AI identification number), passing the Local AI Component identification number and training round number as HTTP multipart request body parts.
- Endpoint **/edge/task-results**, HTTP verb PUT: upload the results (i.e., training loss, training metrics) of a training round performed at a Local AI Component, passing the Local AI Component identification number and training results as a JSON message.

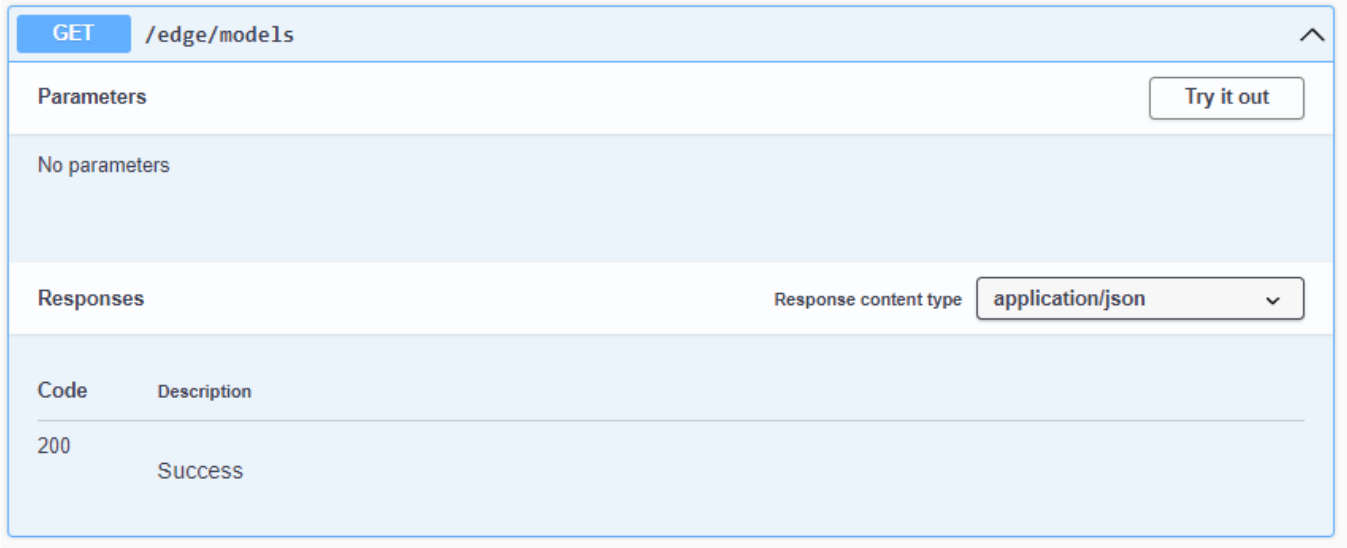


Figure 35: Global AI Component API endpoint for downloading Healthcare AI Models.

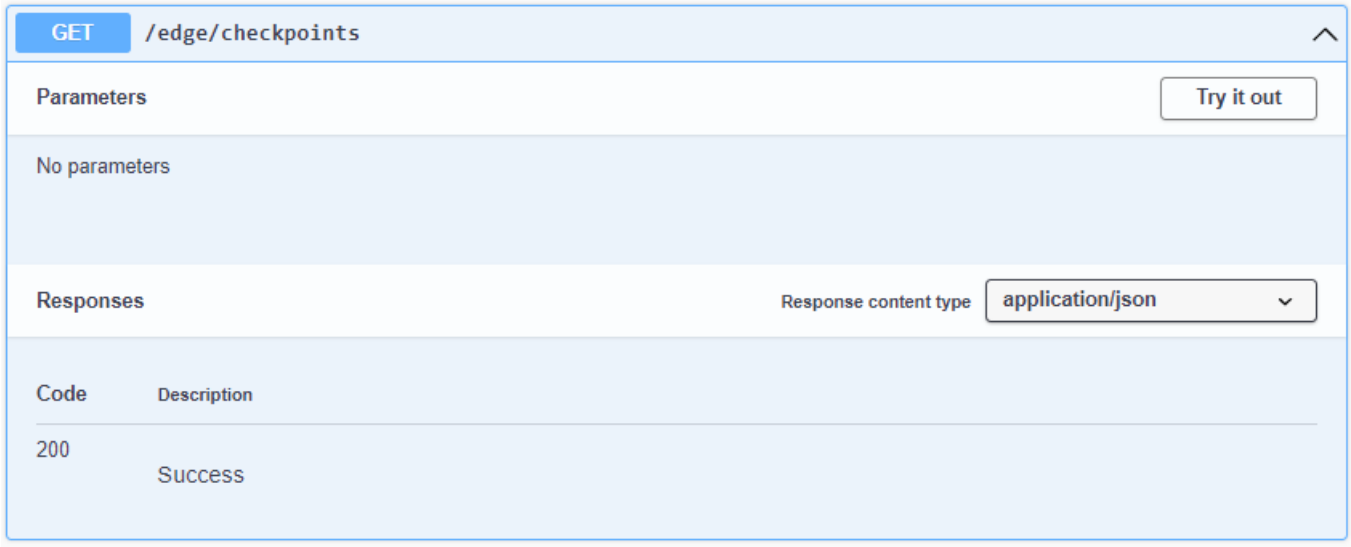


Figure 36: Global AI Component API endpoint for downloading the latest state (i.e., model weights) of a Healthcare AI Model.

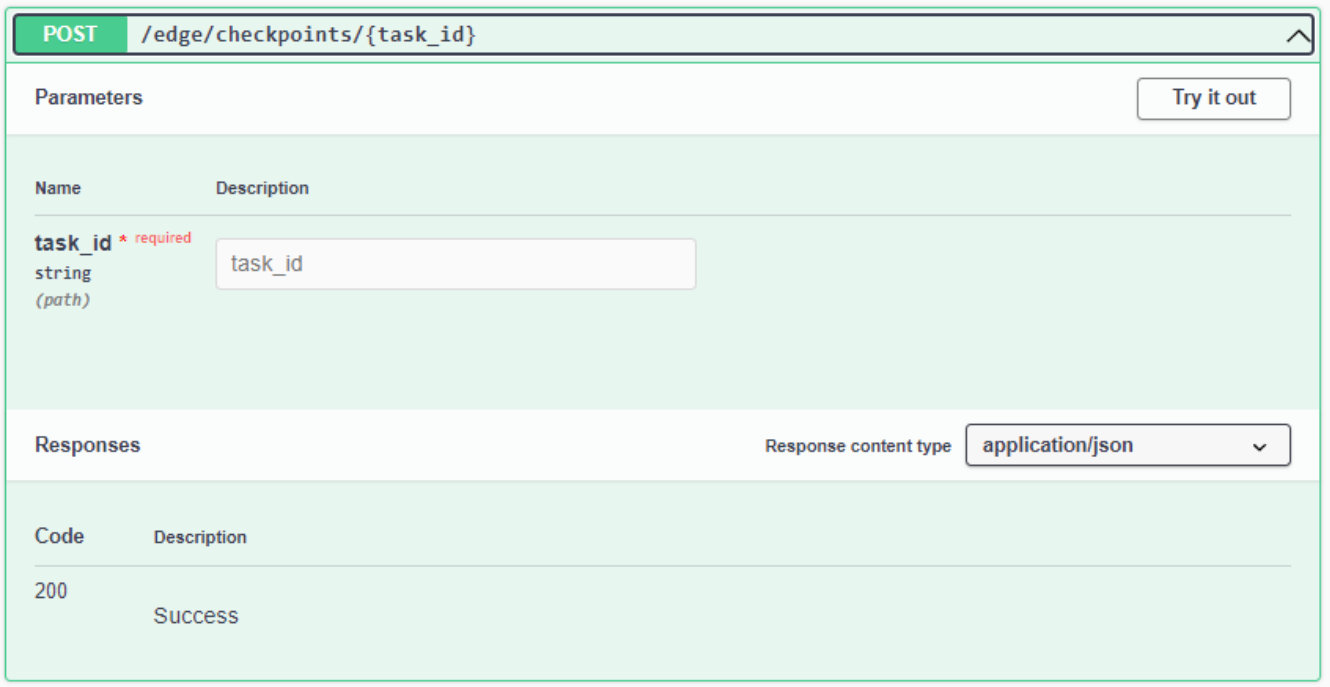


Figure 37: Global AI Component API endpoint for uploading the state (i.e., model weights) of a Healthcare AI Model.

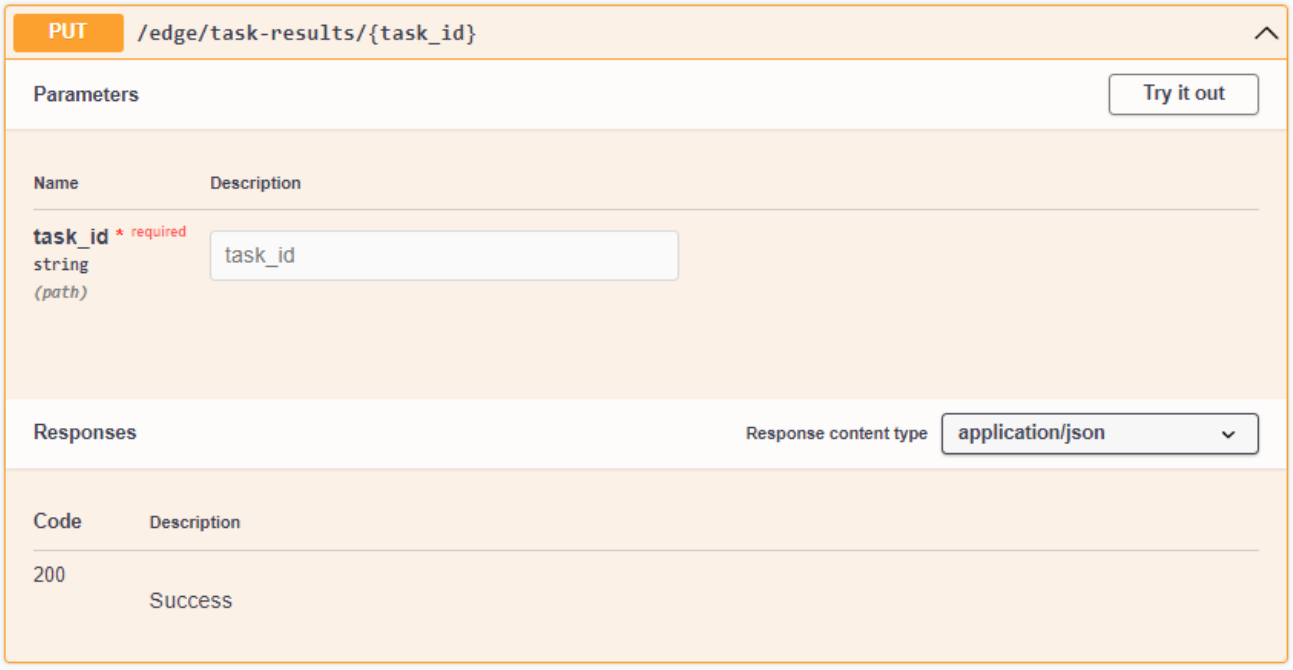


Figure 38: Global AI Component API endpoint for uploading the results (i.e., training loss, training metrics) of training rounds performed by a Local AI Component.

As part of the integration of the Global AI and Local AI Components, a simulation was performed. We use the MNIST fashion dataset and a base (Healthcare) AI Model Component, namely a neural network consisting of 3 TensorFlow Keras layers (one to flatten the images, followed by two consecutive densely-connected layers). Note that in the IntellIoT terminology of UC2, the base model used for federated training is called the Healthcare AI Model Component, but in this test the dataset and AI model are not related to the healthcare domain. Our goal with this choice was two-

fold: to perform a test with a model that executes quickly and to test the deployment of a new AI model that is different to the context-aware, long short-term memory network described in D3.2. The base model is trained on one class of the dataset and is subsequently deployed to 10 simulated Local AI Components. The deployment is simulated using one physical device in which the class of the dataset and the training parameters are changed to simulate multiple physical devices. The deployment was performed on the Samsung Galaxy S21 5G smartphone of the lab setup.

As mentioned, a portion (600 images out of 6000 images) of each of the classes of the dataset is deployed on each of the Local AI Components and for each federated training round. This simulates the extreme case in which each Local AI Component collects data completely different to other Local AI Components. Subsequently, 10 federated learning rounds are performed, adding a new portion of data at each round and so that at the end of the 10 rounds the Local AI Components have trained on all of the available data. Each of the 10 Local AI Components sends the state of the (Healthcare) AI Model at the end of the training round to the Global AI Component. Once all of the 10 Local AI Components have sent their state it is aggregated by the Global AI Component. The evaluation of the performance of the aggregated (Healthcare) AI Model is done globally on a test dataset, which simulates a curated dataset, and contains all of the classes of the MNIST fashion dataset[9].

The results of this integration test are shown in Figure 39 and Figure 40. Figure 39 shows the evaluation of the (Healthcare) AI Model at the Global AI Component, namely the loss (categorical cross entropy) and accuracy calculated on a test dataset composed of all the classes of the MNIST fashion dataset. As can be seen, the accuracy improves considerably between federated training rounds 2 and 3, after which it stabilizes to around 20%. Figure 40 shows the monitoring at the Global AI Component of the training performed by 10 Local AI Components using federated learning. Each of the 10 simulated Local AI Components trains for 100 epochs at each training round. At the end of each round, the model state (i.e., model weights) is uploaded to the Global AI Component, which aggregates the states of all 10 simulated Local AI Components. The training progress (loss) at the Local AI Components shows the same behavior across each Component and across federated training rounds. Note that in this integration test the numerical results are less important.

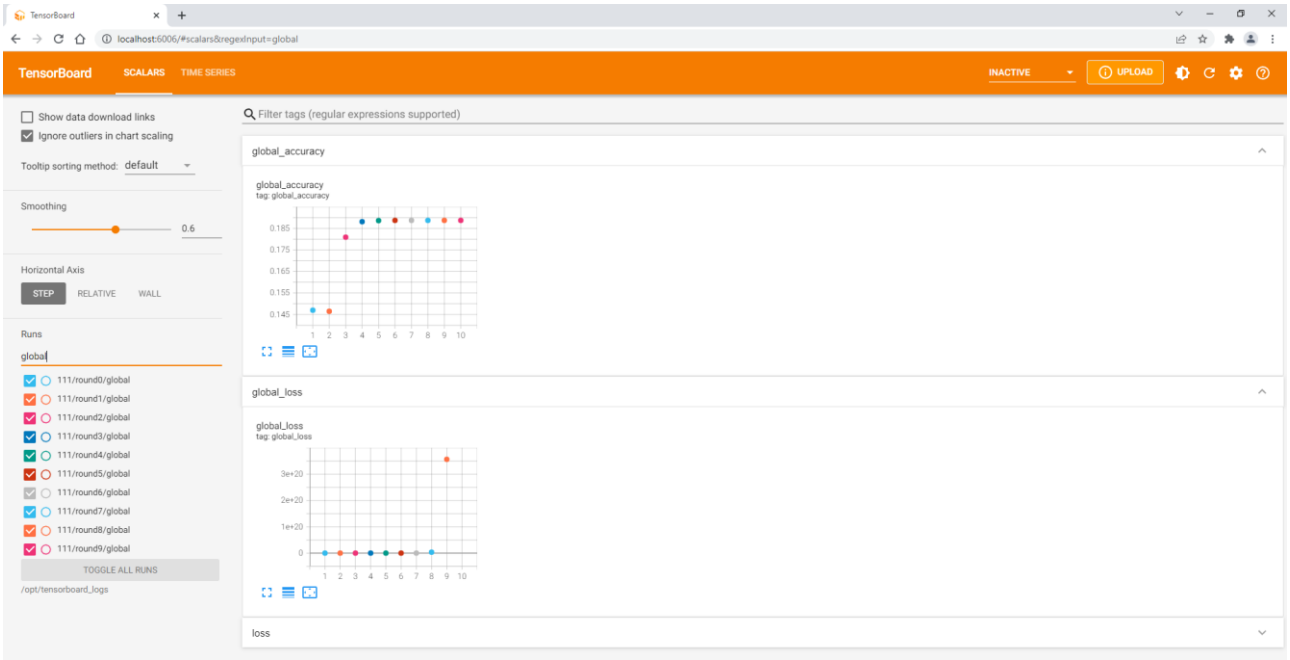


Figure 39: Global AI Component integration with the Local AI Component. Shows the evaluation of a (Healthcare) AI Model that is trained using federated learning, with aggregation of 10 simulated Local AI Components and 10 training rounds of 100 epochs each.

[9] https://www.tensorflow.org/datasets/catalog/fashion_mnist

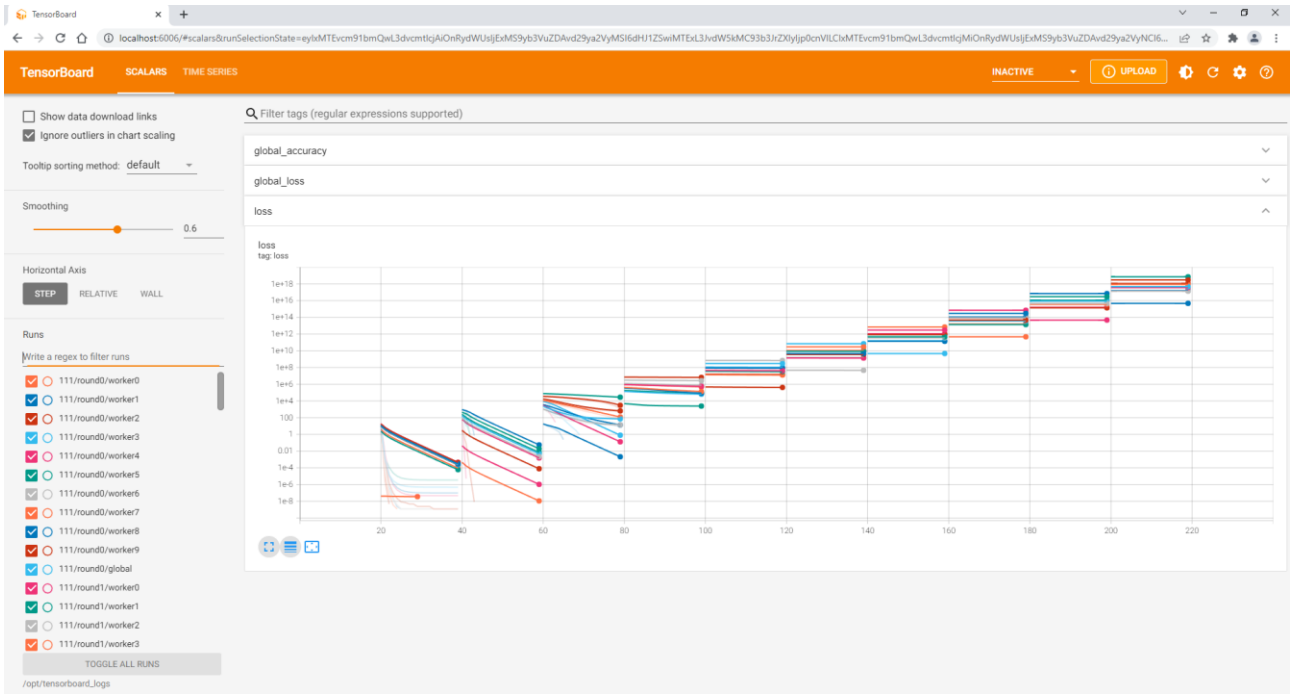


Figure 40: Global AI Component integration with the Local AI Component. Shows the monitoring of the training of the (Healthcare) AI Component by 10 Local AI Components using federated learning.

4.3.2 INTEGRATION AND TESTS OF GLOBAL AI AND IAKM

The integration test of the global IA and the IAKM for UC2 is accomplished by the following phases:

- Step 1: the GlobalAI authenticates to the IAKM via HTTP via webtokens. The IAKM adds the authenticated GlobalAI to the list of authenticated components in its database
- Step 2: the GlobalAI sends an AI request to the IAKM using the HTTP API and a defined AI model description.
- Step 3: The IAKM replies by providing the model if it has been found on the database; or HTTP 204 if no model is found.
- Step 4: If the IAKM receives a request from an unauthenticated GlobalAI, it replies with HTTP 401

A second integration test between the IAKM and the GlobalAI consists of identifying potential available workers, as follows:

- Step 1: the GlobalAI authenticate to the IAKM via HTTP via webtokens. The IAKM adds the authenticated GlobalAI to the list of authenticated components in its database
- Step 2: the GlobalAI sends an AI request to the IAKM using the HTTP API and a description of the AI training context.
- Step 3: The IAKM replies by providing the unique ID of the identified nodes if there are ones found on the database; or HTTP 204 is no model is found.
- Step 4: If the IAKM receives a request from an unauthenticated GlobalAI, it replies with HTTP 401

4.3.3 INTEGRATION AND TESTS OF HYPERMAS AND HMI/OTHER

The integration of the HyperMAS within UC2 is accomplished through two components. The first component is the HyperMAS Infrastructure which is used to expose Thing Descriptions (TDs) of relevant services and to launch and run configured agents that make use of these services. The other component is an application that permits the creation and configuration of agents in a no-code environment. Instead, the agents' procedural knowledge is represented using graphical building blocks in this component; this representation is subsequently converted to the Jason agent programming language in order to be able to run the agents. The blocks represent basic Jason operations as well as operations on services, provided their TDs are exposed on the HyperMAS Infrastructure. These components are detailed in Deliverable D3.1.

In UC2, the role of these components is to create and run an agent that alerts human users by sending textual messages. For the first integration tests on top of the components from D3.1, we have created an application that uses the Twilio Web API to send text messages to given phone numbers. To use this service, agents are required to provide the following information in JSON:

- Message: the content of the message.
- Phone1: the phone that sends the message.
- Phone2: the phone that receives the message.
- Account SID: the Twilio account of the agent.
- API Key: The API Key of the Twilio account.

We created a TD of the application and added this TD to a workspace of the HyperMAS Infrastructure to permit our no-code agent creation environment to access this functionality and thereby permit users to create agents that make use of the application.

This integration is functional and serves as an initial Agent-Human messaging system that can be easily extended with additional communication systems (and different communication channels) in the future. Regarding the integration with the rest of the UC2 system, TD-described APIs will be required not only of the messaging tools (e.g., Twilio, in this implementation) but also for the originators of messages.

Based on the TD of the Twilio Server, we will be able to perform all required integration tests:

1. Registration of the TDs for the Twilio Server with the HyperMAS Infrastructure. The TD shall be successfully registered with the HyperMAS Infrastructure, and it shall be possible to retrieve it given an entry point to an environment or workspace on the infrastructure.
2. Loading the TD from the HyperMAS Infrastructure to the EUP IDE. The EUP IDE shall successfully retrieve the TD from the Hypermedia MAS Infrastructure and render it as composable blocks within its interface.
3. Configuration of an Agent that uses the artifact, using the EUP IDE. The user shall be able to configure a meaningful program for an Agent that sends a request to the Twilio Server by providing its Twilio Account SID, its Twilio API Key, the phone number to send the message, and the phone number to receive the message.

Deployment and running of this Agent. The Agent shall be successfully deployed in an Agent runtime and execute its program. In this process, the Twilio Server shall send the message given by the agent to the phone number specified by the agent.

4.3.4 INTEGRATION AND TESTS OF TRUST COMPONENTS

In the healthcare use case testbed, the generic process of deployment and setup / configuration of the trust components is followed, as detailed in subsection 6.5 above.

In addition to testing the standalone features of the components, setup and testing of the integration between SAP and the rest of the trust components (IDS, MTDs, DLT) will also take place, per the process documented in deliverable D5.1, subsection 2.4.

During Cycle 2, integration of additional event captors will be provided (at patient premises, as well as the clinical backend), to allow for the detection and countering of malware threats (namely, botnet activity at the patient end, and ransomware activity at the clinical backend, per the trust scenario and associated key scenes defined for the use case).

5 UC3: MANUFACTURING

This chapter describes the physical deployment (chapter 5.1) and software and application deployment (chapter 5.2) for IntellIoT's manufacturing use case. Based on that, chapter 5.3 describes the stepwise integration of the software components.

5.1 Hardware deployment

This chapter describes all HW components used in UC3 demonstration and test setup. For each component, the purpose and relevant realization details are described. Constraints on the physical location are mentioned, where applicable. An overview of the physical arrangement is given in Figure 41, and additionally Figure 42 shows an image of a first mockup. Software components deployed on hardware components will be mentioned where relevant, please refer to deliverable D5.1 for detailed descriptions of software components. Chapter 5.2 gives an overview of software component deployment.

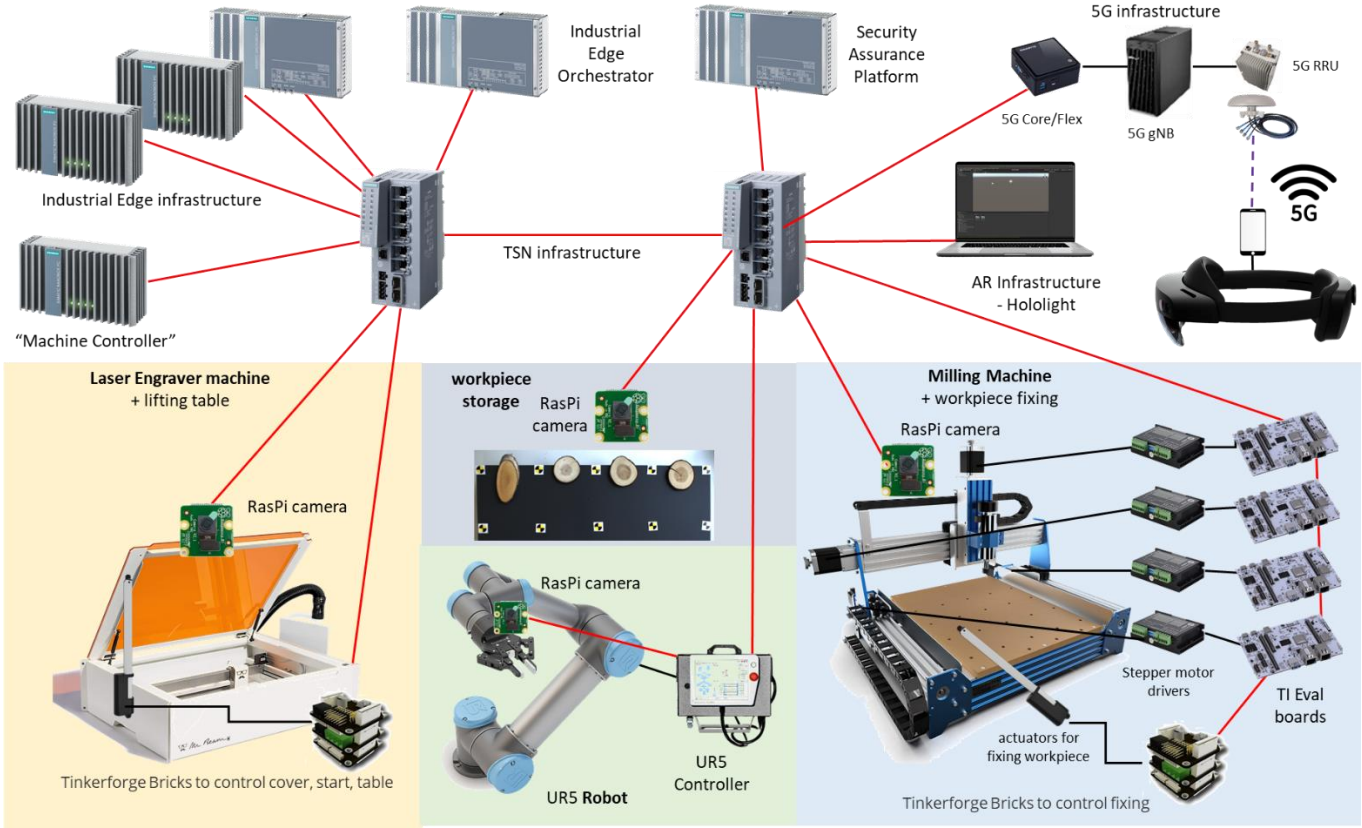


Figure 41: UC3 physical deployment overview

Figure 41 shows the physical components use in UC3 setup. Please note that the figure is meant to give an overview, thus it is not true to scale. The Ethernet / TSN network topology (red lines) is just showing which devices are connected through Ethernet / TSN. The actual topology is not fixed to the topology shown in Figure 41, rather the actual topology will be detected through TSN controller at runtime.

The lower part in Figure 41 shows the so-called "field level" of our plant, comprising a laser engraver machine (left), a milling machine (right), a workpiece storage (upper middle) and a robot (lower middle) to transport workpieces. This can be seen as a projection of the real setup shown in Figure 42. The only constraint on the physical deployment of the devices it, that both milling machine, laser engraver and storage must be in the operation range of the robot. Three RasPi cameras will be mounted to get bird's view images of both machines and the storage. A fourth, smaller RasPi camera will be mounted directly on the head of the robot to allow for real-time view on the gripper during HIL operation.

The upper part of Figure 41 depicts the control level. Physical arrangement there is hardly constrained, which means that the computational resources can be located wherever we find sufficient space in the lab setup.

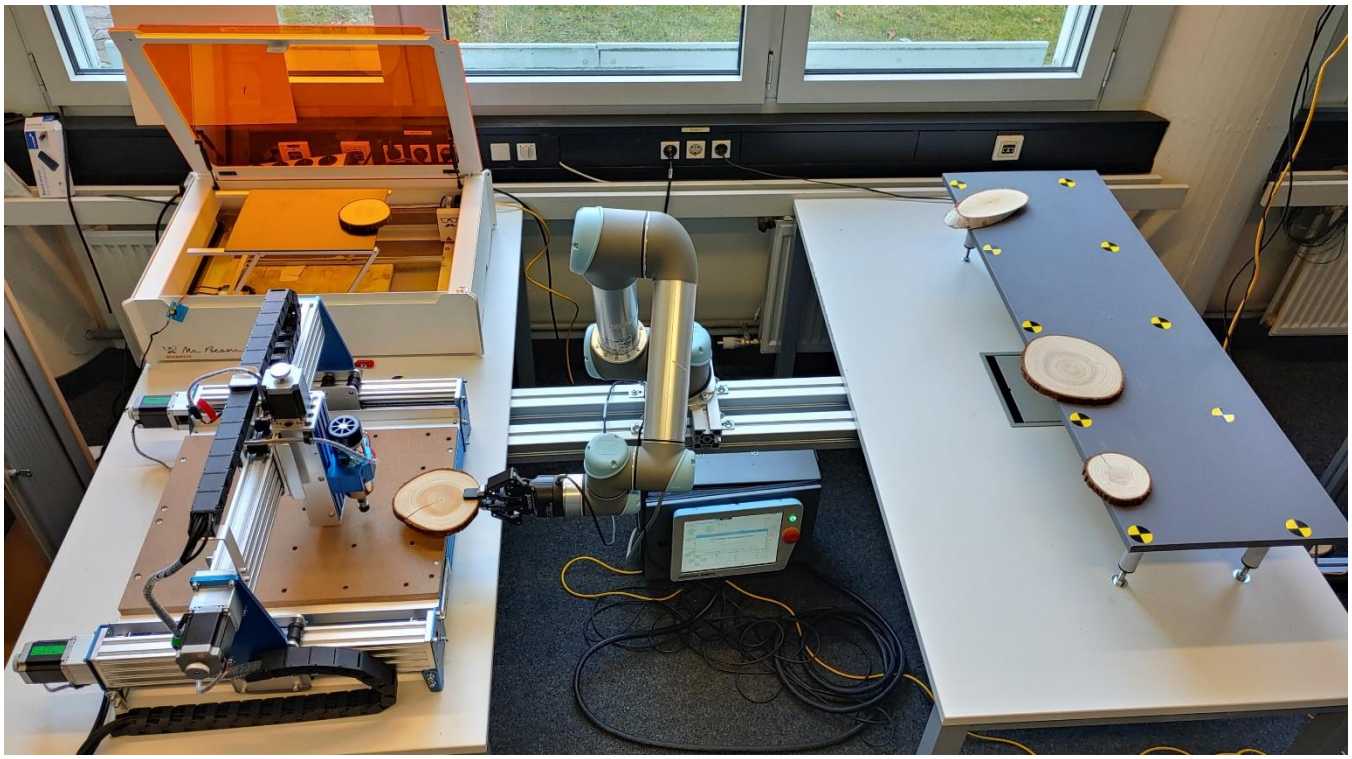


Figure 42: UC3 physical deployment

5.1.1 EDGE INFRASTRUCTURE

The Edge Infrastructure of the Manufacturing use case comprise several hardware components, on which a large part of the software components of the use case is deployed. The types of hardware components, which are counted as infrastructure are as follows:

- The Edge Server is a Linux-based Industrial PC (Siemens SIMATIC IPC627E, 64 GB RAM, 6 cores, 1512 MB SSD), on which the Industrial Edge Manager is deployed in a VM. Additionally, it runs all components, which provide services for the UC3, and which cannot be deployed on Edge Devices. In the next cycle, the Edge Server is supposed to host other Edge Managers too, for example, for managing the VMs on which the HIL Application is deployed. We have one Edge Server in the demo setup.
- The Edge Device is an Industrial PC on which Edge Apps can be deployed dynamically. It is managed by the Industrial Edge Manager (IEM), which is deployed on the Edge Server. The Edge Device is Linux-based but in fact the system access is protected. We have up to three Industrial Edge devices in the demo setup. 1 SIMATIC IPC427E (Microbox), 2 SIMATIC IPC227E (Nanobox).
- All other hardware components, controllers, IPCs, smart devices might interact with the Edge Infrastructure but are not considered as part of it.

The mentioned hardware components are execution environments for different kind of software: containers, VMs, native execution. A specific testing of the hardware components is not required. The functionality of the use case is guaranteed via test and integration strategies for the software.

The core software components of the Edge Infrastructure are:

- Edge Orchestrator
- Industrial Edge Manager (IEM)
- Industrial Edge Device (IED) (runtime)
- Edge Apps (various)

The Edge Orchestrator is a contribution of IntellIoT and is described in D4.1. The IEM as well as the IED runtime are COTS for the Industrial Edge product line of Siemens. Their interoperability is guaranteed by the vendor. The interoperability between Edge Orchestrator and those components has to be guaranteed.

The Edge Orchestrator controls the IEM in order to start/stop, deploy/remove edge apps on IEDs. Therefore, a web API is used. To guarantee the functionality over the course of the development process, unit tests are used, which cover the most relevant mechanisms. This is especially crucial as the IEM as well as the IEDs are regularly upgraded to new vendor software releases. Here, the risk the breaking of the API is high. The Edge Orchestrator has to be compatible with other UC3 IntellIoT services. The strategies for integration are discussed in separate sections in chapter 5.3 .

Edge Apps are different system applications, which are packaged as Edge Apps and which are executed on Edge Devices (here of type IED). For their purpose, separate specific test and integration strategies have to be defined. However, the ability of the interoperability with the Edge Infrastructure in order to execute Edge Apps has to be addressed.

The basis of Edge Apps for the Industrial Edge system is one or multiple docker containers, which are orchestrated with docker-compose. In combination with a proper configuration, Edge Apps can be on-boarded at the IEM and deployed on IEDs. We provided a guideline for edge developers at https://gitlab.eurecom.fr/intelliot-project/edge/apps/spec-iedge/hello-edge-app (see Figure 43). There are instructions for integrating the implemented services with Industrial Edge. Additionally, we provided a minimal viable sample project, which can be taken as a blueprint for own edge apps.

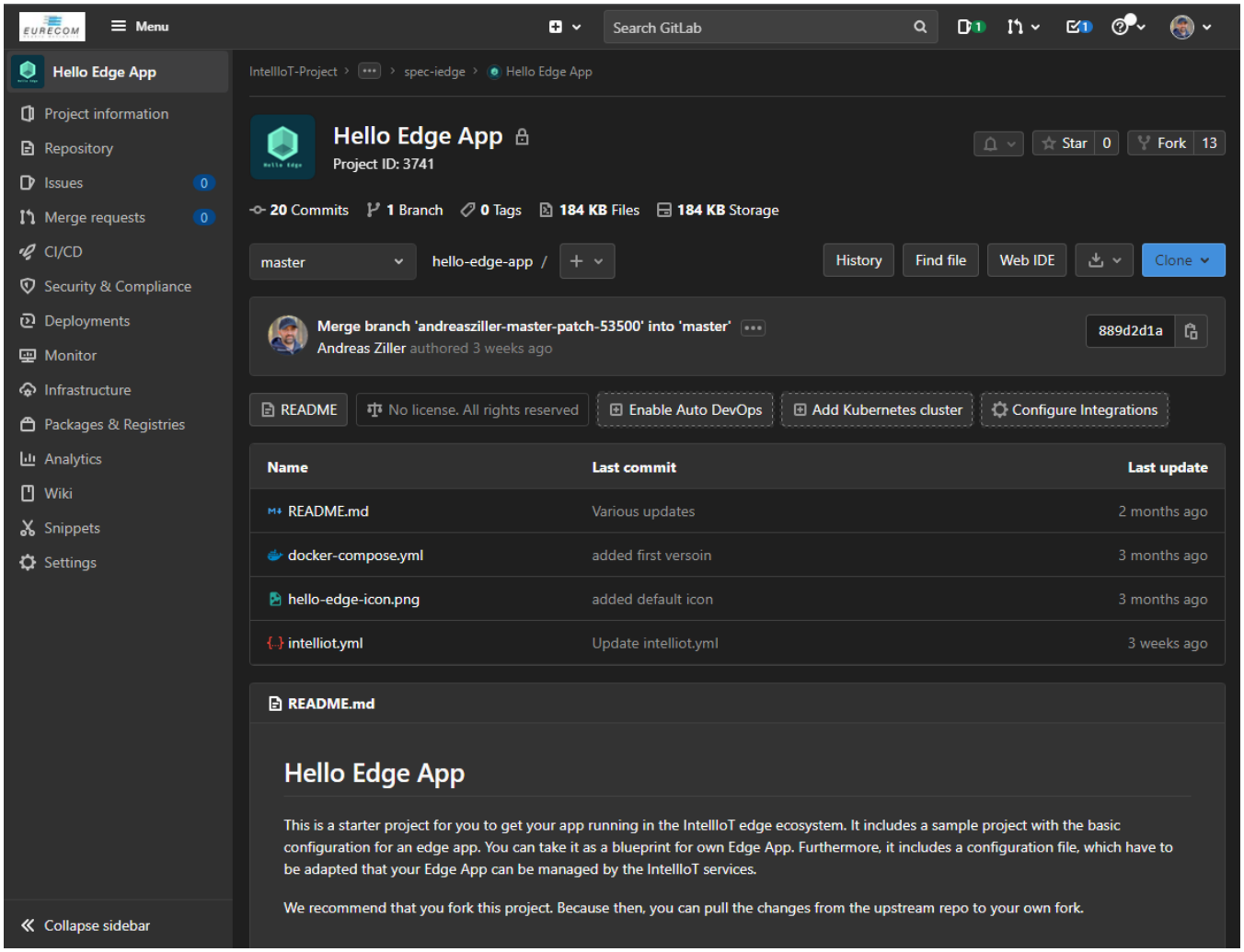


Figure 43: Guidelines and sample project as starting point or Edge App developers

DEPLOYMENT

The Edge Infrastructure consist of various software components, which will be updated during the course of the development process several times. An efficient and highly reliable deployment mechanism is very crucial. Automating the development as much as possible saves much time and allows shorter implement and evaluate cycles.

We automate the deployment of software components to a large extend. While doing this, we have to consider, that the deployment targets are located in isolated networks. A large share of software developers won't have direct access to those development targets. Accordingly, we have to introduce a process to deploy software artefacts of remote software developers.

The definition of the deployment process is still work in progress. However, we have developed the Edge Deployment Tool, which provides functionality for on-boarding Edge Apps in the Industrial Edge System as well as deployment of unmanaged software artefacts on systems, e.g., on the Edge Server.

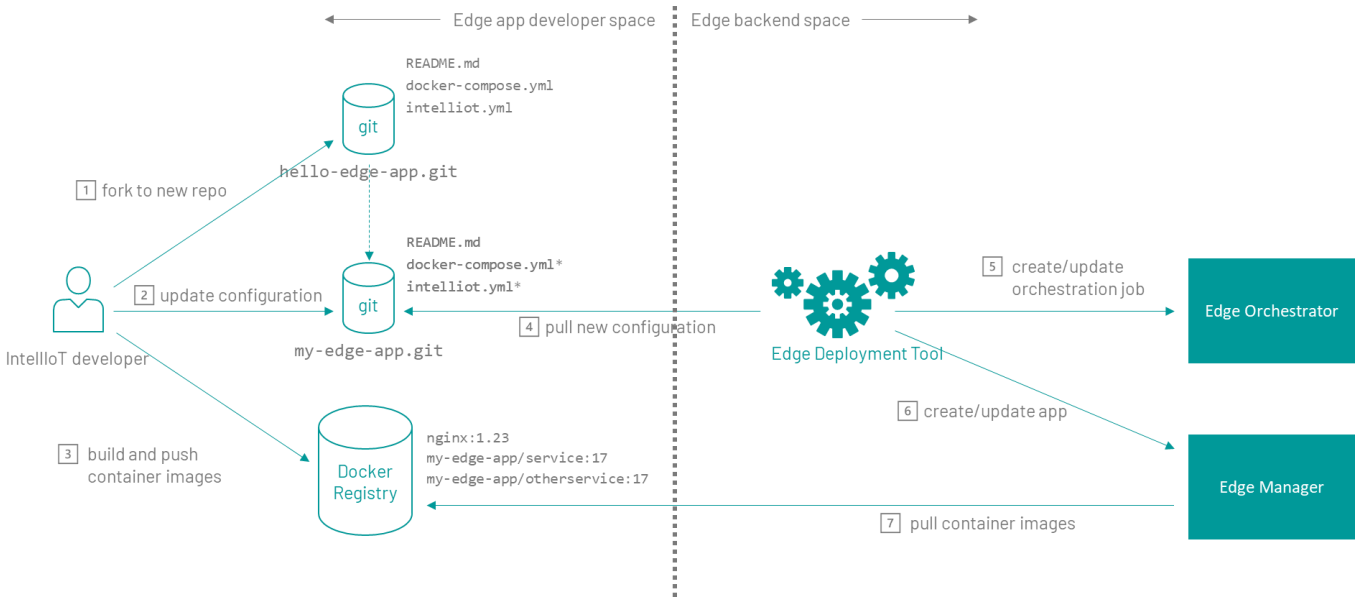


Figure 44: Workflow of onboarding and updating of an Edge App with the Edge Deployment Tool

Figure 44 shows the workflow for the development process of an Edge App. The Edge Deployment Tool relies on a deployment process, where all relevant specification files for the application to be deployed is included in a distinct Gitlab repo. Accordingly, the IntellIoT developer starts the development process by forking the sample app (Hello Edge App), which was described above, into a new Gitlab repo (step 1). In the 2nd step, the developer updates the forked repo with the specification for the new edge app. This includes, amongst others, the docker-compose.yml file for the container orchestration and a intelliot.yml specification file, which includes IntellIoT-specific and Industrial-Edge-specific configuration values. It is important to note, that the original application source code doesn't have to be shared during the deployment process. The app repo has to include only specification files. Accordingly, software artefacts are embedded in Docker containers. The pushing of Docker images happens in step 3. The correct images have to be referenced in the docker-compose.yml, which is included in the Gitlab repo.

The first three steps happen in the environment of the IntellIoT developer. They are timely decoupled of the deployment on target devices. When a deployment is triggered, the Edge Deployment Tool pulls the specification of the Edge App in step 4. According to the specification, it knows how to onboard the Edge App at the Edge Orchestrator (step 5) and at the Edge Manager (IEM) in step 6. The Edge Manager pulls the docker images in step 7, so that they are available once the Edge App is stared on an Edge Device.

The process of the deployment of unmanaged applications, which are not managed by the Edge Orchestrator is similar until step 4. The deployment on the target machine, e.g., the Edge Server, relies on standard mechanisms. Here, the Edge Deployment Tool, uses SSH to connect to the target machine, git to checkout the latest version of the app specification, and docker-compose to start the application. In order to automate this process, Ansible is used.

5.1.2 TSN INFRASTRUCTURE

The communication backbone in the manufacturing use case demo setup is a TSN-enabled Ethernet network. Figure 45 shows the physical deployment shown in Figure 41, with the Ethernet-devices highlighted. Green links indicate standard Ethernet links without special QoS requirements. Red links indicate links with tight QoS requirements, particularly for the real-time control of the robot. The bold red lines show the path from the machine controller to the TI eval boards controlling the milling machine axis, which put even more stringent requirements on the communication service. As mentioned in chapter 5.1, the topology shown is only an example and will be adapted according to the local requirements and the available TSN hardware. Exemplary, two Siemens SCALANCE TSN switches are shown in Figure 45. The machine controller shown on the left will contain a TSN-enabled network interface card from TTTech. The TSN controller will detect the topology and configure the devices accordingly.

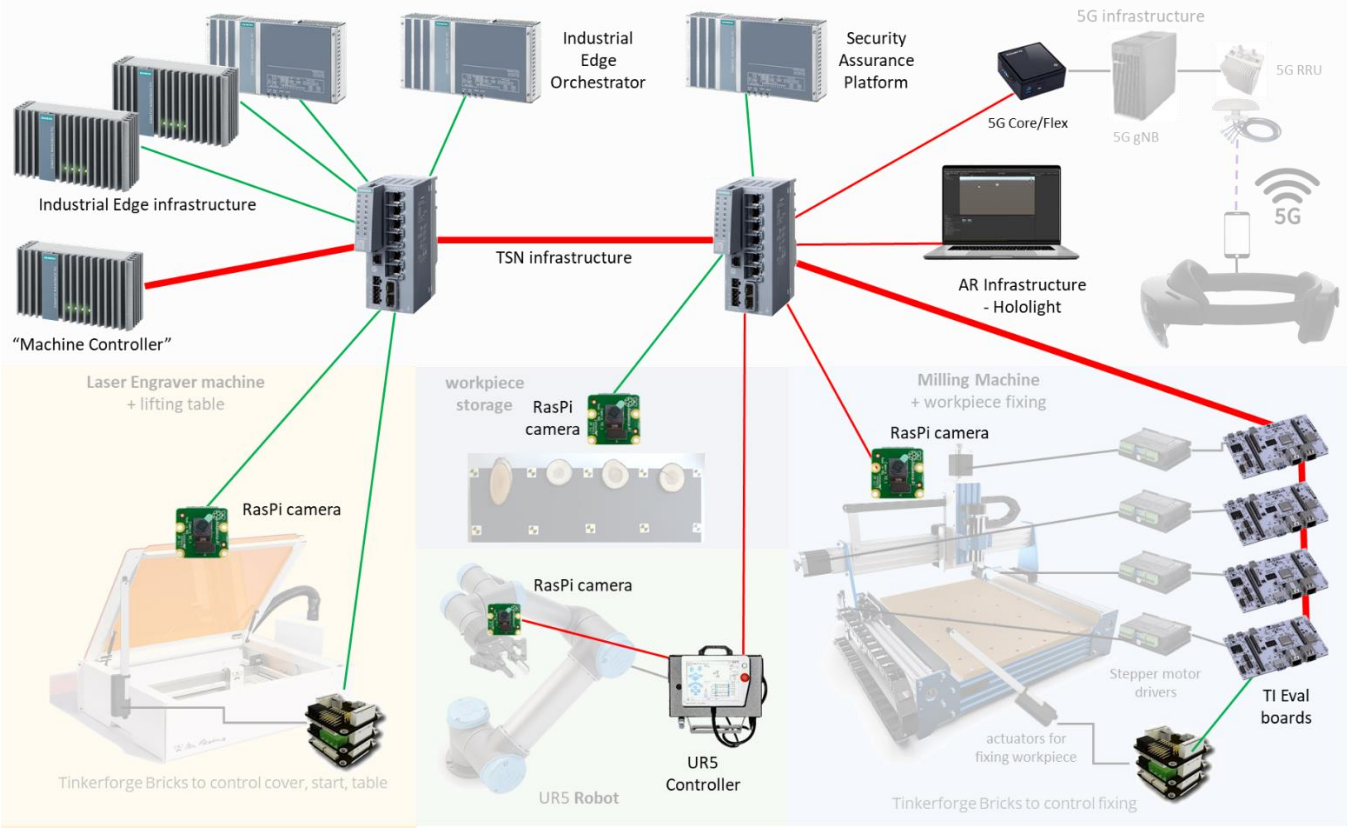


Figure 45: TSN backbone

5.1.3 5G INFRASTRUCTURE

The 5G infrastructure is basically identical to the one used in UC 1. It consists of an OAI 5G gNB with an OAI 5G Core and Mosaic5G FlexRIC and FlexCN dynamic 5G management functions. In contrast to UC 1, the complete 5G infrastructure as shown in Figure 23 and Figure 24 will be installed indoor for UC 3. The IntelIIot infrastructure is connected to the 5G infrastructure via the 5GCore over an ethernet connection. The Mosaic5G FlexRIC and FlexCN will provide APIs to the 5G Communication Manager located on the edge architecture.

Finally, the 5G HW architecture will be completed by a 5G UE (Figure 25) connected to the AR client to connect to the 5G infrastructure. The 5G UE dongle has a USB3 connector, which is connected to the AR client, i.e. the HoloLens.

5.1.4 AR INFRASTRUCTURE

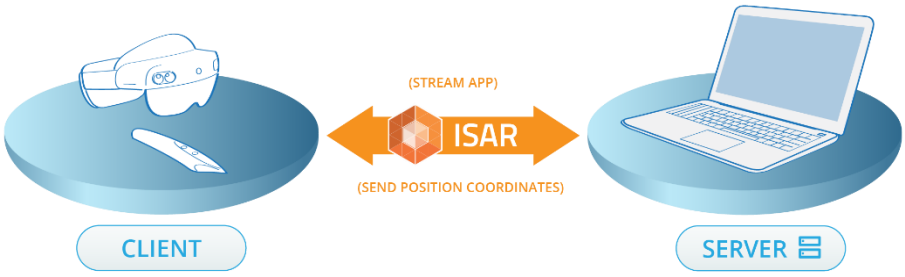


Figure 46. AR Infrastructure

The AR Infrastructure is similar to the VR one, as it is composed of two parties, the Client side (left side) and the Server side (right side) as demonstrated by Figure 46.

The Client is a simple xaml-based application – also containing the Stylus SDK – which is installed on AR glasses (HMD – Head Mounted Device), in this case the Microsoft HoloLens 2 (see Figure 47 for a explode-view of the device).

From a hardware side, the solution consists of 3 devices: HoloLens 2, HMI (Head Mounted Interface) which is clipped onto the lens, as well as the Holo-Stylus Pen. The battery-powered HMI and Stylus connect to each other and the Client on the HoloLens 2 via Bluetooth.

The Client then sends the spatial information to the HIL Application, therefore, allowing the definition of a new waypoint.

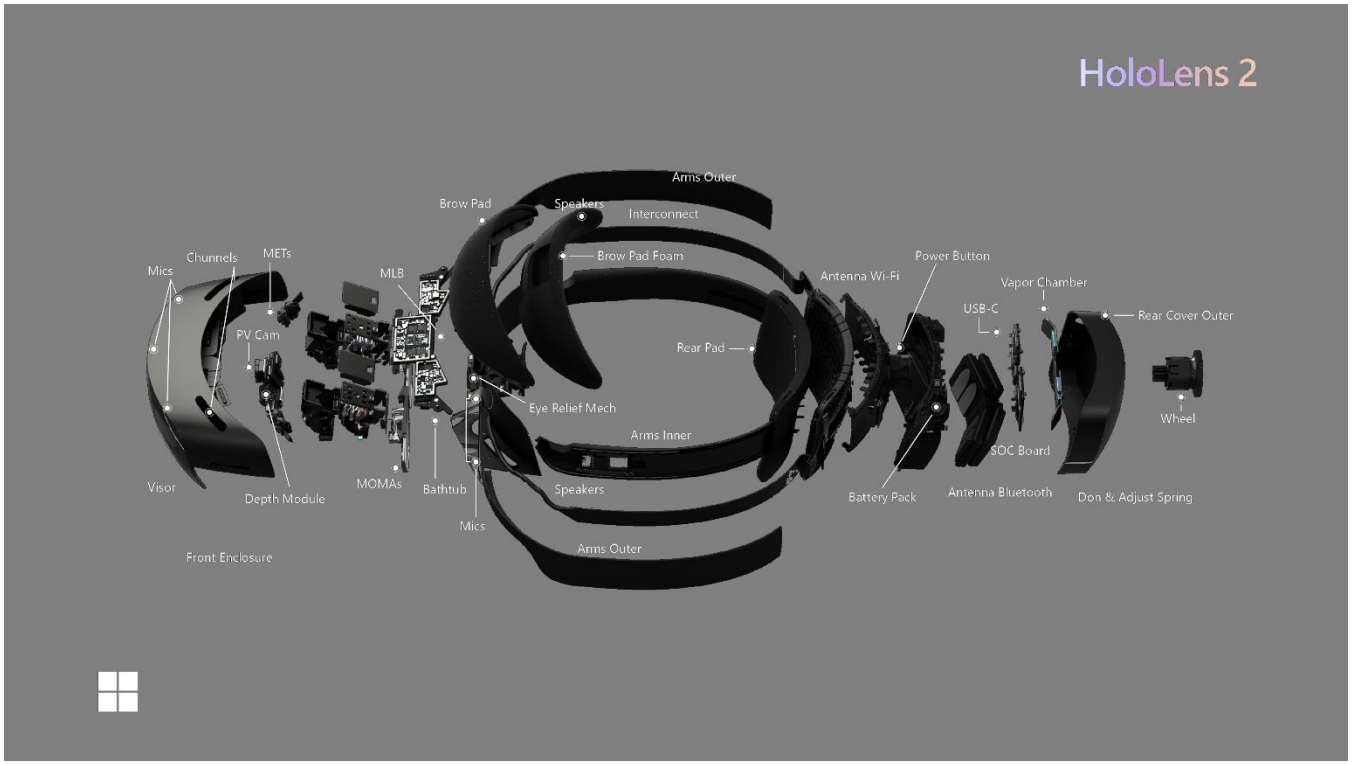


Figure 47. HoloLens 2 Exploded View[10]

[10] (Image from https://docs.microsoft.com/en-us/hololens/hololens2-hardware)

UC 1 and 3 will use the same Laptop as Server. For UC 3 it will host the AR HIL Application. The HW specifications are:

- Operating System (OS): Windows 10 Pro 64-bit (10.0, Build 19042)
- Memory (RAM): 16348MB (16 GB)
- Processor (CPU): Intel(R) Core(TM) i7-11800H @2.30GHz (16 CPUs)
- Graphics Card (GPU): GeForce RTX 3070 Laptop GPU

The Client and Server are linked to each other through ISAR. In the UC3 context, it will stream the data from the HIL Application to the HoloLens 2 and display it to the Operator while receiving positional coordinates from the Stylus, as well as a camera feed from be gripper.

5.1.5 ROBOT

The robot is an off-the-shelf *universal robot UR5*, see https://www.universal-robots.com/de/produkte/ur5-roboter/. It is used in two entirely different ways in the manufacturing use case: during normal operation, the robot-controller as described in D5.1, chapter 3.3.1 (please do not mix it up with the UR5 controller delivered with the robot) receives HTTP requests from agents and computes pre-defined poses from that, which it commands via the standard UR5 interface. During the HIL operation, the UR5 interface is taken over by the HIL application and the robot can directly be controlled by the operator. To grab workpieces, the robot is equipped with an off-the-shelf *Robotiq 2F-85* two finger gripper, see https://robotiq.com/de/produkte/adaptiver-2-finger-robotergreifer-2f85-140. The control of the gripper is directly integrated in the UR5 controller. Additionally, a *RPIZ CAM 5MP 160* Raspberry Pi Zero camera will be attached to the head of the robot, in direct proximity to the gripper. The images of this camera will be required for the operator to control the robot and grab a workpiece during HIL operation. This camera was chosen due to its 160° fisheye-objective and its tiny size, which is important to reduce the installation space on the robot head. A larger camera would limit the freedom of movement of the robot, which would cause tremendous effort to program the movement limitations to avoid a damage of camera and robot.

5.1.6 LASER ENGRAVER MACHINE

For the laser-based engraving, we use the off-the-shell product "Mr Beam II dreamcut" of the manufacturer Mr Beam (https://www.mr-beam.org, Figure 48). The integration of the machine into IntellIoT via Engraving App is described in section 5.4.2 in D5.1.

The Engraving App is implemented and working the Mr Beam laser cutter. It communicates via HTTP REST as well as web sockets with the machine. The support of the mechanical loading and unloading of workpieces as well as the automated push of the start button is planned.

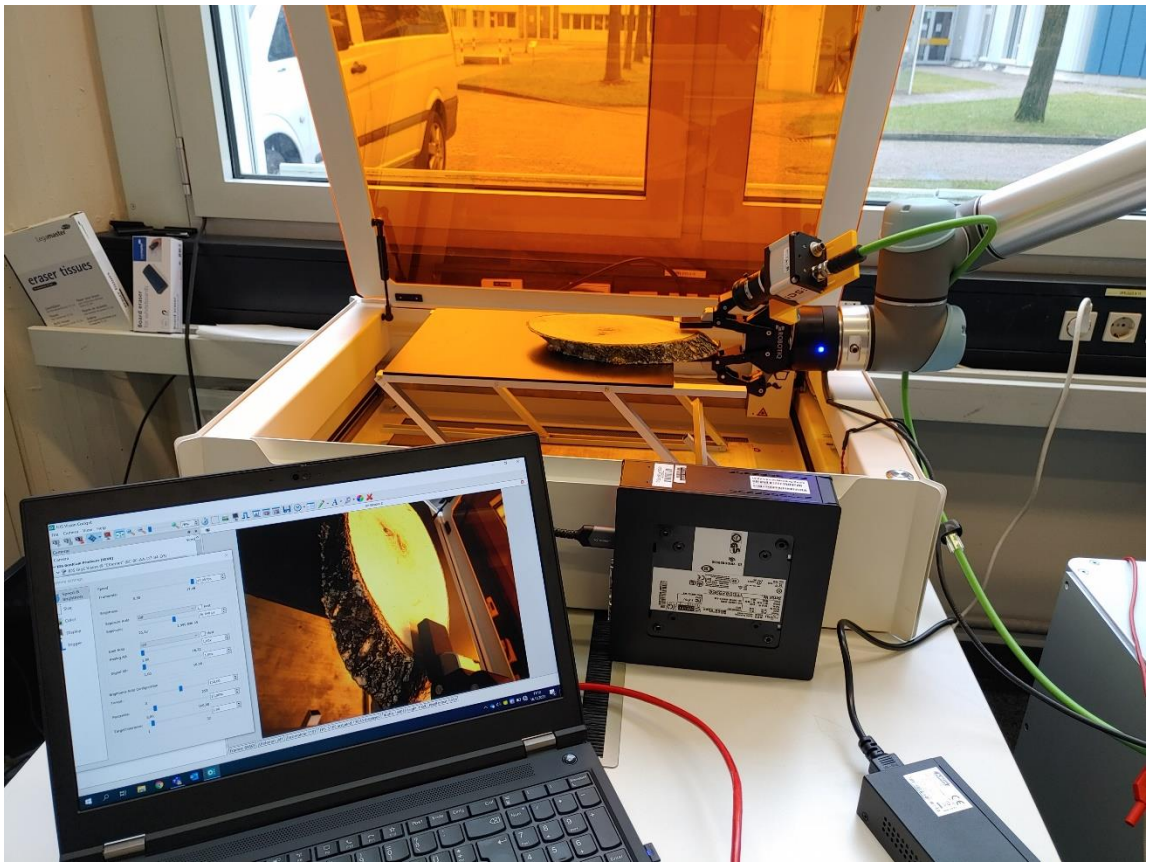


Figure 48: laser cutter with lifted workpiece

5.1.7 MILLING MACHINE

The milling machine hardware is an off-the-shelf "Genmitsu PROVerXL 4030" 3-axis table milling machine, see Figure 49. In the off-the-shelf version, it comes with a monolithic, micro-controller-based control electronics box. This box controls four standard NEMA stepper motors – four are required, as the y-axis is driven by two motors. To enable the demonstration of distributed control in the edge, using TSN to ensure control accuracy, we replace the complete off-the-shelf control box. In the IntellIoT UC3 demonstrator, the control functionality, i.e., computing milling trajectories from an image and computing movement commands to the stepper motors, will be delivered by edge apps, thus do not require computation hardware statically assigned to the milling machine. However, to reach optimal control quality, the edge device computing milling machine control should be attached to the TSN network via a TSN-enabled network interface. To make this transparent, one edge device is named "machine controller" in Figure 41. Still, this device might hold additional edge apps, as well as several devices might be available for the edge orchestrator to choose as machine controller. The control commands will be received from the TSN network via three "ti Sitara AM64x" eval boards, one per axis, which drive the stepper motors via an ACT DM542 stepper motor driver each.

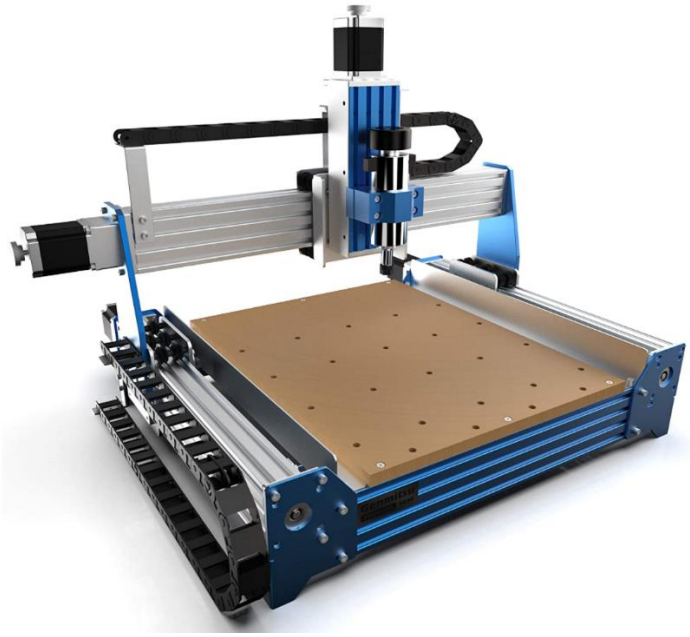


Figure 49: milling machine Hardware

While the off-the-shelf version is designed for manual positioning and fixing of the workpiece, we aim for a fully automated process in IntellIoT. Therefore, we add two additional hardware components to the milling machine: A camera to deliver pictures to the AI, required for computing the grab spot and engraving area, and an electrically driven clamping mechanism.

The camera is a Raspberry Pi HQ camera module, attached to a Raspberry Pi zero. With that, we obtain the possibility to realize a REST-based interface to request images directly at the camera.

The clamp mechanism is based on a Actuonix P16 linear drive with 200mm range and potentiometer feedback. See Figure 50 for a design sketch and an image of the prototype. Control commands and feedback are realized through a "tinkerforge Master Brick 3.1", complemented with an "industrial dual relay bricklet", an "industrial dual analog in bricklet" and an "Ethernet master extension". Tinkerforge[11] provides software libraries to remotely control these bricklets, which can directly be integrated in the edge app controlling the machine.



Figure 50: milling machine clamp mechanism – design and prototype

5.1.8 WORKPIECE STORAGE

[11] www.tinkerforge.com

The workpiece storage, as shown in Figure 51, is just a simple elevated table, which enables the robot to grab the workpieces from the front, which means the gripper goes beyond and below the workpiece, see Figure 52. The black-yellow markers will be mounted at defined positions and support the calibration of the orientation of the camera. The camera is, same as in the milling machine, a Raspberry Pi HQ camera module, attached to a Raspberry Pi zero. With that, we obtain the possibility to realize a REST-based interface to request images directly at the camera. The camera will be mounted vertically in the middle above the workpiece storage to provide an overview image as shown in Figure 51. This will be used by the AI components to find grab spots and engraving areas.



Figure 51: Workpiece storage top view

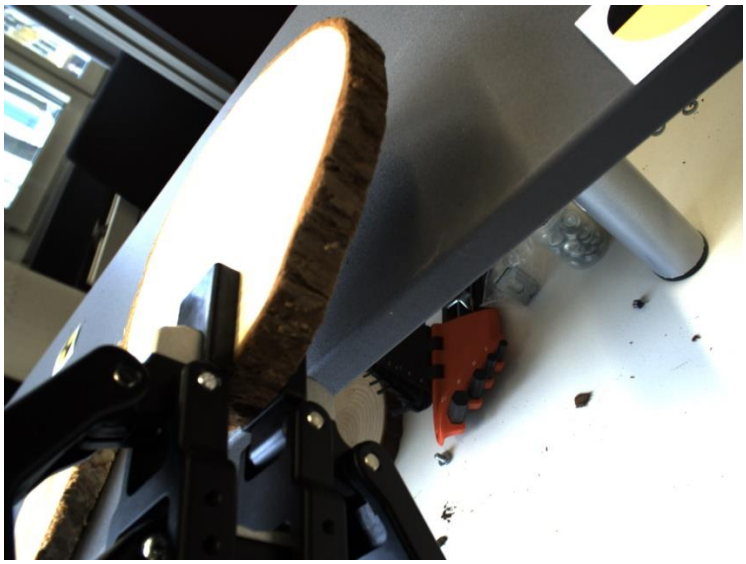


Figure 52: Robot grabbing workpiece from workpiece storage

5.2 Software / Apps deployment

Figure 53 shows the deployment of edge apps and other software components on the devices presented in chapter 5.1. Please refer to WP 3/4 deliverables for descriptions of the software components. In Figure 53, edge apps are depicted in green. Edge orchestrator dynamically deploys instances of edge apps on edge devices based on required and available resources. Some edge apps, in the case of IntelIIoT's UC3 the milling machine controller, bring special constraints, in this case the requirement for a TSN-enabled network interface. Thus, milling machine controller is depicted in orange and deployed on a device fulfilling the constraints, here called the machine controller hardware. As said, it is still possible to deploy additional apps on the machine controller hardware, and to deploy the milling machine controller on any device fulfilling its requirements.

Some SW-components cannot be containerized or for other reasons not managed by edge orchestrator. These are depicted in red in Figure 53 and located at the respective dedicated hardware.

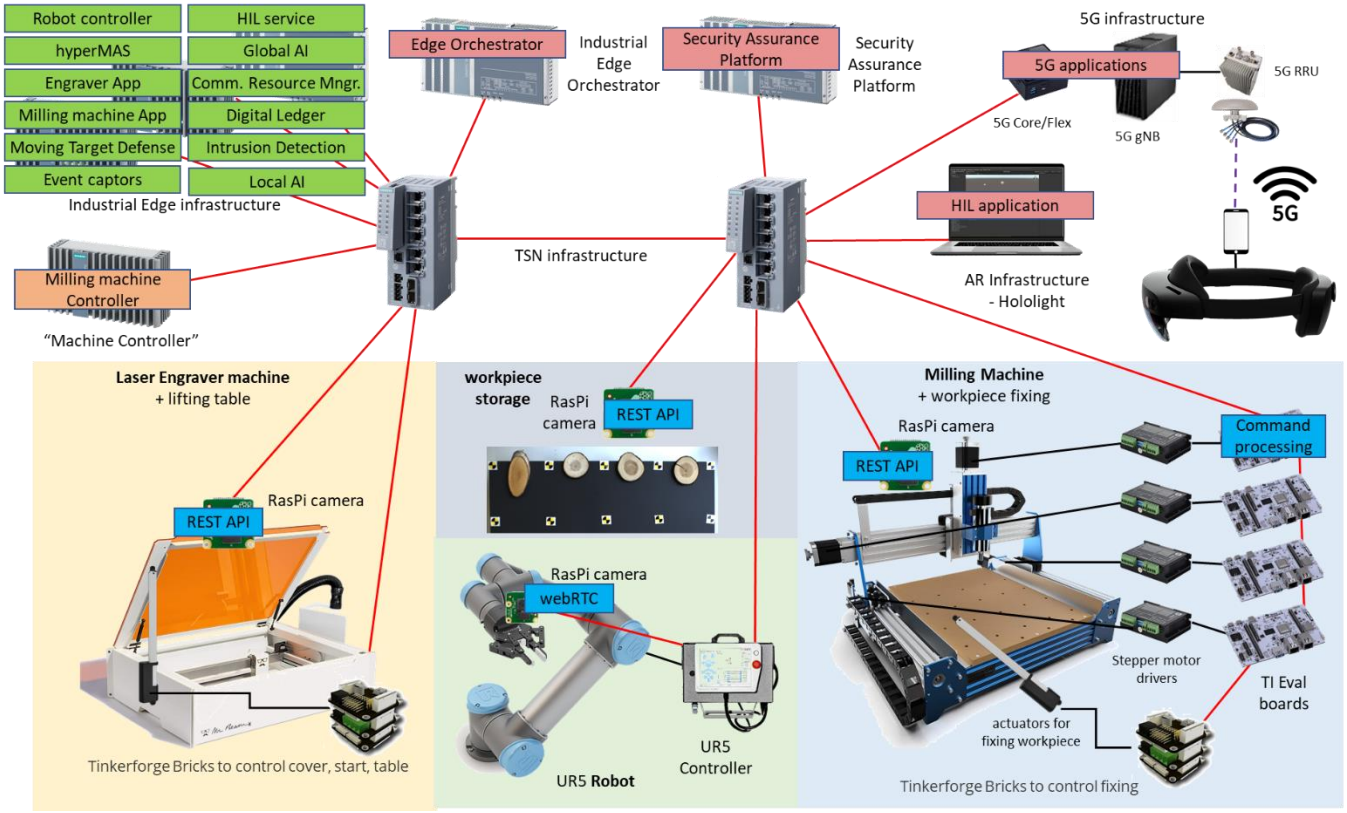


Figure 53: Software/Apps deployment

Additionally, Figure 53 shows some minor software components in blue. These are for example small interface components enabling HTTP-based access to the RasPi cameras or the SW-component interpreting TSN packets from milling machine controller and triggering GPIOs of the TI eval boards accordingly. These minor components are not described in extra chapters but mentioned where they are used.

5.3 Integration Tests

This chapter describes the iterative incremental integration and test of use case 3 specific software components. The described integration steps will build upon the pre-tested cross-UC components and interfaces described in chapter 2 and the deployed hardware infrastructure described in chapter 5.1.

The integration tests described in this chapter will cover most, but not yet all components of the overall IntellIoT framework. The components integrated in cycle 1 will possibly not yet include all features and fulfil all KPIs. With that, we also expect not all interface features being available or fully implemented. However, the basic functionality required for the test and demonstration of selected key scenes from D 2.1 will be available. Basis for the selection of key scenes and components is the description of cycle-1-KPIs given in D 2.2. Updates for most components and interfaces are expected for cycle 2.

5.3.1 INTEGRATION AND TEST OF HYPERMAS AND ENGRAVER MACHINE

- Goal: HyperMAS send a job, Engraver works on it
- TD of Engraver ready and integrated, but never talked to real machine

The Engraver (laser cutter as well as milling machine), the Robot Controller and the AI expose TDs that the agent, present on the HyperMAS infrastructure, can read.

When the agent receives the message to engrave from the customer, the agent is able to control the Robot Controller and the AI, based on their TDs, in order to move the workpiece to the engraver. The agent is then able to control the engraver based on its TD, in order to engrave the message.

Based on these TDs, we will be able to perform all required integration tests with the engraver:

1. Registration of the TDs for the engraver, the robot controller and the AI with the HyperMAS Infrastructure. The TDs shall be successfully registered with the HyperMAS Infrastructure, and it shall be possible to retrieve them given an entry point to an environment or workspace on the infrastructure.
2. Loading of the TDs from the HyperMAS Infrastructure to the EUP IDE. The EUP IDE shall successfully retrieve the TDs from the HyperMAS Infrastructure and render them as composable blocks within its interface.
3. Configuration of an Agent that uses the artifacts, using the EUP IDE. The user shall be able to configure a meaningful program for an Agent that controls the robot and the engraver in order to engrave a workpiece. It shall be possible to extend this Agent with functionality from other registered TDs (also across use cases, e.g., with the messaging service from UC2)
4. Deployment and running of this Agent based on a goal provided by the customer. The Agent shall be successfully deployed in an Agent runtime and execute its program. In this process, the robot controller can move the robot to the engraver and the engraver can engrave the message.

Because of the standardization of TD and our recent interactions about the HTTP API of the robot controller and the engraver, we do not foresee any issues with these tests. After these tests, the next step will be to integrate further services from other partners across IntellIoT.

5.3.2 INTEGRATION AND TEST OF HIL SERVICE

The HIL service mediates between edge orchestrator, hyperMAS, MQTT broker, Operator, Robot controller and AI. Thus, it requires several integration test steps, but most of these steps are quite small and straight forward. As many other components need to be up and running before the HIL service can be completely tested, integration of the HIL service will be performed in a later step in the overall integration process. With the successful integration of the HIL service being done, the human-in-the-loop scenario for use case 3 will be almost completely up and running. As the steps are massively depending on each other, we plan to stick to this given order for integration tests, even if we can start with the particular steps whenever all required interfacing components are available and previous steps are done. Figure 54 gives an overview of the sequence described in the following:

1. This step can be executed at an early stage and only needs edge orchestrator as test partner. To initiate this test step, we start the HIL service. On startup, it will request the orchestration of a configurable number (at least one) instances of HIL application from edge orchestrator. Edge orchestrator replies with one orchestrationID per instance. To check the success of this steps, we check the number of instantiated HIL applications via edge orchestrator's UI.
2. After startup, the HIL service will idle and wait for a help request coming from hyperMAS. Whenever such a request comes, it shall validate it, and reply to the hyperMAS accordingly. We will check this for valid and invalid requests.
3. After receiving a valid help request, the HIL service shall publish the help request through MQTT. For the test we check whether the MQTT publication reaches all active operators.
4. The HIL service is now ready to accept an "operator takeover" event from one operator. For the test, we send one such event and expect an accept message containing an MQTT topic ID for the following communication between the HIL service and the Operator
5. We send a second "operator takeover" event from another operator and expect a negative acknowledge from the HIL service, because the HIL job is already taken.
6. After sending a help request, the HyperMAS will poll for the state of the concerned help operation using the given aiSessionId as reference. In this context we test the following:
    a. While a HIL operation is ongoing, we send a request in every state and expect the according reply
    b. When sending a request with and invalid or non-active ID, we expect negative acknowledge without any succeeding consequences.
7. When an operator has taken over, the HIL service is expected to request the edge orchestrator to update the orchestration for the concerned HIL application, asking for a QoS-enabled communication relation between the concerned robot and operator. The Edge orchestrator is expected to acknowledge this request after interacting with the TSN controller and the communication resource manager. A proper acknowledge is sufficient for the test of the HIL service.
8. After receiving the acknowledgement from the edge orchestrator, the HIL service pushes the start help event to MQTT, which MQTT broker will forward to all operators. The selected operator knows the given aiSessionId and starts operation. The HIL service is not involved in this operation, thus it is not relevant for this test. From the HIL service point of view it is sufficient to check, whether the MQTT publication was sent.
9. After the HIL operation is done, the operator uses HIL service's REST interface to inform it about the operation is done. HIL service accepts this request. We also test, that any issue solved request with wrong or unknown ID is answered with negative request without further implications.
10. After receiving a valid issue solved request, the HIL service is expected
    a. to again update the orchestration at edge orchestrator to free reserved communication resources. The latter is using the same interface as when reserving the resources.
    b. Request the current tool center point (TCP), i.e., the actual grab spot, from robot controller. When HIL service correctly uses the robot controller's interface, we expect TCP coordinates in the reply.
11. Finally, having received TCP coordinates from robot controller, the HIL service will forward these coordinates together with the aiSessionId of the current HIL operation to the HyperMAS. When sending valid coordinates and a valid ID, we expect positive acknowledge from the HyperMAS and the operation is completed.

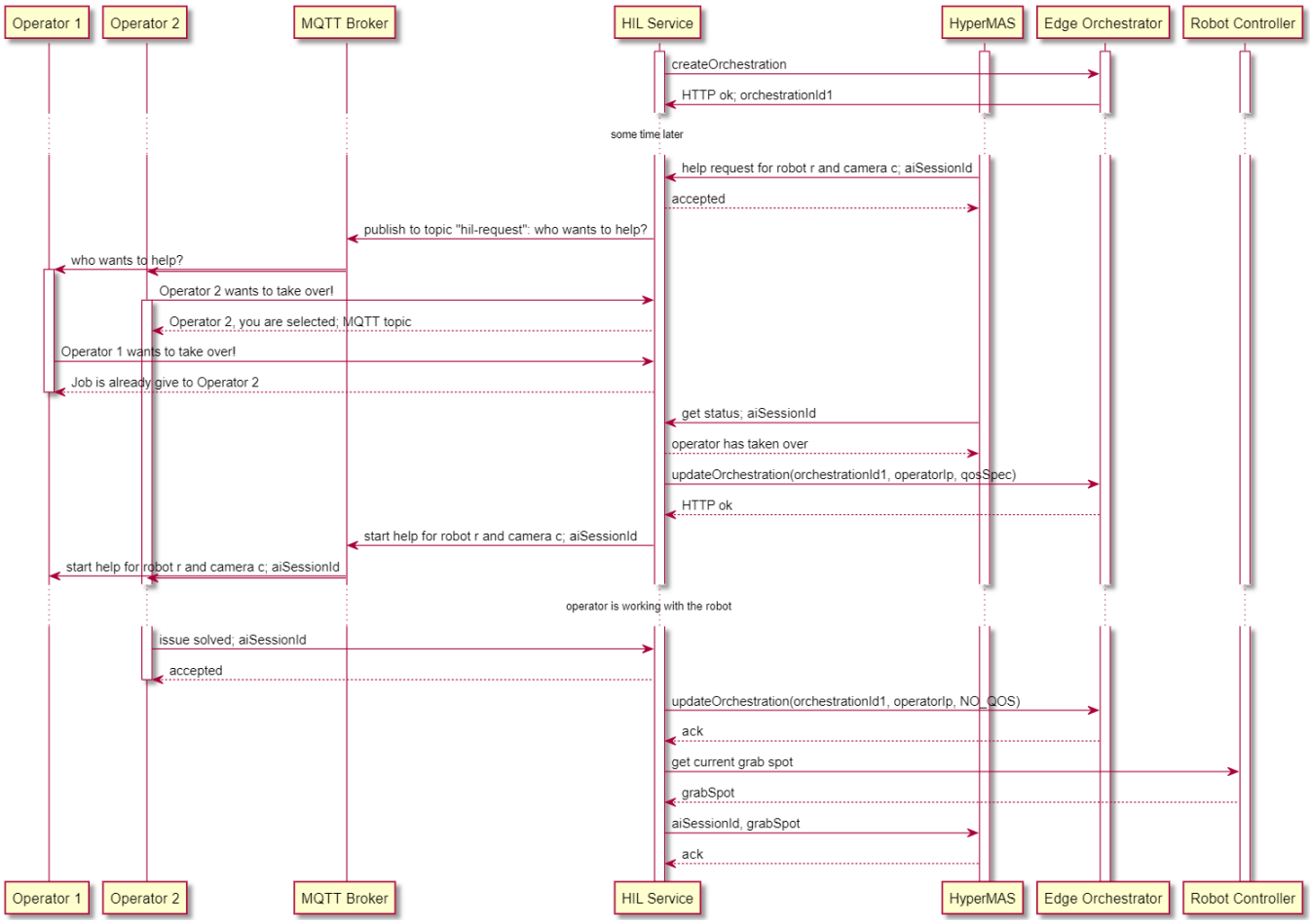


Figure 54: HIL test sequence

5.3.3 INTEGRATION AND TEST OF HIL APPLICATION

The HIL Application's role for the Manufacturing use case is similar to UC1 as the Operators role is to help the AI when it encounters an obstacle.

The Operators will run an application on the HoloLens 2 that subscribes to the MQTT broker and waits for the HIL Service to publish the help request from the AI. Once the request is received, a pop-up will appear in the application to inform the Operators that a problem occurred. The Operators will confirm the task back and the first to be chosen will go on to connect to the HIL Application. The MQTT broker covers the integration of the events between the Operators and the HIL Service.

Similar to UC1, the integration between the HIL Application and the Operators is handled by ISAR.

The HIL Application will simulate the robot movement in the AR space. This is possible by creating a TCP endpoint between the HIL Application and the ROS workspace for the UR5 robot. Please be aware that this integration is still a work in progress.

The HIL Application and the Raspberry Pi installed on the robot arm will connected through a WebRTC interface (Integration in progress).

There will be 2 focus points for testing. The first one is the functional data channels and the second one is interaction with UI elements.

The tests must verify the latency of the video feed, as well as the robot controls. Thresholds have not been defined yet, as more initial testing is needed. Furthermore, intractability, FPS count and Stylus behavior must also be tested. The final test plan will be outlined in the next iteration of this document.

5.3.4 INTEGRATION AND TEST OF ROBOT CONTROLLER AND AI

Before integrating the AI component on the robot, the AI will be trained offline in the lab using a set of labeled training data. The training data is composed of still images, extracted from a video stream from a camera mounted above the work area, that are labeled by a human operator with two different labels (usable area and grabbing position of the workpiece). Offline training is used to train the model until it reaches a certain level of accuracy. The pretrained AI component will then be integrated into the manufacturing plant.

Workpieces are placed on the storage table and on various machines for testing purposes. When the robot needs to select and pick a workpiece during the process, the AI component first captures the current state from the video stream. It then identifies the workpieces and calculates their usable area (the center, radius, and area of the largest circle that can be placed on the workpiece) before reporting to the HyperMAS. The AI component will receive the ID of the workpiece that needs to be transported to a desired destination based on the requirements. When the AI component receives this request, it computes the grabbing point: the (x,y) coordinates and the angle of the gripper. In addition to the grabbing point, the AI component will generate a level of confidence in its inferred decision. If the level of confidence exceeds a predefined threshold, the decision is communicated to the robot controller. Otherwise, the AI escalates to the human operator in accordance with the Human-in-the-Loop scenario discussed in deliverable D2.1 under UC1.

5.3.5 INTEGRATION AND TEST OF TSN CONTROLLER

The TSN controller receives so-called Communication Service Requests via its northbound interface. This HTTP-based interface is described in https://gitlab.eurecom.fr/intelliot-project/networking/tsn-controller and might be used by any application requiring a communication service with guaranteed QoS properties. In IntellIoT, this interface is used only by the edge orchestrator and the security assurance platform. The latter will not request communication services but point at dedicated malicious nodes to be excluded from communication with the rest of the network.

The SW-component TSN controller was intensively tested to configure the underlying TSN infrastructure according to the requested communication service and it's QoS properties in a Siemens-proprietary test bed, refer to deliverable D5.1 for details. As a precondition for this, it's ability to correctly detect the active network topology has been tested beforehand.

To integrate the TSN controller into the IntellIoT manufacturing use case environment, we need a TSN topology with at least one supported switch and two end stations. With this, the following test steps are performed:

- Network controller will gather topology information.
- Edge orchestrator will read this topology information.
- For test purposes, the correctness of the topology received in the edge orchestrator will be checked manually.
- Edge orchestrator will then be triggered to request a communication service from the TSN controller. We will both put feasible and unfeasible requests in order to check if we receive the correct reply.
- For feasible requests, we then need to test if the communication service was actually embedded correctly. Therefore, a Siemens proprietary test tool will be used to measure throughput, delay and jitter of the requested communication services.
- When available, we might additionally run the according application to proof its proper function.

The described test procedure will be repeated, when new devices, particularly switches, are added to the active topology, or when new releases of network controller or edge orchestrator are to be published.

5.3.6 INTEGRATION AND TEST OF COMMUNICATION RESOURCE MANAGER

The communication resource manager receives requests from the Edge Orchestrator via its northbound interface. This HTTP-based interface is described in: https://gitlab.eurecom.fr/intelliot-project/edge/architecture/edge-

architecture-uc3/-/tree/master/commRM. As shown in Figure XX, the performance parameters that can be requested when reserving communication resources are the maximum latency, the maximum jitter and the minimum throughput.

```
    ReservationDescription:
      type: object
      required:
        - clientAddress
        - addressType
      properties:
        clientAddress:
          type: string
        addressType:
          $ref: '#/components/schemas/AddressType'
        maxLatencyMillis:
          type: number
          format: double
        maxJitterMillis:
          type: number
          format: double
        minThroughputMbps:
          type: number
          format: double
```

Figure 55: Communication Resource Manager northbound interface

The southbound interface with OAI is defined in: https://mosaic5g.io/apidocs/flexran/.

Until the first demonstration of the project, the integration plan consists of the following steps:

1. Definition of the northbound API and container. This step has been completed and it is available in the gitlab of the project.

2. Executability test in the Edge Orchestrator. This step is under progress at the time of submitting this deliverable and involves executing the edge app "comm. resource manager" in the edge orchestrator.

3. Functional test with the Edge Orchestrator and an emulated 5G infrastructure (no southbound integration). The basic functionality will be tested, namely starting a new slice, adding UEs to an existing slice, deleting a slice.

4. Functional test with the Edge Orchestrator and OAI 5G infrastructure (southbound integration). After testing the southbound interface with the OAI, the same tests as in the emulated environment will be performed. This functional test is similar to the one described in Section 6.3.7.

5.3.7 INTEGRATION AND TEST OF DLT

The system integration of DLT includes multiple components as below:

- **DLT Manager**: This component includes all modules to build various features of DLT technologies such as consensus, smart contract, data authorization, identity management, and peer-to-peer (P2P) communication. These components must ensure that every change to the ledger is reflected in all copies in seconds or minutes and provide mechanisms for the secure storage of the data generated by IoT devices and parameter configurations. There are numerous DLTs with different characteristics that may be beneficial for different target applications. The DLT nodes can be located everywhere and connected with base stations via the Internet. The data structure of UR5 can be described as below:

publish-data-to-dlt.py 5.96 KB Edit Web IDE Replace Delete

```python
#!/usr/bin/env python

import argparse
import logging
import sys
sys.path.append('/home/user/intelliot/distributed-ledger-technology/DLT-for-Manufacturing/shared-manu/ur5-robot/')
import rtde.rtde as rtde
import rtde.rtde_config as rtde_config
import rtde.csv_writer as csv_writer
import rtde.csv_binary_writer as csv_binary_writer

import json
import time
import random
import json
import web3
from web3 import Web3
from web3.gas_strategies.time_based import (fast_gas_price_strategy,
                                            medium_gas_price_strategy,
                                            slow_gas_price_strategy,
                                            glacial_gas_price_strategy)


#parameters
parser = argparse.ArgumentParser()
parser.add_argument('--host', default='10.0.2.4', help='name of host to connect to (robot controller)')
parser.add_argument('--port', type=int, default=30004, help='port number (30004)')
parser.add_argument('--samples', type=int, default=0, help='number of samples to record')
parser.add_argument('--frequency', type=int, default=125, help='the sampling frequency in Herz')
parser.add_argument('--config', default='record_configuration.xml', help='data configuration file to use (record_configuration.xml)')
parser.add_argument('--output', default='robot_data.csv', help='data output file to write to (robot_data.csv)')
parser.add_argument("--verbose", help="increase output verbosity", action="store_true")
parser.add_argument("--buffered", help="Use buffered receive which doesn't skip data", action="store_true")
parser.add_argument("--binary", help="save the data in binary format", action="store_true")
args = parser.parse_args()

logging.basicConfig(level=logging.INFO)

```

Figure 56: Data structure of UR5 to record to distributed ledger

- **Physical Machines**: This component consists of physical robots, machines, and IoT sensor devices which collect the data and publish to the distributed ledger for accounting or analysing purposes. In our scenario, we use the UR5 robot arm for proof of concept. For example, the configuration of record data of UR5 robot can be shown as below:

record_configuration.xml 4.45 KB

```xml
<?xml version="1.0"?>
<rtde_config>
        <recipe key="out">
                <field name="timestamp" type="DOUBLE"/>
                <field name="target_q" type="VECTOR6D"/>
                <field name="target_qd" type="VECTOR6D"/>
                <field name="target_qdd" type="VECTOR6D"/>
                <field name="target_current" type="VECTOR6D"/>
                <field name="target_moment" type="VECTOR6D"/>
                <field name="actual_q" type="VECTOR6D"/>
                <field name="actual_qd" type="VECTOR6D"/>
                <field name="actual_current" type="VECTOR6D"/>
                <field name="joint_control_output" type="VECTOR6D"/>
                <field name="actual_TCP_pose" type="VECTOR6D"/>
                <field name="actual_TCP_speed" type="VECTOR6D"/>
                <field name="actual_TCP_force" type="VECTOR6D"/>
                <field name="target_TCP_pose" type="VECTOR6D"/>
                <field name="target_TCP_speed" type="VECTOR6D"/>
                <field name="actual_digital_input_bits" type="UINT64"/>
                <field name="joint_temperatures" type="VECTOR6D"/>
                <field name="actual_execution_time" type="DOUBLE"/>
                <field name="robot_mode" type="INT32"/>
                <field name="joint_mode" type="VECTOR6INT32"/>
                <field name="safety_mode" type="INT32"/>
                <field name="actual_tool_accelerometer" type="VECTOR3D"/>
                <field name="speed_scaling" type="DOUBLE"/>
                <field name="target_speed_fraction" type="DOUBLE"/>
                <field name="actual_momentum" type="DOUBLE"/>
                <field name="actual_main_voltage" type="DOUBLE"/>
                <field name="actual_robot_voltage" type="DOUBLE"/>
                <field name="actual_robot_current" type="DOUBLE"/>
                <field name="actual_joint_voltage" type="VECTOR6D"/>
                <field name="actual_digital_output_bits" type="UINT64"/>
                <field name="runtime_state" type="UINT32"/>
<!--
```

Figure 57: Configuration of UR5 robot to communicate with DLT.

- **Plant Edge Devices**: Even though DLT-based solutions offer significant countermeasures to secure data from tampering and support the distributed nature of the IoT, the massive amount of generated data from sensors and the high energy consumption required to verify transactions make these procedures unsuitable to execute directly on resource-limited IoT devices. Instead, edge servers with high computation resources can be used to handle real-time applications and to further increase the degree of privacy (e.g., through cloud computing). The edge network is a potential entity to cooperate with the DLT network in computationally heavy tasks and return the estimation results (e.g., from solving proof-of-work (PoW) puzzles, hashing or algorithm encryption) to the DLT network for verification.
- **External Services**: The devices of the manufacturing environment are typically resource-constrained with limited storage space and low computation capacity. Hence, external infrastructure which operates on the edge may be incorporated to provide external services, such as storage and computing. For example, the Interplanetary File System (IPFS) is a distributed file storage system that can store data generated from IoT networks and return a hash to the ledger based on the content of the data. Since the ledger cannot handle and store the massive amount of manufacturing data collected by the sensors, machines, and robots, the service provided by the IPFS is a vital component.

The testbed setup for the DLT-based manufacturing system is shown below. The UR5 robot communicates with the DLT clients via UR-RTDE library written in Python.

The activities of UR5 robot are recorded timely from robot controller via UR-RTDE library, then the robot controller forwards the data to the DLT clients named Geth installed locally in a remote server which have an inter-connection with the Robot Controller. The DLT clients transmit the activities of the robot to a distributed ledger via a Smart Contract, which defines the selected data needed to be stored in the ledger. In the scope of this use case, we recorded the information about timestamp, movement of robot to the distributed ledger for analyzing and accounting purpose.

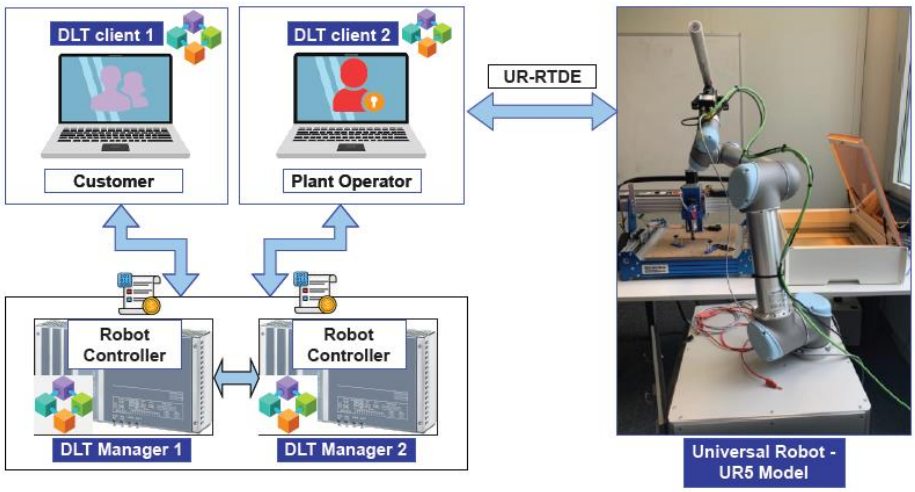
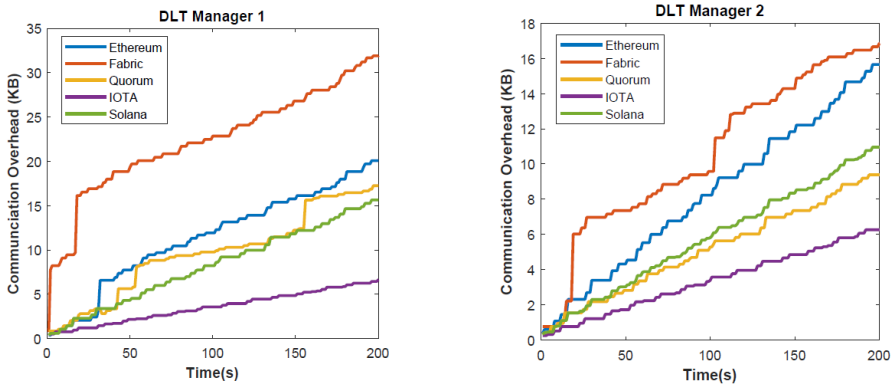


Figure 58: DLT-based manufacturing testbed

The performance of the system can be demonstrated as below:



(a) Communication overhead of *DLT-manager-1*. **(b)** Communication overhead of *DLT-manager-2*.

Figure 59: Communication overhead of the integration system.

5.3.8 INTEGRATION AND TEST OF SAP

In the manufacturing use case testbed, the generic process of deployment and setup / configuration of SAP is followed, as detailed in subsection 6.5 above.

In addition to testing the standalone features of SAP, setup and testing of the integration between SAP and the rest of the trust components will also take place, per the process documented in deliverable D5.1, subsection 2.4.

During Cycle 2, it is expected that this process will also include the deployment of the appropriate event captors at the robot controller, to detect abnormal movements, potentially indicating malicious activity (e.g., robot compromise), the deployment of incident response playbooks to enable a human-in-the-loop mitigation strategy, as well as tighter integration with the AAA & MTD components, to allow for the execution of this strategy.

6 CONCLUSION AND FUTURE WORK

This deliverable has described planned and taken steps to deploy, test and demonstrate hard- and predominantly software components described in work package 3 and 4 deliverables. A main focus was on the integration of the components, which have already been pre-tested in component tests at the responsible partners. With these achievements we set the cornerstone for the demonstrations for cycle one review.

The cycle 2 version of this document will close the gaps between the current state and the scenarios and scenes described in D2.1. As mentioned, improvements will be made to nearly all components, and individual use cases will bring more components in deployment, in order to fully cover all scenarios in the final review.