

Project Deliverable 5.1

Blueprint architecture and integration plan

Worker-Centric Workplaces in Smart Factories

www.facts4workers.eu



Series: Heading

Published by: FACTS4WORKERS: Worker-Centric Workplaces in Smart Factories.

FoF 2014/636778

Volume 1: Blueprint architecture and integration report – Deliverable 5.1 – Project Report FACTS4WORKERS: Worker-Centric Workplaces in Smart Factories

Reference / Citation



Gerhard, Detlef; Dumss, Stefan; Rosenberger, Patrick (2016): Deliverable 5.1: Blueprint Architecture and Integration Plan. Project FACTS4WORKERS: Work-Centric Workplaces in Smart Factories.

www.facts4workers.eu

1. Printing, July 2016

Layout and Setting: Florian Ott, Cooperation Systems Center Munich

Worker-Centric Workplaces in Smart Factories

E-Mail: facts4workers@v2c2.at

Internet: www.facts4workers.eu



This document is published under a Creative Commons Attribution Non Commercial No Derives licence. You are free to copy and redistribute the material in any medium or format. You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. You may not use the material for commercial purposes. If you remix, transform, or build upon the material, you may not distribute the modified material.

<http://creativecommons.org/licenses/by-nc-nd/4.0/>

About this document



Executive Summary

This deliverable 5.1 “Blueprint architecture and integration plan” is a result of the project “FACTS4WORKERS – Worker-Centric Workplaces in Smart Factories” of the European Union’s Horizon 2020 research and innovation programme under the grant agreement No. 636778.

The deliverable reflects on the architectural cornerstones of the FACTS4WORKERS solution in terms of main hardware and software technologies. Furthermore, the deliverable shows the approach the project FACTS4WORKERS has taken to develop, deploy, integrate, and test the worker centric software solution. It starts with the description of the integration plan on a generic level and points out the necessary steps to transfer the individually developed and tested services to an integrated functioning system. Following the agile software engineering approach of the project, integration steps will be performed iteratively improving the solution step by step and increase its maturity continuously. This concept also allows to rapidly integrate feedback from users enabling the worker centric approach. Chapter 3 of the deliverable provides an overview over the deployment and testing phase. Here, the different maturity levels prior to release are summarized. This concept allows to introduce the FACTS4WORKERS solution in several steps and constantly gather feedback from the workers. The final chapter provides an overview over the hardware and software architecture of the FACTS4WORKERS solution. Required hardware technology for deploying the solution on the shop floor as well as used software technologies, frameworks and programming languages are outlined. The deliverable finishes with the description of the tool Docker, which serves as enabler for the concept of building blocks.

Document authors and reviewers

The following persons have made a direct contribution to the document. Please note that many others have also supported our work and we thank them all sincerely.

Lead Authors

Name	Organisation	Role
Detlef Gerhard	VUT / TU Wien	WP5 – Leader
Stefan Dumss	VUT / TU Wien	WP5
Patrick Rosenberger	VUT / TU Wien	WP5

Featuring Authors

Name	Organisation	Role
Alexander Porsch	VUT / TU Wien	WP5
Lucas Bezzi	VUT / TU Wien	WP5
Alessio Caiazza	UFI	WP3

Reviewers

Name	Organisation	Role
Martin Wifling	VIF	WP8 – Lead, Project Coordinator
Gianni Campatelli	UFI	WP3 – Leader
Joachim Van Herwegen	IMI	WP4
Francisco J. Lacueva	ITA	WP6

Table of Contents

Executive Summary	I
Document authors and reviewers.....	III
Table of Contents.....	IV
Table of Figures.....	VI
Index of Abbreviations	VII
1 INTRODUCTION	9
2 INTEGRATION PLAN	11
2.1 Integration process	11
2.2 Roles and Responsibilities.....	13
2.3 General guidelines for implementation and integration.....	15
3 DEPLOYMENT AND TESTING	17
3.1 Deployment.....	17
3.2 Testing.....	20
3.2.1 Code testing.....	20
3.2.2 Usability and requirements fulfillment testing.....	21
4 GENERIC HARDWARE AND SOFTWARE ARCHITECTURE	23
4.1 Generic hardware technology overview.....	23
4.1.1 Devices.....	24
4.1.2 Servers.....	25
4.2 Used software technology overview	26
4.2.1 Communication and routing.....	27
4.2.2 Front-end.....	30
4.2.3 Back-end	31
4.3 Semantic workflow engine	35
4.3.1 Introduction	35
4.3.2 Strength of the Semantic Workflow Engine within the Project.....	35
4.3.3 FACTS4WORKERS example for the semantic workflow engine.....	35

4.4 Docker.....37

4.4.1 Introduction.....37

4.4.2 Containers versus VMs38

4.4.3 Strength of Docker within the Project.....40

REFERENCES.....44

Table of Figures

Figure 1: Schematic view on microservices.....	10
Figure 2: FACTS4WORKERS integration process.....	12
Figure 3: General guidelines.....	16
Figure 4: Deployment steps and TRL	19
Figure 5: Example of possible connections.....	23
Figure 6: Architectural overview	26
Figure 7: HTTP status codes.....	28
Figure 8: JSON objects [<i>JSON, 2016</i>]	29
Figure 9: JSON arrays [<i>JSON, 2016</i>].....	29
Figure 10: SWE schema.....	36
Figure 11: Structure virtual machines [<i>Docker, 2016</i>]	39
Figure 12: Structure of Docker containers [<i>Docker, 2016</i>]	39
Figure 13: VMs versus Docker Containers	40

Index of Abbreviations

API	Application Programming Interface	JSON.....	JavaScript Object Notation
App	Application	Lib	Library
AWS.....	Amazon Web Services	LTE.....	Long Term Evolution
BB.....	Building Block	OS	Operating System
CSS	Cascading Style Sheets	REST	Representational State Transfer
Dx.x.....	Deliverable x.x (x ... placeholder)	SME.....	Small and medium- sized Enterprises
GUI	Graphical User Interface	SQL.....	Structured Query Language
HMI.....	Human Machine Interface	TRL.....	Technology Readiness Level
HSDPA.....	High Speed Downlink Packet Access	UMTS.....	Universal Mobile Telecommunications System
HTML.....	Hypertext Markup Language	URI	Uniform Resource Identifier
HTTP.....	Hypertext Transfer Protocol	VM	Virtual Machine
IP	Industrial Partner	WiFi	Synonym for Wireless Local Area Network
IT	Information Technology	WP	Work Package
JDBC	Java Database Connectivity		

1 Introduction

With respect to development, deployment, integration, and test, the objective was to take the following considerations for the worker centric software solution into account:

- 👤 The software should be developed using a modular concept to enable reuse of building blocks in different use-cases and to ensure expandability.
- 👤 The front-end should be developed as web application, enabling access by different devices (e.g. desktop computers, tablets, smartphones, etc.) according to the situational needs.
- 👤 Building blocks should be developed with the programming language and framework best fitting to the requested demands. This leads to the usage of different programming languages within the software, a circumstance that has to be dealt with while designing the blueprint architecture.
- 👤 It should be easy for developers, who are not participating in this project, to improve and further develop the solution, after the project is finished.
- 👤 Parts of the solution developed within the project should be published under an open source license. If required, other license models are permitted, but should be reduced to a minimum.

To achieve the stated objectives, the project follows an agile software development approach based on the concept of microservices. *“Microservices are small, autonomous services that work together”* [Newman, 2015] to achieve the requested functionalities. The FACTS4WORKERS solution consists of several microservices called “building blocks” (BB) within the project. Each of the BB covers a specific task. For detailed information about BB please refer to D3.1. As BB are developed independently and get composed to a functioning system during the deployment, they can be extended or replaced with less effort compared to monolithic systems. Particularly, each BB can be deployed as an isolated service. This concept allows the usage of different programming languages for each BB and prevents that others get affected if one BB fails.

This document provides a generic view on the integration and deployment process of the FACTS4WORKERS solution. The described processes and operations state a general model that can be applied on the use-case specific implementation plans. These use-case specific considerations will be part of the deliverable D5.2.

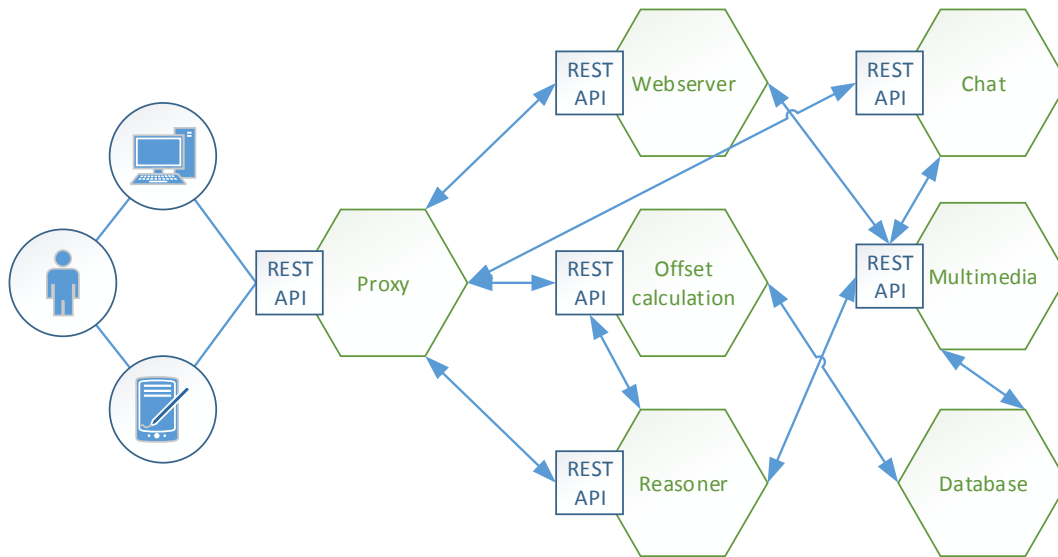


Figure 1: Schematic view on microservices

Structure of the document

Chapter 2 starts with the integration plan. First, the process steps needed for the integration are explained. Then, responsible persons and their tasks are identified. Finally, general guidelines for implementation and integration are introduced to ensure the same direction of development.

Chapter 3 gives an overview over the deployment and testing phase, pointing out the different deployment steps and the maturity levels the FACTS4WORKERS solution has to reach prior to release. Furthermore, activities for testing functionalities and user experience are described providing a detailed insight into the integration process of the project.

Chapter 4 explains the hardware and software architecture, starting with a generic overview over the hardware requirements for deploying the FACTS4WORKER solution on the shop floor level. Afterwards, the used software technologies, programming languages and frameworks are summarized. Furthermore, the Semantic Workflow Engine gets described, showing its functionalities and routing procedures used for the FACTS4WORKERS solution, that allows the capsulation of the software and the operating system deployment. The chapter ends with a description of the deployment tool Docker that enables the usage of different programming languages without having major dependency issues.

2 Integration Plan

This chapter aims to provide a generic overview on the necessary steps in order to transfer the individually developed and tested services to an integrated functioning system. Following the agile approach of the project, integration steps will be performed iteratively, improving the solution step by step and increasing its maturity. This concept allows to rapidly introduce feedback from users enabling the worker centric approach.

2.1 Integration process

The following figure shows the integration process on a generic level. The integration process is based on the requirements gathered directly from the workers within the different use cases. For further information about collected requirements from a worker-centric perspective, please refer to D1.1, D1.2 and D1.3. In addition, company specific requirements on organizational, IT, and management level also have to be taken in account. These include among others in particular safety and security standards as well as software and hardware restrictions.

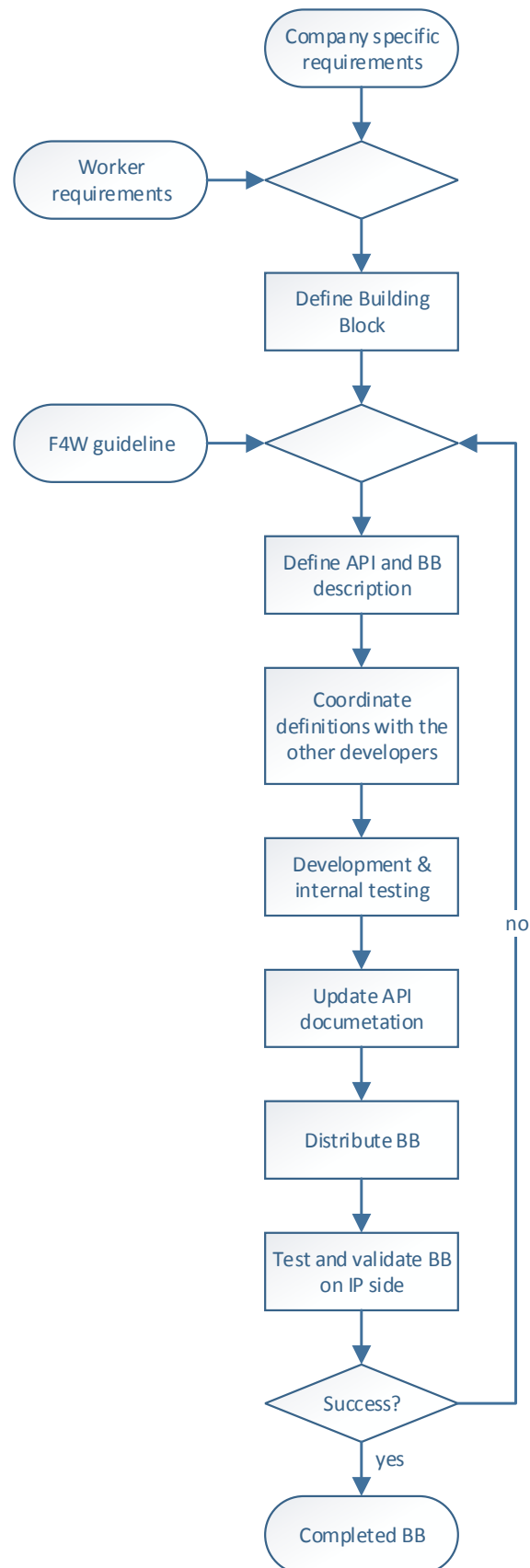


Figure 2: FACTS4WORKERS integration process

In a first step, services are defined upon collected user requirements and in alignment with company specific requirements representing the boundaries stated by management and IT departments. For the definition of services, use cases are split up into building blocks following the microservices concept with a clear input and output. This allows to divide the software development activities between the different development partners.

Under consideration of the integration guideline of chapter 2.3, building blocks and APIs are described in detail. This helps to determine BBs that can be implemented in different use cases. Then, based on the specification, development and testing activities are split up between the developing partners.

Subsequently, the different building blocks are developed and tested in compliance with the procedures stated in chapter 3.2. If a BB needs to be changed or more improved, this is done in coordination with the other developers. Afterwards, the BB and API documentation gets updated regarding possible changes to ensure that all partners have at all times a clear overview over the development activities. After development, BB are distributed to the deploying industrial partners (IP) in order to be integrated into the already existing solution portfolio.

BB are developed and integrated continuously, enriching the software functionalities incrementally. For aligning their functionalities, an extensive documentation is shared among the developers and the use case leaders. The API documentation is a key part of the documentation. Collecting feedback from the user after each rollout allows the continuous comparison between the delivered software and the worker needs. Doing so, deviations can instantly be identified reducing the risks of wasted resources and time.

Regular developer meetings on a two weeks' basis allow the exchange of information and the establishment of a common ground for the further developments. Additionally, mock-ups and demonstrators are reviewed and discussed in order to provide feedback among the developing partners. This procedure enables the creation of BBs that satisfy the needs of all use cases and ensures reusability of building blocks. Regularly meetings for discussing issues regarding the use cases on a two weeks' basis allow the exchange of company specific information and the alignment of the development activities with the worker needs.

2.2 Roles and Responsibilities

As setting up clear responsibilities is a key factor for the successful integration of an individually developed and tested system, several roles are introduced within the project:

Use Case Leader

Use case leaders have the overall responsibility for management and coordination of the whole use case. This includes in particular the interaction between software developers, deployment responsible persons, and industry partners.

- 🔧 Define the needed building blocks and their priority.
- 🔧 Coordinate activities for the use case and manage communication.
- 🔧 Report progress to the other project partners.

Industry partner

Industry partners (IP) are responsible for providing the necessary link to hard- and software infrastructure on company side. Particularly, IT departments of IP have to be involved in order to define interfaces and responsibilities for the deployment of developed FACTS4WORKERS solutions. The following tasks are within the responsibility of the IPs:

- 🔧 Provide an API to existing systems or an interface, which have to be integrated the FACTS4WORKERS solution.
- 🔧 Provide the hardware components required to run server, client and network communication.
- 🔧 Deploy and test the software and provide developers with feedback.
- 🔧 Provide necessary content and cannot be accesses over an existing IT-system

Implementation and deployment work is separated into three different roles, which work under main responsibility of use case leaders:

Back-end developer

Back-end developers are responsible for implementing the required core functionalities of the FACTS4WORKERS solution. Based on requirements identified within WP 1, corresponding building blocks are developed according to the BB concept. As the software consists of several BB that communicate over APIs, they are developed individually and get integrated after their release. Releases do not have to be final but a stable version, which can be tested in the company environment for gathering feedback and enable further improvement.

All BB are built as so-called Docker containers (see chapter 4.3) capturing a particular service. The following tasks represent the main responsibilities of the backend developers:

- 🔧 Define the API based on the requirements and implement the service.

- 👤 Include already existing modules or commercial solutions if available.
- 👤 Coordinate with other developers regarding the reusability of the building blocks.
- 👤 Build Docker containers that are ready for testing and deployment.

Front-end developer

Front-end developers are responsible for implementing the required user interfaces of the FACTS4WORKERS solution. Based on requirements identified within WP 1 and generated mock-ups or demonstrators, corresponding GUI are implemented and linked to the respective building blocks. The following tasks represent the main responsibilities of the frontend developers:

- 👤 Create GUI layout for different devices based on the requirements.
- 👤 Include already existing modules or commercial solutions if available.
- 👤 Build Docker container that are ready for testing and deployment.

System integrator

System integrators are in charge of deployment and integration of the developed solutions at the partner sites in cooperation with nominated persons of the industry partners. Integration also includes the identification of missing building blocks if they are identified during the integration process. The following tasks are within the responsibility of the system integrators.

- 👤 Develop connectors to the existing IT systems of the IPs if required.
- 👤 Install the connectors linking to the FACTS4WORKERS solution to existing IT infrastructure.
- 👤 Deploy the Docker containers and set up a reverse proxy at the industry partners.

2.3 General guidelines for implementation and integration

Since the microservices are designed as BB, which interact with each other, standards are needed in order to ensure full functionality:

- 👤 Each back-end BB is developed independent from the others and capsulated in a Docker container.
- 👤 The front-end BB is hosted on a web server Docker container.
- 👤 Front-end and the back-end services primarily communicate over REST APIs (RESTful HTTP requests using JSON).

- For smart devices such as glasses, a WebSocket is used since this provides advantages compared to HTTP real-time applications. The corresponding web server has to have an API for other container.
- The API for each BB is documented with his exposed methods (GET, POST, PUT, etc.).
- For authentication with the server, each BB uses OAuth2.0.
- In case of static workflows, the communication is handled by a reverse proxy. In case of dynamic or complex workflows the reverse proxy is supported by the semantic workflow engine.
- For the communication with the existing IT infrastructure (e.g. ERP connection) of the IPs, connectors are used.
- Database connectors to the databases are realized using programming language specific connectors (e.g. JDBC). In contrast to the approach to use a database for every microservice, only one will be implemented for each IP. If needed additional ones can be included.

The following figure depicts the architecture with the applied guidelines. Front-end and the back-end are separated by a reverse proxy, this is for simplification and security reasons.

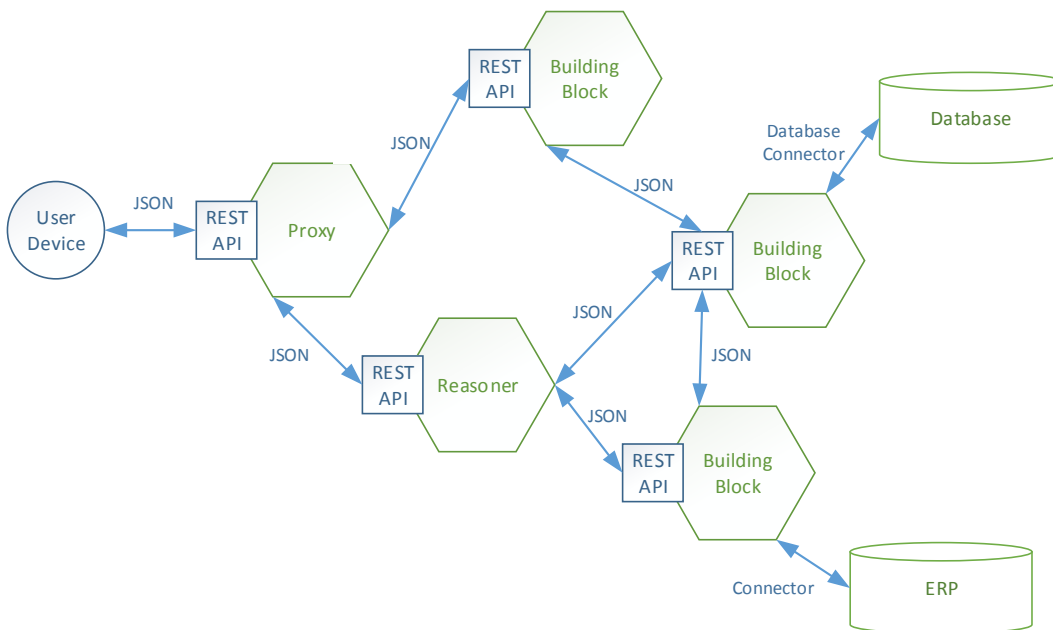


Figure 3: General guidelines

3 Deployment and Testing

3.1 Deployment

As already stated Docker is used as deployment tool, since it simplifies the deployment efforts by minimizing the dependencies between the building blocks. Though Docker reduces the dependencies, Docker is not a classic deployment tool, demanding that the project follows a general approach for the deployment of the developed and tested Docker containers. In line with the principles of agile software development, containers are deployed continuously as they are released in order to get the immediate feedback from the workers.

Phase I – development phase – constitutes the traditional software development circle, which consists of requirements definition, technology development and technology deployment. The project aims to deploy a first set of building blocks and components to be tested at the industry partner's sites together with workers in the following phases. To build up a first user-base of workers, these BBs will remain open for tests and refinements of requirements until the end of the project. The goal is to quickly demonstrate benefits to the involved workers on the shop floor in order to ensure a high technology acceptance and to receive feedback to be integrated in the development on a very early stage.

Phase II – refinement phase – can be considered as the first prototype stage. A first version of building blocks is available at the industry partner sites, but they are not fully stabilized and finished according to the expectations of workers and to the organizational boundaries. This phase starts with an evaluation of the developed BBs on both, technical level as well as together with workers. Improvements will be developed iteratively and included into the refinement of BBs.

Phase III – perpetual beta – includes the continuous evaluation of the BBs, improvement as a result of the integrated evaluation feedback, as well as the integrated FACTS4WORKERS solution. We expect an increasing number of workers piloting the solution at the industry partner sites and thus a higher quantity and quality of feedback and recommendations. Qualitative user feedbacks using innovative approaches like thinking aloud or emotion tracking will provide more detailed insights and generated benefits. For further information, please refer to deliverable D6.1. The last step is to transfer BB into a more generic FACTS4WORKERS solution with respect to selection, integration and maintenance. This solution to be developed will result in a powerful platform ready for exploitation in various production scenarios and applicable to a wide range of manufacturing employees of all ages carrying out manufacturing tasks at small and medium-sized enterprises (SME) as well as at large-scale enterprises.

Further detailing the three-phase approach, the first phase aims to create mock-ups and demonstrators, the second phase validators, and the third phase pilots for the specific use cases of the industry partners. From a deployment perspective, following steps are necessary:

Mock-ups

Mock-ups are developed to present a draft of the “look and feel” of the software to the future users (workers) in order to gain quick feedback from them. Mock-ups consist of images or simple HTML pages indicating the appearance of the Human Machine Interface (HMI). A simple navigation is possible but not required.

Demonstrator

In addition to mock-ups, demonstrators show a proof of concept and take test data into account. This enables a feedback in terms of requirements verification, feasibility of the software, and implementation concept using dummy data. Following this approach, fast iteration loops to adjust and refine the solution proposal are possible.

A demonstrator can be seen as advanced mock-up, allowing to review the workflow and the navigation. Therefore, demonstrators are realized using “dummy data” showing the intended interaction with the HMI.

First implementation (Validator)

This requires working BBs and designated hardware as well as test data for verification. Requirements have to be completely defined in advance. With the first implementation validator, Docker containers are in place for deployment and testing. As it is the first implementation, maybe not all security needs or all final requirements are already fulfilled, but it shows clearly where the direction is going. The first implementation could also be used temporarily on the production site to let the worker review the solution in running conditions.

Continuous deployment phase (Validator)

After a first implementation follows further development of the services and therefore of the Docker container. The container and services will evolve to be more generic, so they could be reused in several similar cases. Together with the implementation, the development of the connectors for software integration on the industry partner side starts.

Final implementation (Pilot)

The final step - which is the on-site pilot implementation covering full functionality – leads to the perpetual beta status with required interfaces implemented and deployed. The software is at this point already in a state where it could be used in the company environment.

The following overview extends the overview of deliverable D1.2 by linking the technology readiness levels (TRL) to the different deployment steps:

	Aiming for	Functionality	Hardware	Corresponding Dataset	TLR
Mock-up	Proof of feasibility	None - Functions are faked, e.g. Power-Point, Balsamiq-Mock-up, HTML-Click-Mock-up etc.	Standard-Hardware e.g. Standard Tablet (OS not critical) or not digital device (paper based)	none	1
Demonstrator	Proof of concept <i>(for the technology building blocks involved)</i>	Key functions as I/O functions are working (maybe in another context) but can be explained and showed	Designated hardware type, e.g. tablet or SmartGlass, but not necessarily final hardware or OS	Test data-set or dummy-data, offline/local	2-3
Validator	Proof of value	Some functionality of Use case is covered, building blocks are working	Designated hardware	At least test data-set	4-5
Pilot	Proof of use <i>(on site evaluations of project targets)</i>	Use case functionality is fully covered, measurement of productivity and attractiveness can be done	Designated (off-the-shelf) hardware, freely available to consumers, provided by IP	Actual/Live data, software interfaces (bi-directional if required)	6-7

Figure 4: Deployment steps and TRL

All steps include a feedback loop with the workers and project partners. With feedback loops, it is ensured that the development is still on the way to a worker centric solution.

3.2 Testing

Testing includes code testing on the one hand side and usability testing on the other hand side. Code testing mostly relies on the development partners, and usability testing includes all partners, in particular the shop floor workers at IP sites.

3.2.1 Code testing

As worker satisfaction relies closely on a functioning software, the functionality testing is of high importance. The following tests have to be performed for the individually developed building blocks:

- 👤 Code testing
- 👤 Integration testing of multiple BBs
- 👤 BB specific deployment testing
- 👤 Testing of the deployed software

After development, BBs are going to be tested individually using real data provided by the industry partners. Afterwards, each BB get integrated with the corresponding BBs (front-end, back-end, reverse proxy and/or semantic workflow engine) in order to form a functional service. Occurring bugs regarding the workflow between BB are identified and fixed. If this test is passed, a BB is ready to be integrated within the already deployed FACTS4WORKERS solution and the IT system of the company. This allows the workers to actively use the service and report if any further error or malfunction occurs during usage. Referring to chapter 2.2 (Roles and Responsibilities), front-end developers, back-end developers and system integrators participate in code testing.



Front-end & back-end Developer

Developers are responsible for testing the code of the BBs and for adapting them to the feedback from the worker:

- 👤 Test the developed code
- 👤 Test the workflow between BBs
- 👤 Fix bugs
- 👤 Receive feedback
- 👤 Adapt and release adapted BBs



System integrator

System integrators are in charge of deployment and integration of the implemented solutions at the partner sites in cooperation with the responsible IT:

-  Deploy the Docker containers at the industry partners
-  Configure the reverse proxy

3.2.2 Usability and requirements fulfillment testing

Due to the user-centered approach of this project, main emphasis is put on tests regarding the usability and the requirements fulfillment. As described in detail within deliverable D6.1, corresponding tests and evaluation consists of two major steps:





-  Gathering of indirect feedback
-  Interview users

Gathering of indirect feedback from users can be done by monitoring their behavior during usage. The advantage of this method lies in the possibility of revealing implicit wishes and needs, whereby their fulfillment leads to an increase in the workers' satisfaction above average. Interviews cover direct feedback of workers where they have the possibility to express their thoughts and feelings regarding the new software tool. For more information, please refer to D6.1.

The testing activities will be performed repeatedly during the deployment of a solution covering a full use case. After users have spent some time testing the new functionalities, they will be asked about feedback regarding their experience and possible disadvantages. Together with the automatically gathered feedback, the collected input will then be used to further improve BBs. Involved roles in the usability testing process are evaluators and IPs with users on different levels from shop floor to management




Evaluator

Evaluators are in charge of collecting feedback from users and providing it to the developers.

-  Collocate testing parameters
-  Author testing procedure and framework
-  Gather feedback from workers
-  Provide developers with the feedback

Industrial Partner

The industrial partners are responsible for testing the provided services:

-  Deploy a testing team
-  Test the software in context of the respective processes
-  Provide feedback regarding usability and errors/malfunctions

4 Generic Hardware and Software Architecture

This chapter aims to provide an overview over the software and hardware components of the FACTS4WORKERS solution. **Subchapter 4.1** gives a generic overview over the devices on the client and on the server side. **Subchapter 4.2** then states the software technology used in the project. Since the general approach of providing small interconnected services is the core competence of Docker, **subchapter 4.3** describes the technology.

4.1 Generic hardware technology overview

As stated in Chapter 2, the FACTS4WORKERS solution follows a classic server-client model. The following figure shows an example of the connection between the hardware components.

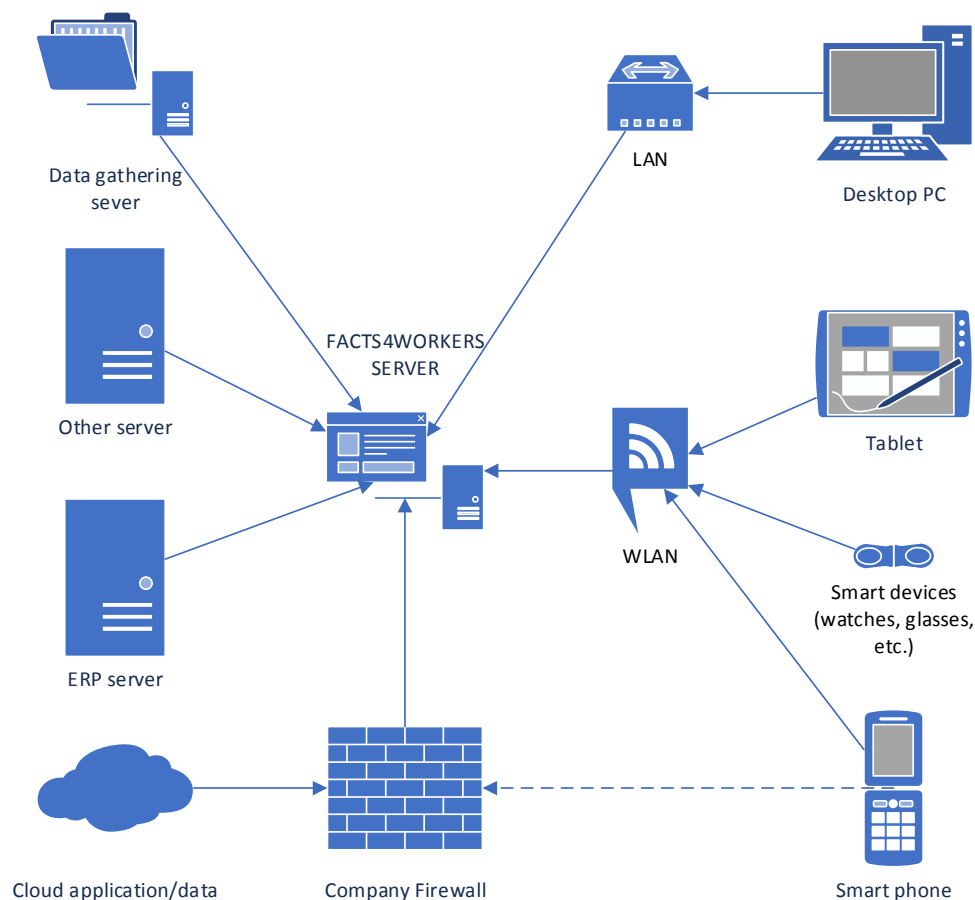


Figure 5: Example of possible connections

Three major hardware components are forming the FACTS4WORKERS solution:

- Devices
- Servers
- Network components

4.1.1 Devices

Within industrial company environments, common user groups using similar devices can be identified. The following description lists considerations that have to be taken in account when providing workers with a smart device. For a detailed description of smart devices, please refer to D2.1.

Office worker

Office workers are employees already working on a desktop PC. It is not foreseen to introduce special devices. Instead, the FACTS4WORKERS solution can be accessed using existing infrastructure components. Further considerations regarding the environmental conditions do not have to be made. In addition, the devices are most likely already connected to the local network, which allows to access the FACTS4WORKERS server. More emphasis is put on the software side, respectively the graphical user interface (GUI). The resulting requirement is, that the used front-end technology works on desktop PC in the same way it does on mobile smart devices.

Shift leader

Like office workers, shift leaders are mostly using desktop PC to perform their tasks. If they are not strictly bound to their PC or if they have to change their location, it might be necessary to additionally equip them with mobile smart devices, e.g. tablets.

Workers on shop floor level

Workers on the shop floor commonly have to access the FACTS4WORKERS solution using a smart device. The following considerations have to be taken in account:

- A fixed working environment allows the adaption of the device to the specific conditions.
- Depending on the working environment the device needs to be resistant against rough conditions, e.g. shock, dirt or fluids. Particular standards apply, such as the US Military Standard 810 [*MIL-STD-810G, 2016*]. For more information, please refer to D2.1.

- 👤 The display and the GUI elements have to be large enough to be operated in rough environmental condition and to be able to gather the needed information easily.
- 👤 The display has to be bright enough even under changing conditions.

Workers in changing work areas

Workers in changing work areas also need a flexible mobile device to access the FACTS4WORKERS solution. In addition to the considerations made above, the following points have to be taken in account:

- 👤 Changing working environments complicate the adaption to the specific environment.
- 👤 Working outdoor requires specific considerations (e.g. protection against rain for outdoor maintenance, please refer to D2.1).
- 👤 The device has to be lightweight to avoid hampering of workers.
- 👤 The device needs to be able to access the servers using different communication standards since one may not be available at all times. The device should mainly access the local network using WiFi. At some places, especially large industrial plant areas, WiFi connections are not available. Therefore, mobile cellular phone networks (UMTS/HSDPA/LTE) have to be supported by the devices in some use cases.

4.1.2 Servers

The FACTS4WORKERS solution requires a dedicated but not extraordinary server hardware environment for running the different services within the Docker environment. The reason behind this is described in chapter 4.4. The following considerations have to be taken in account before setting up the servers:

FACTS4WORKERS server

- 👤 For running Docker, the FACTS4WORKERS server has to work on an LINUX based operation system. Since the server is a central element of the solution, it has to be fail-safe and secure.
- 👤 The software Docker Compose is required for an easier running of multi-container applications.
- 👤 As long as Docker and Docker Compose can be installed, no specific hardware or software is required.
- 👤 The Semantic Workflow Engine runs as a Docker container. Operation is possible on every server fulfilling the above-mentioned requirements.

- Due to the usage of Docker, it is possible to use a dedicated server or virtual server to host the Docker containers. On premise installation as well as usage of a cloud server is possible. The impact of Docker within the project is described in chapter 4.3.

4.2 Used software technology overview

The FACTS4WORKERS solution is characterized by providing a wide range of functionalities supporting worker in different processes on the shop floor. To deal with the arising complexity and to develop functionalities suiting the workers' needs, different technologies, frameworks and programming languages are used within the project.

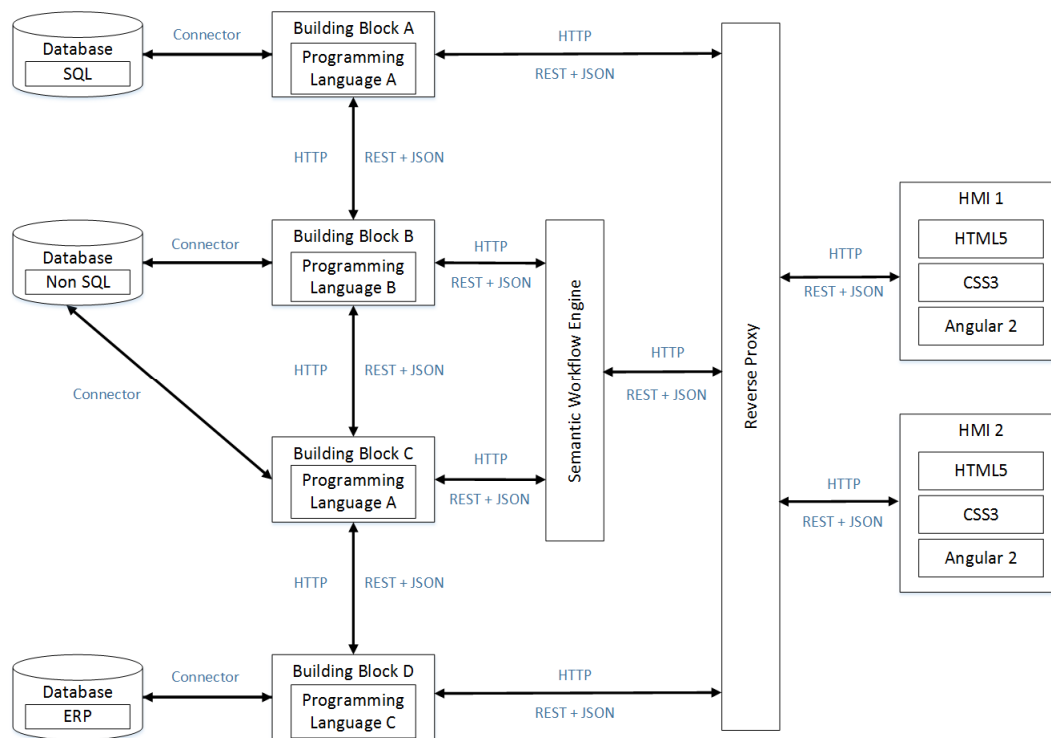







Figure 6: Architectural overview

For the front-end BBs, the markup languages HTML5 and CSS3 and the framework Angular2 (JavaScript) are used in combination. The back-end BBs are created using frameworks like Rails (Ruby), Spring (Java), Flask (Python), Phoenix (Elixir) or the programming language Go, depending on the requested requirements. For communication and exchange of data between the different BBs, either a proxy or a direct communication is implemented. In case of more sophisticated tasks, communication on back-end side is realized by a semantic workflow engine. Due to the used technology,




a reverse proxy is used to bundle all communication between the HMI and the back-end BBs.

This chapter introduces the core technologies and paradigms for the transfer of data within the FACTS4WORKERS system.





Communication and routing

-  *Hypertext Transfer Protocol (HTTP)*
-  *JavaScript Object Notation (JSON)*
-  *Representational State Transfer (REST)*
-  *Reverse Proxy*
-  *NodeJS*

Front-end

-  *Hypertext Markup Language (HTML)*
-  *Cascading Style Sheets (CSS)*
-  *Angular 2*

Back-end

-  *Ruby*
-  *Spring*
-  *Phoenix*
-  *Flask*
-  *Go*

4.2.1 Communication and routing

Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is a stateless protocol for the exchange of data in distributed, hypertext information systems, such as the World Wide Web. It can be used in any system for the exchange of data between distributed units [*Internet Engineering Task Force, 2015*]. HTTP takes care of addressing objects using a uniform resource identifier (URI), handling the interaction between the client and the server and adjusting the formats between them. HTTP communication follows the request-response-pattern. Server responses contain a HTTP status code indicating the status of the transaction [*Internet Engineering Task Force, 2015*].

Status code	Meaning	Examples
1xx	Information	100: Continue 101: Switching protocols
2xx	Success	200: OK 202: Accepted
3xx	Redirection	301: Moved Permanently 302: Moved Temporarily
4xx	Client Error	400: Bad Request 401: Unauthorized 402: Forbidden 404: Not found
5xx	Server Error	500: Internal Server Error 502: Bad Gateway 503: Service Unavailable

Figure 7: HTTP status codes

Representational State Transfer

Representational State Transfer (REST) is an architectural style for creating and organizing distributed systems. While the main application field lies within the interconnection of web services, it can be applied on any distributed system using any transfer protocol as long as the following constraints are met [Doglio, 2016]:

- 🧑 **Client-Server:** One principle is the separation of front-end and back-end software code. This enables the independent development of both sides improving the flexibility of the application.
- 🧑 **Stateless:** Communication between server and client is stateless. This means, that every request from client to server needs to contain all information needed for the server to understand the request without accessing stored data.
- 🧑 **Cacheable:** This constraint imposes that every response to a request has to be cacheable, allowing the system to store and bypass certain requests in order to favour others.
- 🧑 **Uniform Interface:** By demanding a uniform interface between the system components, interaction with the system is simplified. Implementation of the client is independent from the server providing a clear set of rules to follow.
- 🧑 **Layered System:** As REST is designed for web applications, it is expected that a respective software implementation works properly, even when dealing with high traffic volumes. By separating the system components into layers, the complexity of the system is reduced. Especially for large systems, this provides a major benefit.

JavaScript Object Notation (JSON)

The JavaScript Object Notation is a compact and lightweight format for the exchange of data. Based on the syntax of the JavaScript programming language, it was designed to be easy to read and write. Despite the name, the format is completely independent from any programming language. JSON strings can be built as objects or arrays [JSON, 2016]:

Objects are unordered lists of name/value pairs enclosed by curly braces. Inside the braces, name/value pairs are separated by commas. The values are split up from their name using a colon.

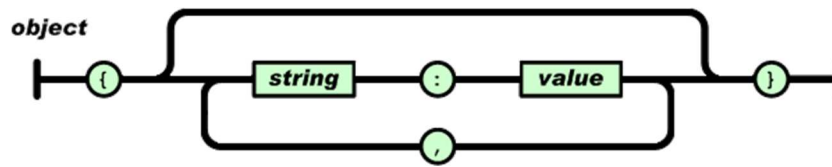


Figure 8: JSON objects [JSON, 2016]

Arrays are ordered lists of values enclosed by square brackets. The values are separated by commas.

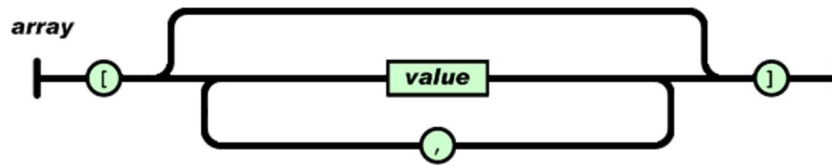


Figure 9: JSON arrays [JSON, 2016]

NodeJS

NodeJS is based on the Google V8 engine, an open source project that compiles JavaScript to native machine code. The deployment of a compiled language allows usage of JavaScript for server or back-end applications. Additionally, programming principles like the event driven approach, known from front-end applications, can be used for back-end code. This results in some great advantages making use of this programming language [Doglio, 2016]:

- 🔗 **Asynchronous programming:** NodeJS is designed for programming applications that make use of the asynchronous programming method, making the implementation of its principles easily.
- 🔗 **Simplicity:** Based on a the popular and wide spread language JavaScript, it is easy to learn and understand NodeJS.
- 🔗 **Effective integration of JSON-based services:** Since JSON and NodeJS are based on the same syntax, JSON based services can be integrated easily and effectively.

For the semantic workflow engine, the programming language NodeJS is used due to its asynchronous character. On the front-end side, this allows to process request as they arrive. Thereby, threads waiting on input from the HMI can be suspended to process other request in the meantime. On back-end side, the SWE does not expect an answer for every communication with a building block. Instead, the process flow directly moves forward and reacts on responses as they arrive. This concept speeds up the overall process flow, enabling large amounts of requests in a short time period.

Reverse proxy

A reverse proxy, as the one used within the FACTS4WORKERS solution, requests resources for the client from one or more servers and returns them, as they only come from the proxy itself. Since the proxy is visible over the internet connection, it is not necessary irrelevant that the client knows the storage location of the specific resources within the network [Du et. al., 2011].

4.2.2 Front-end

Hypertext Markup Language

The Hyper Text Markup Language (HTML) is, as the name indicates, a plain text mark-up language mainly used for web pages and web applications. The purpose is to structure the content of the document and to provide a semantic description using a standardized structure [World Wide Web Consortium, 2014].

To meet with the increasing demand for interactive content, HTML5 implements different APIs, making the usage of proprietary plug-ins or custom programming for basic APIs obsolete. The most common APIs cover the usage of multimedia content or WebSockets for the open connection between the client and the server in real time [World Wide Web Consortium, 2014].

Cascading Style Sheets

While HTML defines the structure of a web document, Cascading Style Sheets (CSS) define its appearance. The idea is not to apply styles directly to the individual elements, but to store the styling rules in a separate document and reference them to the HTML elements offering the following advantages [Powers, 2013]:

- A clearer, less-cluttered HTML code leads to advantages while authoring, reading or maintaining it.
- Defining or adapting the look of the document across several pages can be done by editing just a single source.

 Authors have an extended control over the look of the document.

One important feature of CSS is the way the rules are applied. While a web browser includes the styling rules into the HTML document, commands get executed one after the other. This makes it possible to first define a basic style for the whole document and later on define separate rules for specific areas whereby the initially rules are overwritten [Powers, 2013].

Angular 2

Created in 2009, AngularJS is an open-source client-side web application framework based on JavaScript. The main goal is to promote the productivity during web development and lower necessary efforts for building a natural and fluid looking web application. To achieve this, Angular extends the HTML syntax by adding new vocabulary and functionalities. In 2016 the newest version – Angular 2.0.0 – was released [Branas, 2013].

When a web page is loaded, a browser sends a request and waits for the response. When a HTML document or parts of it are received, the browser analyses its content and builds the Document Object Model (DOM) tree. Afterwards, the AngularJS compiler analyses the DOM model regarding special elements known as directives. A directive is an extension of the HTML vocabulary that allows the creation of new behaviors, meaning that the developer can create reusable components looking like HTML syntax. By default, the Angular 2 framework provides a set of standard variables taking care of the data connection between the model and the view [Branas, 2013].

Within the project, Angular 2 is used as front-end programming framework due to its modularity and reusability. The structure of Angular encourages a strict separation of different modules and services. This encourages the reusability and allows the usage of the building blocks crossover the different use cases.

4.2.3 Back-end

As Docker encapsulates the programming code inside containers, different building blocks can be created using different programming languages and frameworks. Thereby, web frameworks support the creation of REST-APIs by providing a standard basis and standard solutions that can be adapted. Within the project we currently use Ruby on Rails, Spring, Flask, Phoenix, Go and NodeJS. These frameworks, or in case of Go and NodeJS the programming languages, are very common in usage, well documented and especially known for providing APIs. In the following, we provide an outline of the mentioned frameworks/languages with their characteristics, strengths and advantages within the project.

Rails

Ruby on Rails is a web development framework based on the programming language Ruby. It provides a time-tested approach to keep medium and large size applications clear and maintainable. The great advantage of Rails is that the framework contains almost everything needed for creation of a functional product [Raymond, 2007].

One of the most common tasks during the web development project is the creation of a web-based user interface to manage a relational database. Targeting this problem, Ruby on Rails introduces several features interesting for the project. This allows to implement a database-backed user interface in short time [Tate et al., 2006]:

- 🔗 **Active Record:** Part of Rails is the active record framework that is responsible for the object storage in the database. Based on a design pattern, Rails automatically discovers the columns in the database schema and attaches them using metaprogramming.
- 🔗 **Built-in testing:** Rails automatically creates extendable automated tests for simple applications.
- 🔗 **Three environments:** development, testing and production: Rails provides the three default environments development, testing and production. Since each of them behaves slightly different, it eases the application development cycle.

Within the project, Ruby on Rails was chosen because of the following advantages:

- 🔗 It is capable of handling database migrations, allowing to ship Docker containers that can autonomously handle database schema upgrades.
- 🔗 In development environment, Rails supports dynamic class reloading; removing completely the code-compile-deploy development cycle typical of other languages allows for a huge improvement in development velocity.
- 🔗 Ruby on Rails improves standard ruby testing framework with lots of web friendly checks. REST calls can be easily tested with few lines of code without the need of performing API call simulation with external tools.
- 🔗 The ruby community puts a huge effort in the way of handling software dependencies, libraries are called gems and are collected in a central repository. If needed, the right gem can be searched in just one place and added to a Rails project with just a line of code.
- 🔗 As an extra feature, Ruby on Rails containers are officially supported by Docker Inc.

With this in mind, we consider Rails as a safe and sure starting point for the Building Block development.

Spring

Developed in the programming language Java, Spring is a lightweight framework for building web applications. Well known for distributed transactions, it assures a high reliability and scalability. Designed as meta-framework, Spring can be implemented independently from any particular platform. Furthermore, Spring includes several mechanisms for assuring the safety of the created web applications. Based on its architecture the framework offers the following advantages [*Spring, 2016*]:

- Unit-tests can instantly be performed without the need of creating corresponding programs
- Spring offers a consistent framework for the data access
- Business-interfaces can be implement as Java-classes

Based on a modular structure, the framework offers a great amount of functionalities, which can independently be implemented, allowing the developer to freely choose the included range. Furthermore, Spring supports the integration of other frameworks easing development of applications [*Spring, 2016*].

Spring is a widely known, well-established framework. Within the project, Spring is used because of its wide range of embedded features, offering well tested standard solutions for many problems.

Phoenix

Written in Elixir, Phoenix is a web development framework for the creation of server side applications using the MVC pattern. Aiming for high productivity and application performance, the framework Phoenix is made up out of distinct parts with specified purposes [*Phoenixframework, 2016*]:

- The Endpoint handles the requests and dispatches them into the designated router.
- The Router parses the requests and sends them to the controller.
- The Controller provides the actions or functions in order to handle the requests.
- The Views act as view-layer and render templates.
- The Templates are precomputed models minimizing the programming efforts.
- The Channels manage the sockets in order to provide real-time communication.

Phoenix is elixir's "Ruby on Rails". With the exception of the official support by Docker, all the Rails benefits also applies to phoenix. When dealing with the Data Analytics BB, we were concerned about the response time and the number of concurrent requests. This BB should provide near real-time information about the production

lines in order to display control charts and detect potential errors. To meet this requirement, Phoenix was chosen because of its short response time and the possibility to process a large number of concurrent requests.

Flask

Flask is a micro-framework for the programming language Python, meaning that it aims to keep the core of the framework small and only provide the basic services. At the same time, the framework is highly extensible allowing the programmers to choose the extension packages needed [Grinberg, 2014].

Within the Flask framework, there is no native support for validation web forms, accessing databases or other high-level tasks. These key services and others most web applications need are included through extensions and integrate with the core functionalities giving the developer the power to choose the ones which fits best the current needs [Grinberg, 2014].

Within the project, Flask is used due to being a minimalistic framework, including only absolutely necessary extensions. To support full functionality, needed services like REST-APIs can easily be included. With the focus on being light-weighted, Flask focuses on easy customization. This allows to adapt the framework to the individual needs of the BB.

Go

Go itself simply offers a way to handle incoming HTTP request, all the rest is up to the developer. There are object relational mapping libraries that can help to convert data between incompatible type systems, but database migrations still have to be handled manually.

The lack of many advanced features brings a very small memory footprint, a native support for multithreading and a low-level access to the HTTP body.

Go has been chosen as the language for Multimedia BB, because the API definition is really simple and from the kind of data handled arises complexity. File upload in JSON-based REST API is a tough matter. Frameworks usually want to impose multipart/form-data encoding which is typical for a web form. Go offers any kind of HTTP body parser, allowing us to use a multipart/related composed of JSON metadata and the uploaded file.

Although being a very simple and low-level language, Go features some very rich testing libraries, allowing us to obtain the same expressivity in tests of Ruby on Rails and phoenix.

As an extra feature, Go containers are officially supported by Docker Inc.

4.3 Semantic workflow engine

4.3.1 Introduction

Modern shop floors rely on a large amount of services provided from different sources that have to work together properly in order to fulfill the business goals. As in general the diversity and number of production machines and devices increases constantly, the integration of new units to the existing systems becomes increasingly complex as well. To deal with this problem, the usage of workflow systems effectively reduces the needed tasks for setting up of new machines and services by making the manual integration obsolete.

4.3.2 Strength of the Semantic Workflow Engine within the Project

By giving access to the building blocks through RESTful Web APIs, which semantically describe the functionality of the service, the semantic workflow engine (SWE) is able to understand these descriptions and combine them in order to achieve a desired goal. Thereby, the system is completely flexible, so that the workflow engine can take interchangeable BB and combine them to achieve the needed functionality [Arndt *et. al.*, 2016].

Overall the semantic workflow engine ensures the following advantages:

- 🔧 The semantic workflow engine eases the maintenance of the workflows. New components can be integrated by adding their functional descriptions and no longer required components can be removed by deleting these descriptions.
- 🔧 New tasks can easily be integrated using existing building blocks, as the SWE is aware of the BB functionalities. After defining the goal of the new tasks, the SWE is capable of automatically combining the needed BBs to achieve this goal.
- 🔧 The easy implementation lightens the deployment process. This favors to start with a first implementation using a small set of building blocks and then enrich is over time.
- 🔧 When a building block does not respond, another can take over.

4.3.3 FACTS4WORKERS example for the semantic workflow engine

For better understanding of how the semantic workflow engine works, we provide an example:

Starting point is an autonomously produced product. However, sometimes the product does not meet the quality requirements or is defective. In these cases, the worker is responsible for noticing, and if possible finding a solution and correcting the problem. Without the help of the FACTS4WORKERS solution, the worker would have to talk to colleagues and search for help till someone is able to fix the problem. This is time consuming and leads to the dissatisfaction and demotivation of the worker.

Even with a supporting information system following a static workflow, the worker would always need to follow the systems workflow to access the needed information. Thereby, the worker would have to search the current issue manually in a fault database and select the most fitting error in a list of all possible errors. Afterwards, the worker would need to review possible solutions and choose the most promising.

By using the SWE, the workflow can be adapted dynamically. Doing so, the SWE needs two inputs: a concrete precondition and a wished outcome. Let us assume the worker detects a product error (e.g. material defect) he is not familiar with (precondition). In this case, the worker is looking for input about how to fix this issue (wished outcome). Having these two inputs, the SWE will dynamically create a workflow to achieve the wished outcome.

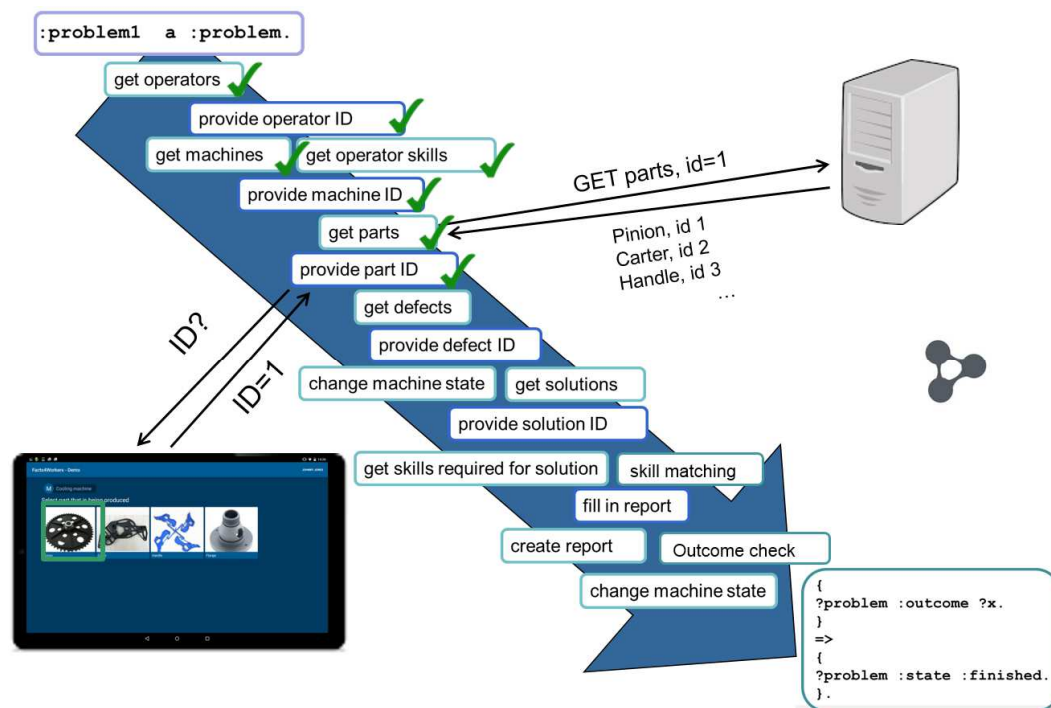


Figure 10: SWE schema

In addition, if there is only one problem and one solution that meets the current issue, there is no need for showing the worker a list of all possible errors and solutions. Instead, the problem and solution descriptions will be displayed immediately with no need for the execution of the steps in between, reducing the required interaction and

therefore preventing an information overload. Besides, using a dynamic workflow allows to easily extend or alter the workflow, for example by taking the workers skill level on the specific product into account. If the worker is not experienced enough to fix the problem himself, the semantic workflow will directly offer the worker an option to contact a person that is available and trained enough to solve this specific problem.





In conclusion, the SWE can be used in cases where different workflow can be used for solving a problem. The SWE is like a navigation system that guides the user on the fastest and easiest way to his goal.

4.4 Docker

4.4.1 Introduction

Docker is the selected tool for the deployment of the FACTS4WORKERS solution, consisting of several BBs that are configured and deployed within the different industry partner use cases. Docker uses so called “containers”, which capture everything that is needed to run a chosen software (e.g. code, runtime, system tools, system libraries, binaries, dependencies, etc.). Docker containers represent one encapsulated unit of functionality to the “external world”. In this way, it is assured that the code will run in any selected environment the same way [Docker, 2016].

By using libcontainer, cgroups and other lower level Linux components, Docker containers run isolated without the need of a fully build operating system. However, these components are typically used for process isolation in Linux. Therefore, Docker currently is mostly used on Linux distributions [Janetakis, 2015]. Since mid-2016 Docker is also available for Microsoft Windows Server operation systems, limited to the versions Windows Server 2016, Nano Server, Windows 10 Professional and Enterprise [Docker for Windows, 2016]. However, since Docker for Windows requires special Windows containers, we decided to support only Linux containers. Concerning the integration and deployment workflow, the Docker approach provides the following benefits:

-  Support of developers by simplifying options to submit their software development work for deployment in different scenarios and use cases
-  Bundling of application and needed operating system in one specific image
-  Use of packaged parts of an application for testing and distribution to particular target-environments
-  Reduction of dependencies from specific hardware requirements

Through the usage of Docker containers, developers can build their software without consideration of specific settings of the target environment and deployment issues. Developers simply create and test software in their development environments and then form a Docker container of all needed software and dependencies. Docker containers are shipped as a whole to system integrators for implementation in the environment. By using this approach, developers and system integrators reduce their need to do iterative adaption cycles together. This saves considerable efforts in an agile software development cycle. [Matthias *et. al.*, 2015].

4.4.2 Containers versus VMs

In general, a container is an independent, isolated environment that is used to execute the contained software. There are some distinct advantages of the Docker technology in comparison to the well-known Hypervisor or Virtual Machine (VM) technology. The following paragraphs outline, why the decision was made for the use of Docker within FACTS4WORKERS.

Resource Efficiency

VMs need considerable resources, which means the performance of the system decreases significantly with the number of deployed VMs on a host system. As shown in Figure 11, to run three applications with different requirements it is necessary to run three distinct VMs on a host system and a complete operating system on each VM. Depending on the hardware, it is only possible to run a limited number of VMs on one host system

In contrast to VM, Docker uses the kernel of a host system together with other containers [Matthias *et. al.*, 2015]. Hence, there are no additional operating systems required that consume host system resources. Consequently, a lot more Docker container units can be deployed on the same hardware than it would be possible with VMs. [Raj *et. al.*, 2015] This makes the usage of containers highly efficient concerning the usage of hardware resources, as there is no need for a distinct operating system for each isolated function [Mouat, 2015]. As shown in Figure 12, the same three applications as above could be deployed by the use of Docker containers that would use the same kernel as the host system.

Another benefit regarding resource efficiency is that in contrast to running an application natively on an operating system, Docker containers need significantly less overhead data. [Mouat, 2015]. Despite of these differences, both approaches are equivalent in all matters concerning the end-user experience. By the use of Docker, it is a lot easier to scale from a use case solution to a company wide solution, which is an interesting aspect for scalability and particularly interesting for the FACTS4WORKERS project.

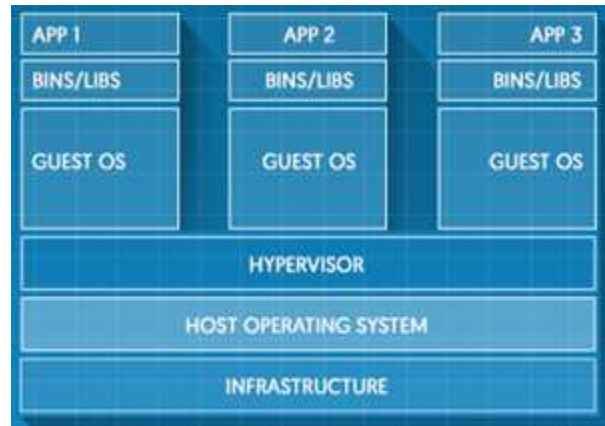


Figure 11: Structure virtual machines [Docker, 2016]

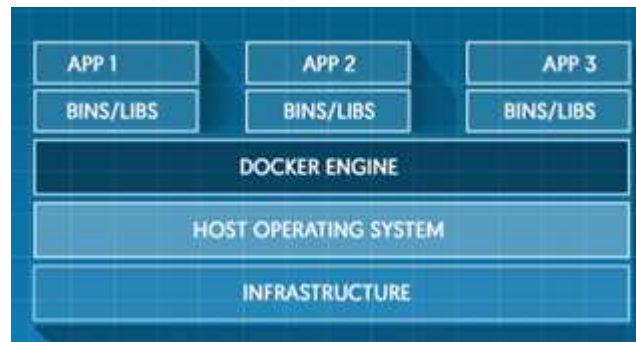


Figure 12: Structure of Docker containers [Docker, 2016]

Persistence

VMs are used especially for long-term abstractions of real hardware. On the opposite site, containers could be used to serve one particular task and be deleted when the task is no longer needed. Since the FACTS4WORKERS project follows an agile software development process, several steps are undertaken to create the final solution. Therefore, small containers serving a particular task are more suitable to distribute among partners than bigger sized VMs. [Matthias *et. al.*, 2015]

Isolation

Although Docker containers as well as VMs provide an isolation of applications from each other, the isolation provided by VMs can be seen as “one level higher”, as applications can even be isolated by different operating systems. VMs are completely isolated from the other VMs as well as from the underlying host system. Docker containers are isolated on process level and hence, they are liable for any kind of security incursion. As this point is critical, it is better to have a stronger isolation for software that is still in development. For this reason, a VM or dedicated server running Docker containers only for FACTS4WORKERS services is recommended. [Raj, *et. al.*, 2015]

Portability

For the use of Docker containers, it is only necessary to set up the container engine, distribute and deploy the respective container, which can be seen as a capsule around a process [Matthias et. al., 2015]. This characteristic makes containers easily movable, very compact, and clearly detached and requires only minor deployment effort [Raj et. al., 2015]. When using VMs on the opposite, it is necessary to set up an individual VM for each application, which usually takes considerable efforts, particularly because time-consuming configuration tasks are required to get the application running [Mouat, 2015].




In summary, this table lists the mentioned aspects in a clearly arranged way:

	Virtual Machines	Docker Containers
Resource Efficiency	Need an additional operating system. Only a few VMs can be deployed on a single physical host system.	Use the same Kernel as the host system. Many containers can be deployed on a single physical host system.
Persistence	Usually long-term oriented.	Can be used for several months or even to run containing application only once.
Isolation	Isolation starts at the level of the operation systems. Full isolation of VMs.	Isolation happens on process level. No isolated operation system
Portability	Require significant effort to distribute and deploy an application	Are easily distributable and deployable.

Figure 13: VMs versus Docker Containers

4.4.3 Strength of Docker within the Project

Concerning the deployment of the developed software solutions, the project “Facts4Workers” faces particular challenges:

-  Research partners of the project are spread over Europe and each partner has different competences and preferences concerning the used software to build the final solution.
-  Industry partners have running infrastructure and production lines that must not be disturbed. Therefore, each solution has to be designed as fully capable of being integrated in the existing system of each industry partner. Additionally, there are very different existing IT solutions on the partner sides.
-  The overall approach is to increase worker satisfaction and to develop solutions that fit as good as possible to the working process of the worker. This increases the complexity of the development process, as it adds a whole new level of requirements and limitations.

In consideration of these challenges, deployment of the build solutions needs close attention to reduce the possibility of error as much as possible. FACTS4WORKERS solutions will include back-end software code written in different programming languages as well as a combination of, for example, SQL-databases, JavaScript frameworks and other programming languages. In addition, these parts have to run on a variety of soft- and hardware environments at the industry partner locations. Hence, to meet the given requirements, the approach of Docker containers to standardize and modularize IT building blocks that perform a particular encapsulated task, is a very useful tool to use.

Usually the process of developing and deploying software can be a tedious task. There is normally an enormous demand for interdisciplinary exchange of different teams, which are highly dependent and have to understand each other's work [Matthias *et. al.*, 2015]. With Docker, these circumstances in the FACTS4WORKERS project are no longer a problem to handle. Implementation partners are able to develop their building blocks as full functional containers, that provide the required microservices. These microservices provide portability and flexibility that is needed to create the best suiting and efficient IT-processes FACTS4WORKERS aims for. Possible changes can be deployed quite quickly by simply changing the specific container, allowing parts of the solution to be improved without having to change the whole system. Usually even small changes in the environment of a native running application could cause several forms of errors or malfunctions. Using container services, small changes on a single container have no impact on the functionality of the whole application and furthermore, assure that no bugs can corrupt the system of the industry partner.

As containers include everything an application needs to run, they are highly independent from the given environment. After distributing a container, industry partners neither need to adapt their system nor have to make big efforts to configure the application. As containers are independent from their environment, once they are deployed, they can be run without substantial effort.

Of course, a system constructed of countless of these microservices includes a high complexity affecting the interaction between these services. Therefore, the deployment and implementation of the IT-solutions needs special attention to ensure, a stable and high-quality result.

In foresight, it may also be a benefit for projects in the future, that most of large public clouds have made their systems compatible to Docker (e.g. AWS Elastic Beanstalk, Google AppEngine, IBM Cloud, Microsoft Azure, Rackspace Cloud). Even Google is going to use Docker as their preferred container-building tool. This support leads to the matter of fact, that Docker will probably become the most prevalent system used to create cloud applications [Matthias *et. al.*, 2015]. Hence, if the IPs would like to deepen their work with Docker containers or expand the FACTS4WORKERS solution to more of their IT systems, they can immediately use one of the named services.

In summary, for the implementation of microservices Docker provides a suitable tool to deploy and manage the individual services.

References

Arndt, D.; Van Herwegen, J.; Verborgh, R.; Mannens, E.; Van de Walle, R.; 2016: Using Rules to Generate and Execute Workflows in Smart Factories, *in: Proceedings of the RuleML 2016 Challenge, Doctoral Consortium and Industry Track Hosted by the 10th International Web Rule Symposium, CEUR Workshop Proceedings*

Branas Rodrigo; 2014: AngularJS Essentials, Design and construct reusable, maintainable, and modular web applications with AngularJS, Packt Publishing, Birmingham – pp.3

Docker; 2016: <https://www.docker.com/what-docker> – 12.07.2016

Docker for Windows, 2016: https://msdn.microsoft.com/en-us/virtualization/windowscontainers/deployment/system_requirements - 27.07.2016

Doglio, Fernando; 2015: Pro REST API Development with Node.js, Apress, New York – pp.2

Du, Jiang; Nie, GuoXin, 2011: Design and Implementation of Security Reverse Data Proxy Server Based on SSL, Springer, Berlin, Heidelberg, p.2

Grinberg Miguel; 2014: Flask Web Development, Developing Web Applications with Python, O'Reilly Media, Sebastopol – p.3

Internet Engineering Task Force; 2015: Hypertext Transfer Protocol Version 2 (HTTP/2), Internet Standards Track document, ISSN: 2070-172 – pp.4

Spring; 2016: <https://spring.io/> – 27.06.2016

Janetakakis, Nick; 2015: Deploy web apps with Docker, Rescue yourself from the complexity of DevOps, O'Reilly Media, Sebastopol – p.1

JSON; 2016: <http://www.json.org/> – 27.06.2017

Matthias, Karl; Kane, Sean P; 2015: Docker Up & Running, Shipping Reliable Containers in Production, O'Reilly Media, Sebastopol, pp.1; pp. 8; p.15; pp.59; p.104

MIL-STD-810; 2016: http://everyspec.com/MIL-STD/MIL-STD-0800-0899/MIL-STD-810G_12306/ – 03.07.2017

Newman Sam; 2015: Building Microservices, Designing Fine-grained Systems, O'Reilly Media, Sebastopol – pp.18

Mouat, Adrian; 2015: Using Docker, Developing and Deploying Software with Containers, O'Reilly Media, Sebastopol – pp.3; pp.96

Powers Davis; 2012: Beginning CSS3, Mastering the language of web desing, Apress, New York – pp.3

Raj, Pethuru; Chelladhurai, Jeeva S.; Singh, Vinod; 2015: Learning Docker, Packt Publishing Ltd, Birmingham, – p.19; pp.33t

Raymond Scott; 2007: Ajax on Rails, Build Dynamic Web Applications with Ruby, O'Reilly Media, Sebastopol – pp.10

Phoenixframework; 2016: <http://www.phoenixframework.org/docs/overview> – 22.06.2016

Summerfield Mark; 2015: Programming in Go, Creation Applications for the 21st Century, Addison-Wesley, New Jersey – pp.1

Tate Bruce A., Hibbs Curt; 2006: Ruby on Rails, Up and Running, O'Reilly Media, Sebastopol – pp.1

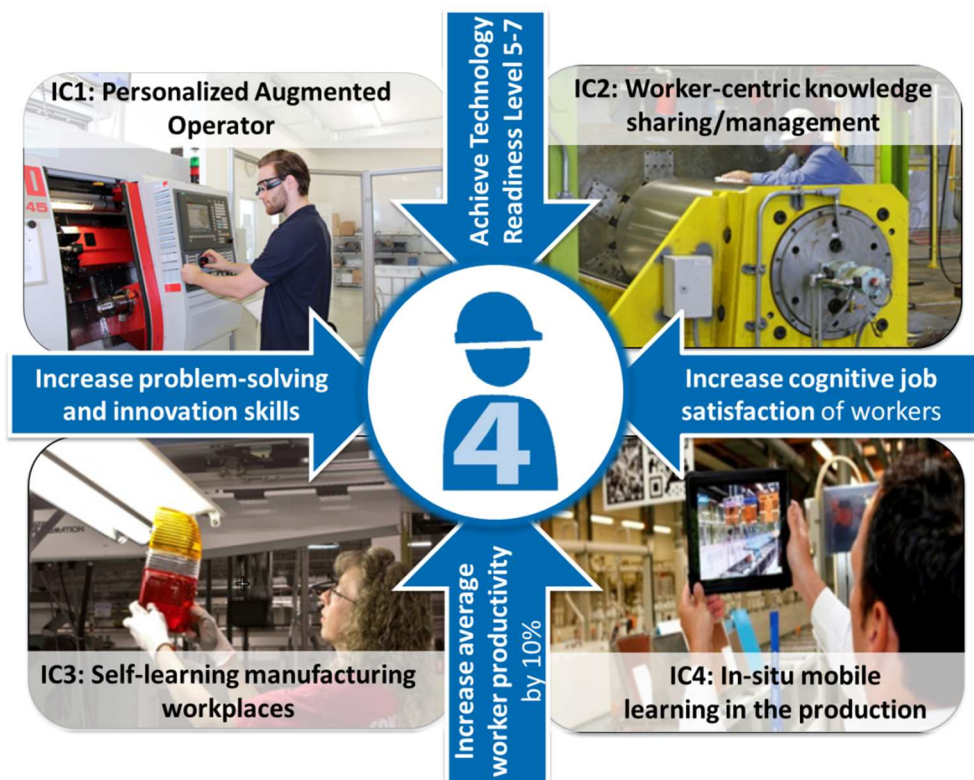
World Wide Web Consortium; 2014: HTML5, A vocabulary and associated APIs for HTML and XHTML, <https://www.w3.org/TR/html5/> - 27.06.2017

ABOUT THE PROJECT

The high ambition of the project FACTS4WORKERS is to create Factories of the Future with a pervasive, networked information and communication technology that collects processes and presents large amounts of data. These smart factories will autonomously keep track of inventory, machine parameters, product quality and workforce activities. But at the same time, the worker will play the central role within the future form of production. The ambition of the project is to create »FACTories for WORKERS« (FACTS4WORKERS), to strengthen human workforce on all levels from shop floor to management since it is the most skilled, flexible, sophisticated and productive asset of any production system and this way ensure a long-term competitiveness of manufacturing industry. Therefore, a serious effort will be put into integrating already available IT enablers into a seamless and flexible Smart Factory infrastructure based on work-centric and data-driven technology building blocks.

These solutions will be developed according to the following four industrial challenges, which are generalizable to manufacturing in general:

- Personalized augmented operator,
- Worked-centric rich-media knowledge sharing management,
- Self-learning manufacturing workplaces,
- In-situ mobile learning in the production.





















PROJECT PARTNERS

The FACTS4WORKERS project is composed of 15 partners from 8 different European countries:

Virtual Vehicle Research Center	Austria
Hidria TC Tehnološki center d.o.o.	Slovenia
Università degli Studi di Firenze, Department of Industrial Engineering	Italy
Technische Universität Wien	Austria
ThyssenKrupp Steel Europe AG	Germany
Hidria Rotomatika d.o.o.,	
Industrija Rotacijskih Sistemov	Slovenia
iMinds VZW	Belgium
Sieva d.o.o.	Slovenia
University of Zurich, Department of Informatics	Switzerland
Thermolympic S.L.	Spain
EMO-Orodjarna d.o.o.	Slovenia
Evolaris Next Level GmbH	Austria
Itainnova - Instituto Tecnológico	
de Aragón	Spain
Schaeffler Technologies AG & Co. KG	Germany
Lappeenranta University of Technology	Finland

PROJECT COORDINATOR / CONTACT:



VIRTUAL VEHICLE Research Center

Inffeldgasse 21A

8010 Graz, AUSTRIA

Tel.: +43-316-873-9077

Fax: +43-316-873-9002

E-Mail: facts4workers@v2c2.at

FOLLOW US AT:

 **FACTS4WORKERS**
 **@FACTS4WORKERS**
 **facts4workers-project**



Blueprint architecture and integration plan

The deliverable shows the approach the project FACTS4WORKERS has taken to develop, deploy, integrate and test the worker centric software solution. It with the description of the integration plan on a generic level and points out the necessary steps in order to transfer the individually developed and tested services to an integrated functioning system. Following the agile approach of the project, integration steps will be performed iteratively improving the solution step by step and increase its maturity continuously. This concept also allows to rapidly integrate feedback from users enabling the worker centric approach. The second part of the deliverable provides an overview over the deployment and testing phase. Thereby the different maturity levels

prior to release are summarized. This concept allows to introduce the FACTS4WORKERS solution in several steps and constantly gather feedback from the workers. Referring to the feedback, the second chapter also states the testing activities undertaken to create a worker-centric solution. The final part provides an overview over the hardware and software architecture of the FACTS4WORKERS solution. First, the required hardware technology for deploying the solution on the shop floor is stated before the software technology, frameworks and programming languages are outlined. The deliverable finishes with the description of the tool Docker, which serves as enabler for the concept of building blocks.

