Deliverable D7.2 Verification and validation methods

WP7

Document information

Project Identifier	ECSEL-2017-783221
Project Acronym	AFarCloud
Project Full Name	Aggregate Farming in the Cloud
Document Version	11
Planned Delivery Date	30/4/2019
Actual Delivery Date	15/5/2019
Deliverable Name	D7.2
Work Package	7
Task	7.2
Document Type	Report
Dissemination level	Public
Abstract	This document describes the integration and verification methodology and the overall validation plan that will serve as the point of reference during the development, integration and demonstration of the AFarCloud components and overall platform.

Document History

Version	***Date***	***Contributing partner***	***Contribution***
1	15/10/2018	INTRA	ToC
2	25/11/2018	INTRA	1st Draft
3	28/01/2019	INTRA, Tecnalia, UPM	2nd Draft
4	25/02/2019	INTRA, UPM	3rd Draft
5	19/3/2019	UPM	Comments to 3rd Draft
6	20/3/2019	INTRA	Integration of comments
7	19/4/2019	INTRA	Pre-final Draft
8	24/4/2019	AIT	Review and contribution
9	25/4/2019	BEV	Review
10	26/4/2019	UPM	Review and contribution
11	29/4/2019	INTRA	Final

Document Contributors

Partner name	***Partner member***	***e-Mail***	***Skype ID/Phone number***
Intrasoft International	14	*Theofanis.Orphanoudakis@intrasoft-intl.com*	thorphan
Intrasoft International	14	*Dimitrios.Skias@intrasoft-intl.com*	dimskias
Tecnalia	3	*belen.martinez@tecnalia.com*	
Tecnalia	3	*Leire.Orue-Echevarria@tecnalia.com*	
Tecnalia	3	*sonia.bilbao@tecnalia.com*	
UPM	1	*Jesus.rodriguezm@upm.es*	j_rodri_mo
UPM	1	*mv.beltran@upm.es*	
UPM	1	*gregorio.rubio@upm.es*	
UPM	1	*lourdes.lopez@upm.es*	
UPM	1	*vicente.hernandez@upm.es*	
UPM	1	*jf.martinez@upm.es*	
UPM	1	*pedro.castillejo@upm.es*	
HUA	58	*vdalakas@hua.gr*	vdalakas
HUA	58	*tsadimas@hua.gr*	
AIT	15	*erwin.kristen@ait.ac.at*	
BEV	23	*dario.pedro@beyond-vision.pt*	
SINTEF	26	*mariann.merz@sintef.no*	+47 93059358
SINTEF	26	*gorm.johansen@sintef.no*	+47 92228551
SENSOWAVE	8	*candres@sensowave.com*	

Table of Contents

List of Figures

List of Tables

Definitions and Acronyms

Acronym	Definition	Remark
CI	Continuous Integration	
DevOps	(Development and Operations)	
AT-UC	Austrian - Use Case	
CoAP	Constrained Application Protocol	
CD	Continuous Delivery	
CSRS	CyberSecurity Requirement Specification	
CPS	Cypher Physical Systems	
DSS	Decision Support System	
DoW~~A~~	Description of Work	
FMVEA	Failure Mode and Vulnerability Effect Analysis	
GIS	Geographic Information System	
GNSS	Global Navigation Satellite System	
IT	Information Technology	
IG	Integration Guidelines	
	International Software Testing Qualifications Board	
JSON	JavaScript Object Notation	
KPI	Key Performance Indicator	
Le~~E~~SS	Large Scale Scrum	
MQTT	Message Queuing Telemetry Transport	
MMT	Mission Management Tool	
Nordic CDX	Nordic Cattle Data eXchange	
OAS	OpenAPI Specification	
OS	Operating System	

REST	Representational State Transfer	
SL	Security Level	
SL_A	Security Level Archived	
SL-C	Security Level Capability	
SL-T	Security Level Target	
SCM	Source Code Management	
SuC	System under Consideration	
SoS	Systeme of Systems	
TDD	Test Driven Development	
TARA	Threat Analysis and Risk Assessment	
UAV	Unmanned Area Vehicle	
UGV	Unmanned Ground Vehicle	
WSN	Wireless Sensor Network	
YAML	Yet Another Markup Language	

Table 1: Definitions, Acronyms and Abbreviations

Executive summary

This document describes the integration and verification methodology and the overall validation plan that will serve as the point of reference during the development, integration and demonstration of the AFarCloud components and overall platform. The deliverable starts with the description of the design, development and integration approach addressing both the background methodologies of complex system development and their practical implementation following the project structure. Then the integration and verification plan is described addressing aspects like component development and verification practices and overall plan towards system integration into local and holistic demonstrators. After demonstrator integration the validation strategy that should be implemented into the different demonstrators is included addressing relevant KPIs as well as security assessment practices. Finally, detailed references and guidelines to technologies and tools that will be used for continuous integration throughout the AFarCloud platform lifecycle are given.

1. Introduction

In this section we make an overview of the scope and structure of the document.

1.1. Scope of the document

The purpose of this document is to describe the steps towards the integration of the AFarCloud platform components, their verification and their validation during the demonstration phases of the project.
The current document is the deliverable D7.2: Verification and validation methods and is the first outcome of Task T7.2 Validation methodology and system integration of work package WP7: Demonstrators Definition, Integration, Verification and Validation

1.2. Structure of the document

The rest of the document contains the following sections detailed below:

Section 2. Design, development and integration methodology
In this section we describe overall project approach and the basic principles of complex system development that apply to the AfarCloud platform development laying out the basic steps towards system design, integration verification and validation.

Section 3. Integration and verification plan
In this section we describe the Integration and verification strategy and the initial plans for AfarCloud platform releases (related to the plans for the development of the AfarCloud local and holistic demonstrators) and the definition of the initial product backlog towards integration and demonstration.

Section 4. Validation strategy
In this section we describe the validation strategy and the initial identification of KPIs that will be used to validate the main platform releases during periodic AfarCloud platform demonstrations.

Section 5. Technologies and tools
In this section we describe respective technologies and software development practices for the AfarCloud platform components that will be developed and integrated as well as the tools to be used for maintaining the AfarCloud platform code repository and issue management and development team collaboration.

2. Design, development and integration methodology

The envisaged AFarCloud platform aims to promote novel precision farming solutions by providing cyber physical systems (CPS), as well as a monitoring and sensing framework able to utilize new autonomous robotics platforms and incorporating the legacy systems already deployed in the farms. In this context the AFarCloud platform can be classified as a directed type of a Systems of Systems (SoS) architecture. As such AFarCloud develops a set of innovative software components and hardware adaptations to set the baseline for a new generation of cooperating CPS for smart farming.

In this deliverable we will not address specific hardware system development. Each hardware component will be considered as confined in the work package where it is developed. Thus, design, assembly and partial integration, verification and testing of hardware components needs to be completed before integration in the demonstration platforms, during each AFarCloud platform integration phase and hardware modules need to be initially considered functioning according to specifications. Of course, during overall system integration and verification in the case of identified errors in the expected way of operation of components developed within other work packages, these will be reported and fed back to the relevant work package. The focus of this deliverable is on AFarCloud platform integration in the form of intercommunicating and interoperable software programmable components that functioning together support and implement the Farming-as-a-Service AFarCloud concept.

2.1. AFarCloud project strategy

As mentioned above the AFarCloud platform ultimately represents a System-of-Systems, (SoS) comprising cyber physical systems (CPS), as well as a monitoring and sensing framework. This development will be performed by 60 different partners carrying out development work divided in 5 different technical work packages delivering their outputs to work package 7 for system integration, validation and demonstration. Software component development in AFarCloud will follow an Agile methodology, characterized by being iterative and incremental while focusing on a product mindset instead of a project mindset.

To this end the final AFarCloud products are expected to be delivered, integrated, demonstrated and validated through the planned 8 local and 3 holistic demonstrators. The AfarCloud project's strategy for demonstrators is bottom-up, in the sense that only the functionality (and SW components) that are tested in local demonstrators will be deployed for the holistic demonstrator as shown in Figure 1.

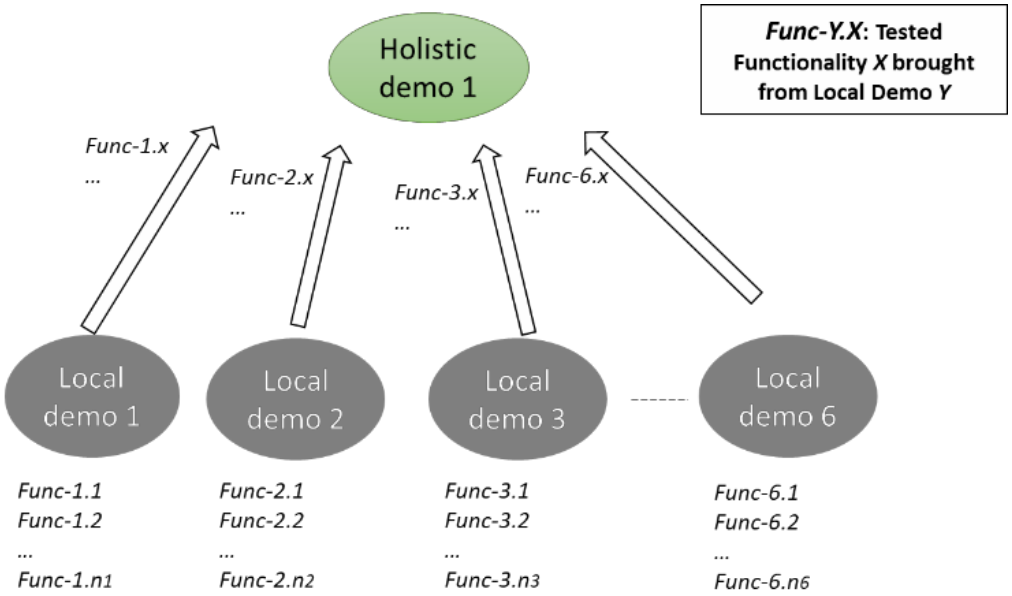

Figure 1. AFarCloud demonstration-centric project strategy

The goal is to select the local demonstrators' functionalities that meet the subset of user requirements that we would like to demonstrate at the holistic demonstrators. Here, a demonstrator functionality (Func-Y.X) is a high-level functionality that can be implemented in a demonstrator (it's not technological) as for example "*Measure the level of soil humidity*".

Within this strategy, a clear well-defined definition of each task in WP2 and WP7, beyond the technical development of components in WP3-6, is necessary. As shown in Figure 2 below, WP2 is responsible for delivering a set of user requirements (from the end-user perspective, not technological) and the AfarCloud platform's architecture (i.e., a definition of SW components and their dependencies). These two inputs are taken by WP7 in order to enforce a demo-centric strategy for the project through the 8 local demonstrators and 3 holistic demonstrators.

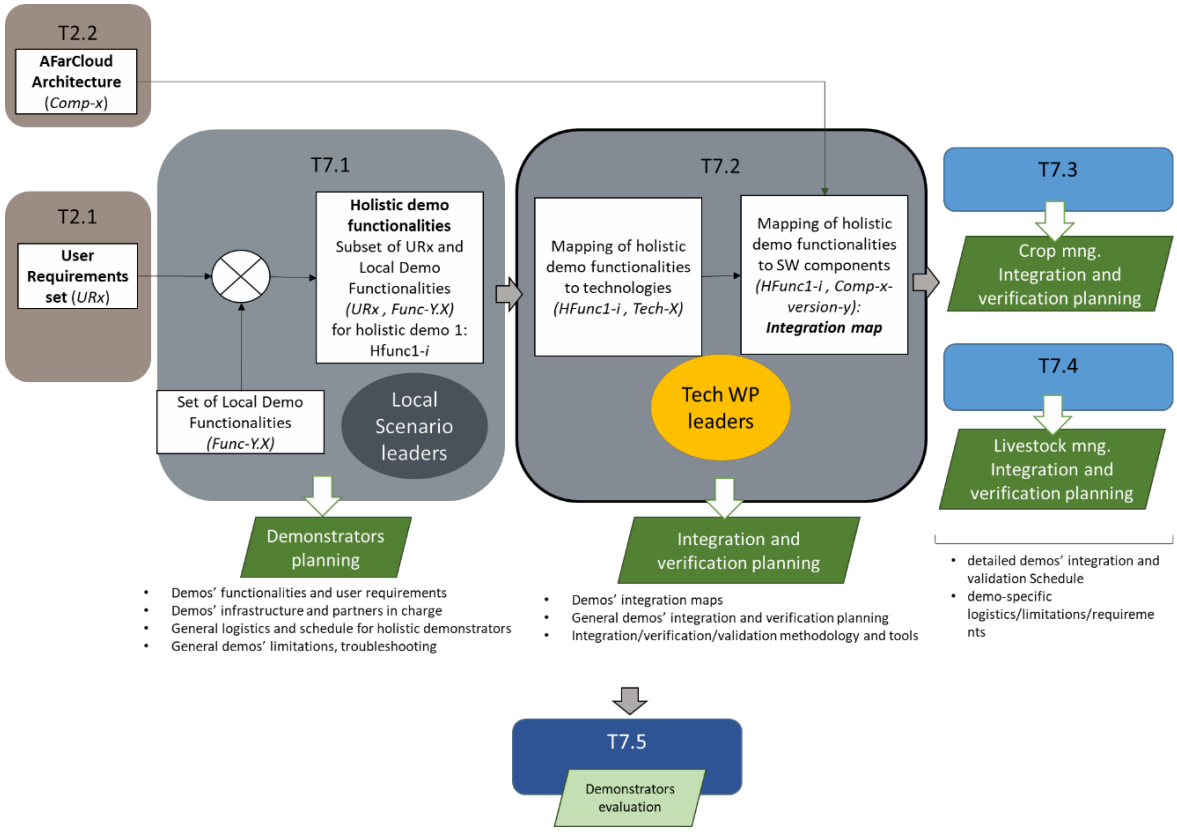

Figure 2. AFarCloud detailed plan towards integration

Within WP7, T7.1 will rely on demonstrator's leaders to define the functionalities that should be implemented in each local demonstrator. From here, a demonstrator plan will be designed for the local and holistic demonstrators. The demonstrators planning will at least include the functionalities and user requirements to be implemented in each demonstrator as well as the general logistics and schedule for the holistic demonstrators (i.e., safe period for testing, necessary infrastructure HW, HW providers, etc.). This means that the demonstrator planning will describe the general strategy of all the demonstrators towards the success of the holistic demonstrators. As shown in Figure 2 the output of this task will include among others a detailed list of functionalities in a pre-defined format to be delivered to T7.2. In turn the overall demonstration planning together with the integration plan will be taken by T7.3 and T7.4 as the foundation to elaborate their respective detailed demonstrators planning for the crop management and livestock scenarios, respectively.

T7.2 will strongly coordinate with technical WP leaders (i.e., WP2-6) to translate the demonstrator planning from T7.1 into a set of integration maps (one for each demonstrator, each year) that have the ultimate goal to drive local demonstrators towards the holistic one, every year. To this end, the functionalities of each demonstrator (provided by T7.1) will be translated to the required technologies with the help of WP leaders. In turn, technologies will be mapped to the SW components of the architecture, with the help of WP leaders too. Technologies and the corresponding SW/HW components in the AfarCloud architecture will be selected based on the confidence level of WP leaders about the SW/HW component being ready for the holistic demonstrator, that is meeting the

schedule of the local demonstrator's plan towards the holistic one. Integration maps will include the partners providing the specific component of the architecture, the version of each component and dependencies between components. T7.2 will provide the integration and verification planning that will include at least the integration methodology (as described in Section 3 below), verification and validation, general integration/verification/validation roadmap (that must be according to the general demonstrator planning from T7.1), KPIs, integration maps, technologies and functionalities for each demonstrator methodology (as described in Sections 4 and 5 below).

T7.3 and T7.4 will provide the detailed demonstrators strategy for the crop and livestock management scenarios based on the demonstrators planning from T7.1 and the integration and verification planning from T7.2. The crop management integration and validation planning from T7.3 and livestock management integration and validation planning from T7.4 will include the detailed deployment planning and schedule including integration activities and logistics as well as demo-specific limitations and requirements for the crop management and livestock management scenarios, respectively.

2.2. **Methodological framework for complex system development**

In terms of the AFarCloud core platform component development due the complex system development nature of the AFarCloud project and the collaborative effort involving self-organizing and cross-functional teams, developing components in parallel through the technical work packages of the project, an Agile software development is envisaged. An empirical knowledge work development/control framework, which was developed to serve Agile software development projects is Scrum [1]. In Scrum a cross-functional self-managing Team develops a product in an iterative incremental manner. Since in Scrum the emphasis is for one Team and not many Teams working together, Large-Scale Scrum (LeSS) (and LeSS-Huge serving better more than 8 Teams, 8 being an upper-limit empirical observation) has been proposed to operate Scrum at scale, applied to many teams working together on the same product [2]. In many aspects the technical development in AFarCloud will follow practices close to the principles of LeSS/Scaled Scrum, which will be reviewed in the remainder of this section.

2.3. **Terms and definitions**

When adopting Scrum/LeSS, an accurate definition of the end **Product** is required first, since it will affect the scope of the **Product Backlog**, who will be the **Product Owner** and the size (in teams) of the Product [2],[3]. In the context of this deliverable the product refers to the AFarCloud platform that will be demonstrated according to the selected functionality lists that will be adopted according to the demonstration plan provided by T7.1. An initial plan for the selected functionalities, which will evolve on a yearly basis is provided by T7.1, and the AFarCloud technical development is focusing on

decomposing these lists into technologies and components that will be delivered by the technical Workpackages and integrated on the local and holistic demonstrators to provide these functionalities.

Multiple teams building a single product work from a single **Product Backlog** that defines all of the work to be done on the product. Product Backlog Items are not pre-assigned to the teams. The LeSS Product Backlog is the same as in a one-team Scrum environment.

The Scrum team consists of three main roles; the Product Owner, the Scrum Master and the Development Team.

The **Product Owner** is responsible for the overall product backlog i.e. the complete list of functionalities that are required for the final demonstration of the AFarCloud platform. The Product Owner is responsible for the content of the product backlog before each development iteration, as well as the prioritization of the product backlog before each development iteration. The product owner should also ensure that the development team understands what is expected from the features of the backlog. Following the demo-centric approach of AFarCloud the Product Backlog shall be maintained in the form of functionality lists maintained by the demonstrator leaders as described in section 2.1.

On the other hand, the **Scrum Master** is the person who owns the Scrum process. This person is responsible to ensure that the team adheres to the Scrum theory, practices and rules and is the facilitator of the Scrum events (meetings / reviews) as well as to remove any obstacles to the team and enable the communication between the team members. In AFarCloud the Product Owner shall be the Task 7.2 Leader.

Finally, **the development team** is the one who owns the software. It is self-organising, meaning that no one can tell the development team, not even the scrum master, how to take the backlog of features and turn them into increments of working software. The team is also cross-functional, consisting of everyone and everything that they need to complete the development iteration and the overall product and does not depend on anyone who is not part of the team. Its members do everything from coding to testing and documenting while all members are equally responsible for the software quality, the technical implementation of features driven by the storyboards and the delivery of the product increment at every development iteration. In AFarCloud the development teams are represented by the consortium member teams involved in the development of components of each Work package to be contributed to the AFarCloud Product Backlog.

In LeSS Huge one additional role is introduced. Each Requirement Area has an **Area Product Owner** who specializes in that area and focuses on its Area Product Backlog. The Product Owner may also serve a double duty as an Area Product Owner for one area. Additionally, in LeSS Huge the Product Owner groups every Product Backlog item under exactly one requirement category—its requirements area leading to the generation of different views on the overall Product Backlog—called an **Area Backlog**. The Area Backlogs are prioritized by an Area Product Owner who specializes in part of the product. Each Requirement Area has several feature teams working from the Area Backlog. In AFarCloud the Requirement Areas are represented by the technical Work package definitions overseen by the project's Technical Committee (i.e., WP leaders and the Project Coordinator).

According to the above methodology the Scrum team can create and use other artifacts like user roles, workflow descriptions, user interface guidelines, storyboards, or user interface prototypes to

complement the product backlog. In AFarCloud this is related to the maintenance of the architectural maps, component list definitions, integration maps that include component interfaces and data flows and the demonstrator plans that will determine the sprint planning as will described in more detail in the following sections.

3. Integration and verification plan

3.1. Overall Methodology

In AFarCloud verification aims at guaranteeing that the components delivered by the technical Work packages during partial system integration phases meets the set of design specifications defined during the architectural specification phase and cover the respective items of the Product Backlog. During intermediate development and partial system integration phases, verification procedures will be developed. Such verification procedures should define specific test scenarios modelling or simulating system components up to the complete AFarCloud platform and methods for evaluating and analysing the modelling results in order to guarantee operation according to specifications. AFarCloud adopts a DevOps approach for development, integration, testing and deployment. In section 5.2, integration framework and tools are described. In addition, guidelines on adopting specific tools for the purposes of AFarCloud are provided. Verification is tightly coupled to the AFarCloud system development and integration methodology as described in the remainder of this section

3.1.1. Continuous Integration (CI) principles

Continuous Integration is a developer practice to keep a working system by small changes growing the system by integrating frequently (usually at least daily) on the mainline by means of appropriate tools supporting automation with lots of automated tests. This enables teams to work on shared code and increases the visibility into the development and quality of the system. By referring to a developer practice Continuous Integration (CI) typically expects developers to implement Test-driven development (TDD) with constant refactoring practice. When a developer is unit-test-driving his code, he ensures that his local copy is always working.

Applying the Scrum/LeSS methodology as described above in the context of CI the development phase through which a verified prototype is delivered in Scrum is called **Sprint**. The sprint refers to a development effort that is restricted to a specific duration, which is fixed in advance for each sprint and is more or less equivalent to the definition of similar development events like hackathons etc. The time allocated to the Sprint should be kept as is, while the scope of what will need to be done should be adjusted. No changes can be made during the Sprint that will affect the goal that has been set out for the specific sprint. Any changes that cannot go into the current sprint will enter the Backlog and prioritized for the following sprints. In AFarCloud the periodic sprint events will be aligned to the yearly demonstration phases as described in section 3.3, 6 sprint events shall be scheduled (3 for the first year since not all technical development tasks start at M1, so that all tasks will have started delivering to the product backlog) before each intermediate release to be used in the project demonstrators with

a time interval between them every 1 to 2 months. The processes involved with sprint execution are the following:

- **Sprint planning** occurring at the beginning of each Sprint. According to the LeSS framework the Scrum definition of Sprint Planning is named **Sprint Planning Two**, which is a separate meeting per team whereas a second level is added, named **Sprint Planning One**, which is a meeting for all of the Teams together (potentially via representatives) where they decide which team will work on which items.
- **Sprint backlog** definition is the list of work the development team must address during the next sprint. The list is derived by the scrum team progressively selecting product backlog items in priority order from the top of the product backlog until they feel they have enough work to fill the sprint.
- **Sprint Review** held at the end of each sprint to review what the team has done during the sprint and adjust the product backlog if necessary.
- **Sprint Retrospective** held at the end of each Sprint and before the next one. Its purpose is to inspect how the team performed and create a plan for improvements for the next sprint. Usually the team discusses on what went well, what went wrong and what could be done to improve it.

AFarCloud will implement a **sprint-based** verification and validation i.e. the top-priority features identified in the product backlog, which are drawn from the AFarCloud functionality lists and translated into technical specifications, will be evaluated on a sprint-based timeframe.

- Each leader of a technical feature, which could involve multiple partners, will be responsible of evaluating the feature. The reporting of the validation will be done in validation forms which will contain all the technical features of the product backlog with an understandable and measurable characterization, such as: *feature not yet addressed, not working, partially working, fully working*. In case of a partially working feature, some analysis will need to be provided to explain what is lacking. For fully working features and to facilitate the validation process, WP7 partners strongly recommend technical partners to proceed to unitary testing on a continuous integration system basis to verify continuously and automatically that new developments did not break the previously working features. Moreover, each fully or partially implemented feature will be documented, within the validation forms, indicating the required steps to use the feature. The validation forms will be shared with WP7 leader to centralize and track the validation progress.
- The sprint-wise verification and validation strategy requires a software release of the WP results to be issued regularly (not necessarily in the end of every sprint) on the AFarCloud repository. This accessible release will enable complementary evaluations performed by WP7 partners using the same and shared validation forms. These forms will enable to track the validation progress where it is expected to see a list of features characterized as “not addressed” features in the beginning of the project to a list of “fully working” features (at least for) the highest priority features. The validation achieved by WP7 partners, containing details and analysis of needed complementary work features, will be shared with technical WP leaders to assist them completing the call into question features.

3.1.2. Test Levels

According to the International Software Testing Qualifications Board's (ISTQB's) Agile Test Extension [4] the following test levels can be defined:

Component testing (also known as unit, module or program testing) searches for defects in, and verifies the function of, software modules programs, objects, classes, etc., that are separately testable. It may be done in isolation from the rest of the system, depending of the context of the development life cycle and the system. Stubs, drivers and simulators may be used. In the context of AFarCloud platform development separate component tests will be planned and executed in each technical Work package delivering components of the AFarCloud product backlog. Such tests will facilitate the verification at component level (unit-test).

Integration testing tests interfaces between components, interactions with different parts of a system and interfaces between systems. Systematic integration strategies may be based on system architecture (such as top-down and bottom-up), functional tasks, transaction processing sequences or some other aspect of the system or components. In order to ease fault isolation and detect defects early, integration should normally be incremental rather than "big bang". In AFarCloud integration testing should be scheduled within sprint events.

System testing is concerned with the behaviour of a whole product. In system testing, the test environment should correspond to the final target or production environment as much as possible in order to minimize the risk of environment-specific failures not being found in testing. System testing may include tests based on risks and/or on requirements specifications, business processes, use cases, or other high-level text descriptions or models of system behaviour, interactions with the operating system, and system resources. In AFarCloud this level of testing should be scheduled during product releases and is expected to be facilitated by the AFarCloud demonstrators.

Acceptance testing aims to establish confidence in the system, parts of the system or specific non-functional characteristics of the system. It is often the responsibility of the customers or users of a system; other stakeholders may be involved as well. Finding defects is not the main focus in acceptance testing. Acceptance testing may assess the system's readiness for deployment and use, although it is not necessarily the final level of testing. In AFarCloud this level of testing is not foreseen, since commercialization of the AFarCloud platform is not expected within the project timeframe and only final product validation during AFarCloud demonstrators is planned.

3.2. Product backlog, development and integration

The AFarCloud Product Backlog contains the list of features and tasks and will be the point of reference for the Sprint iteration planning. The backlog, as already explained will be mapped to application scenarios, technologies and components to a level that specific application scenarios can be reasonably assigned for the component. The items of the backlog are assigned to the partner(s) responsible of completing the component. A tree structure is also used in some cases in order to show dependencies with other tasks and help identify possible bottlenecks in the planning. An initial list of functionalities is provided in Table 2 below.

ID	FUNCTIONALITY DESCRIPTION	Goal
F1	Monitor environment: temperature (ambient, and the plant), wind, and weather forecast.	G1: Frost Effect/impact Reduction
F2	DSS for deciding about if it will be frost or not.	
F3	Detection of cereals nutrients composition (energy, protein and humidity analysis)	G2: Improve harvesting
F4	Using DSS take decision regarding when and where to harvest	
F5	Monitor NKP (sensors or imagery)	G3: Optimize the amount of fertilization
F6	Measure the needs of fertilization with high spatial precision	
F7	DDS for decision about when to fertilize	
F8	Outdoor livestock location tracking	G5: Improving the quality and the productivity with respect to the animals' well-fare, and meat/milk quality
F9	Detection of livestock heat	
F10	Detection of livestock calving	
F11	Detection of livestock rumination and eating	
F12	Determination of livestock growth rate	
F13	Inference of the livestock habits patterns for health and reproduction	
F14	Measure field water content/vigour	G6: Reducing water waste and cost in horticulture
F15	Measure water stress	
F16	DSS for decision about how much water	
F17	Automatic actuation on rooftop (open,close)	G7: Reducing water waste and cost in greenhouse
F18	monitor greenhouse temperature and humidity	

F19	Using actuators, irrigate with correct amount and location	G6: Reducing water waste and cost in horticulture
		G7: Reducing water waste and cost in greenhouse
F20	Detect plant illness (imaginary near infrared)	G8: Plant disease detection
F21	Monitor Gases	G5: Improving the quality and the productivity with respect to the animals' well-fare, and meat/milk quality
F22	NIR silage analysis	G4: Achieve the best nutrition components for feeding
F23	Livestock indoor positioning	G5: Improving the quality and the productivity with respect to the animals' well-fare, and meat/milk quality
F24	Livestock identification	
F25	Nutrition monitoring through rumen scanning	
F26	Extract and analyze data from milky robots	
F27	Livestock digestion monitoring	
F28	Fleet management: tracking of farm vehicles	G2: Improve harvesting

Table 2: Table of product backlog as defined in D7.1

As described above to deliver the complete product backlog the AFarCloud teams must implement the AFarCloud architecture as defined in WP2 (currently defined as shown in Figure 3) and develop and integrate the respective components that will be designed and delivered by the AFarCloud technical work packages.

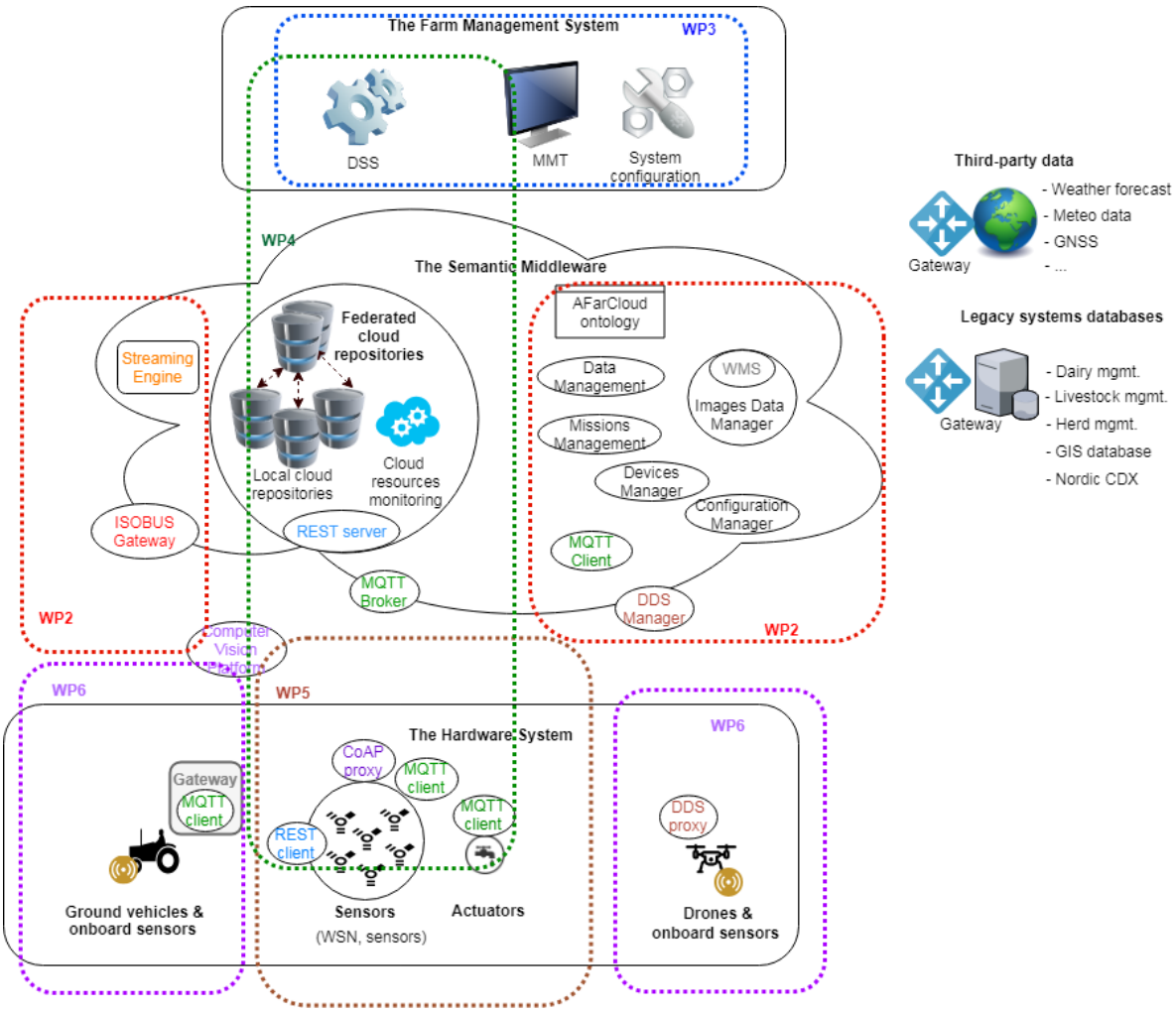

Figure 3. AFarCloud architecture

As described in section 2.1, AFarCloud will implement this architecture towards product realization. Thus, the product backlog will be complemented by the list of component definitions and integration maps that include component interfaces, communication services, data flows and the demonstrator plans that will determine the sprint planning as will described in more detail in the following sections. Thereinafter, Integration maps will be enhanced by APIs descriptions and by the respective data models to be used. For that reason, a API description template has been created and presented in section 8.1. Table 3, below, shows the current list of components planned to be delivered and integrated towards delivering the AFarCloud platform in its final form. A yearly plan for delivering AFarCloud components that address specific functionalities will be defined and following an evolutionary approach. Table 5 shows the format of an integration map that will be used to define the component integration plan to be used during integration phases to deliver operational systems to the AFarCloud local and holistic demonstrators.

WP2 - DDS Semantic Middleware	**WP3 - Farm Management System**	**WP4 - Environment characterization**	**WP5 - Sensors & actuators**	**WP6**
Cloud Repositories	Mission Management Tool	Gateway Algorithms for pre-processing	**COMMUNICATION / GWs**	UAS/DRONE
AFarCloud Data Model	Decision Support System (T3.3)	Data fusion server	**SENSORS, CAMERAS & ACTUATORS**	AGV
Data Access Manager	MMT - *Hierarchical planning*	Knowledge extraction (crops)	Soil sensors	TRACTOR
Data Processing and Fusion	MMT - *mobile MMT GUI*	Knowledge extraction (livestock)	Environmental sensors: Tº, wind, humidity, etc	Human Intervention SW
Cyber-security Management	Report Generator (WP3)	Cloud Resources Monitoring	Gas sensors	
Semantic Query	System Configuration	Footprint calculation proof of concept	NIR	
Streaming Engine	Cyber Security Management	MQTT Broker	IR camera	
Missions Manager		MQTT Client	Multispectral	
Configuration Manager		REST Server	TOF Camera	
Images Data Manager		Cloud Resources Monitoring	Visible Camera	
Missions Reporter			Collars 4 cows	
Environment Reporter			**Actuators**	
Computer Vision Platform			NTP device/ actuator for air treatment in greenhouse/ indoor environment	
DDS manager			**HIGH-LEVEL SOFTWARE**	
ISOBUS Gateway			Image Processing software (Computer Vision Platform	
Events Reporter				
DDS proxy				
Lorawan Gateway				
Lorawan Server				

Table 3. AfarCloud initial list of components

Following the analysis in T7.1 the mapping of the above components to the AFarCloud local and holistic demonstrators in order to fulfil the target functionalities during the first year of the project is shown in Table 4 below.

Demonstrators Involved	AFarCloud Component		
	Name	WP/ Task	ID/ Version
AS01	Data Access Manager	2	
	Semantic Query	2	
	MQTT Broker	4	
	MQTT Client	4	
	REST Server	4	
	Cloud Resources Monitoring	4	
	Gateway	5	
	Environmental sensors: Tº, wind, humidity, etc	5	
	NIR	5	
	Visible Camera	5	
	UAS/DRONE (IMCS)	6	
AS02	Images Data Manager	2	
	MQTT Broker	4	
	MQTT Client	4	
	REST Server	4	
	Cloud Resources Monitoring	4	
	Gateway	5	
	Environmental sensors: Tº, wind, humidity, etc	5	
	NIR	5	
	Visible Camera	5	
	UAS/DRONE (IMCS)	6	
AS03	Data Access Manager	2	
	Semantic Query	2	
	Environment Reporter	2	
	Data Processing and Fusion	2	
	Images Data Manager	2	
	MMT	3	
	MMT Hierarchical Planning	3	
	MQTT Broker	4	
	MQTT Client	4	
	REST Server	4	
	Cloud Resources Monitoring	4	

	Gateway	4	
	Footprint calculation proof of concept	4	
	Data fusion server	4	
	COMMUNICATION/ GWs	5	
	Soil sensors	5	
	Multispectral	5	
	Collars 4 cows	5	
	UAS/DRONE	6	
	Human Intervention SW	6	
AS04	Data Access Manager	2	
	Semantic Query	2	
	Lorawan Gateway	2	
	Lorawan Server	2	
	Environment Processing	2	
	Environment Reporter	2	
	Computer Vision Platform	2	
	MMT	3	
	MMT Hierarchical Planning	3	
	MQTT Broker	4	
	MQTT Client	4	
	REST Server	4	
	Cloud Resources Monitoring	4	
	Footprint calculation proof of concept	4	
	Data fusion server	4	
	Knowledge extraction (crops)	4	
	Gateway	4	
	COMMUNICATION/ GWs	5	
	Soil sensors	5	
	NIR	5	
	IR camera	5	
	Multispectral	5	
	Visible Camera	5	
	Computer Vision Platform	5	
	UAS/DRONE	6	
AS05	Data Access Manager	2	
	Semantic Query	2	
	Lorawan Gateway	2	
	Lorawan Server	2	
	Environment Processing	2	
	Environment Reporter	2	
	Computer Vision Platform	2	
	MMT	3	
	MQTT Broker	4	
	MQTT Client	4	

	REST Server	4	
	Cloud Resources Monitoring	4	
	Data fusion server	4	
	Gateway	4	
	COMMUNICATION/ GWs	5	
	Soil sensors	5	
	NTP device/ actuator for air treatment in greenhouse/ indoor environment	5	
	UAS/DRONE	6	
	TRACTOR	6	
AS06	Data Access Manager	2	
	Semantic Query	2	
	Lorawan Gateway	2	
	Lorawan Server	2	
	Environment Processing	2	
	Environment Reporter	2	
	Computer Vision Platform	2	
	MQTT Broker	4	
	MQTT Client	4	
	REST Server	4	
	Cloud Resources Monitoring	4	
	Gateway	4	
	Collars 4 cows	5	
	COMMUNICATION/ GWs	5	
	UAS/DRONE	6	
AS07	Data Access Manager	2	
	Semantic Query	2	
	Knowledge Extraction	4	
	Environment Processing	2	
	Environment Reporter	2	
	MQTT Broker	4	
	REST Server	4	
	MQTT Client	4	
	Cloud Resources Monitoring	4	
	Gateway	4	
	Environmental sensors	5	
	Collars 4 cows	5	
	COMMUNICATION/ GWs	5	
AS08	Data Access Manager	2	
	Semantic Query	2	
	Lorawan Gateway	2	
	Lorawan Server	2	

	Environment Processing	2	
	Environment Reporter	2	
	DSS	3	
	MQTT Broker	4	
	MQTT Client	4	
	REST Server	4	
	Cloud Resources Monitoring	4	
	Gateway	4	
	COMMUNICATION/ GWs	5	
	GAS sensors	5	
AS09	Data Access Manager	2	
	Semantic Query	2	
	Environment Processing	2	
	Environment Reporter	2	
	MMT	3	
	MMT Hierarchical Planning	3	
	MMT mobile MMT GUI	3	
	DDS	3	
	Data Fusion Server	4	
	MQTT Broker	4	
	MQTT Client	4	
	REST Server	4	
	Cloud Resources Monitoring	4	
	Environmental effects	4	
	Gateway	4	
	Soil sensors	5	
	NIR	5	
	IR Camera	5	
	Multispectral	5	
	Visible Camera	5	
	Computer Vision Platform	5	
	TOF Camera	5	
	Collars 4 cows	5	
	COMMUNICATION/ GWs	5	
	AGV	6	
	Software	6	
	UAS/DRONE	6	
AS10	Data Access Manager	2	
	Semantic Query	2	

	Environment Processing	2	
	Environment Reporter	2	
	MQTT Broker	4	
	MQTT Client	4	
	REST Server	4	
	Cloud Resources Monitoring	4	
	Visible Camera	5	
	COMMUNICATION/ GWs	5	
AS11	MQTT Broker	4	
	MQTT Client	4	
	REST Server	4	
	Gateway	4	
	Data Fusion Server	4	
	Cloud Resources Monitoring	4	
	NTP device/ actuator	5	

Table 4. Mapping of AfarCloud components to demonstrators

Following the above mapping complete integration maps will be maintained throughout the integration phases of the project to track the integration effort and progress in the format shown in Table 5 below.

Demonstrator Functionalities	Demonstrators Involved	AFarCloud Component			Functionality provided by the component + Special requirements/Dependencies							Integration plan		Release
		Name	WP/ Task	ID/ Version	DESCRIPTION	INPUT (from comp. x, y,z)	IN Communication interface(s)/service(s)	OUTPUT (to comp. x, y,z)	OUT Communication interface(s)/service(s)	Other Dependencies	RESPONSIBLE Partner	START	DURATION	M12: 1

Table 5. Format of integration map

3.3. **Integration phases and time plan**

As described above the project will evolve through discrete phases that involve:

- Continuous Integration implementing DevOps procedures that will be described in the following sections
- Partial integration and verification events before demonstrator integration (sprint events)
- Validation of intermediate releases at demonstration sites

Thus, the overall planning of the above-mentioned phases includes the integration, verification and validation steps as described above. A graphical representation of the integration, verification and validation methodology is shown in Figure 4 below. The time plan for each phase will be defined well before each demonstration event and the integration results will be collected and reported in periodic reports.

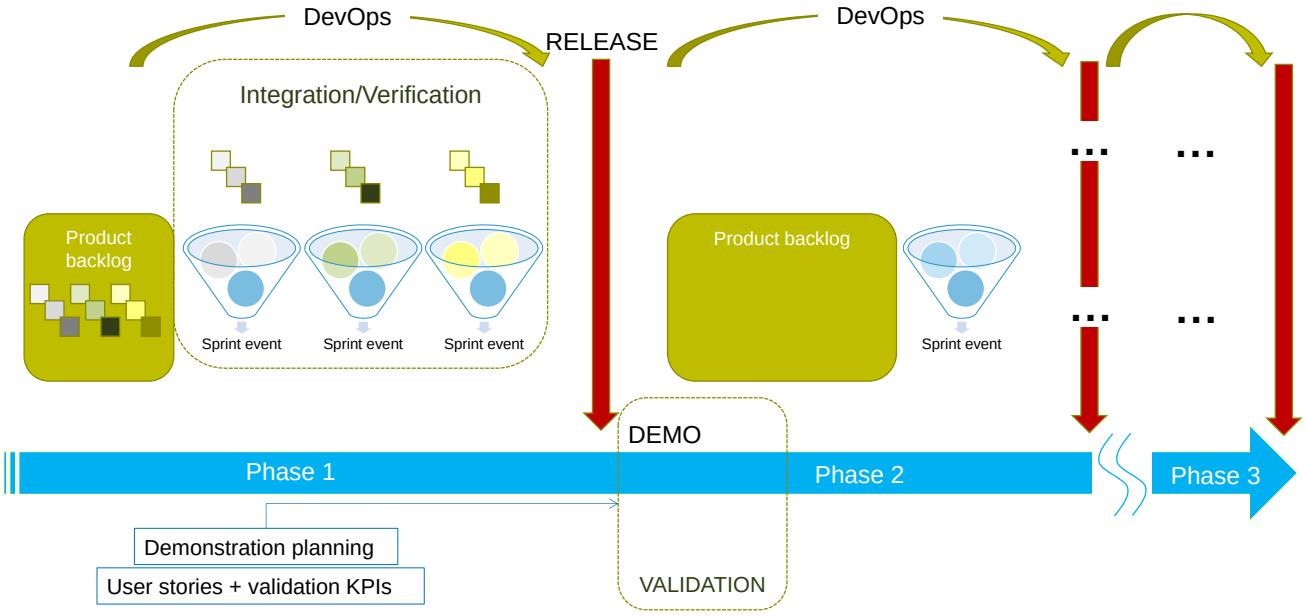

Figure 4. Overall integration and release plan

As shown in Figure 5 before each sprint and during the sprint planning event held at the beginning of each sprint, the sprint scope is defined by selecting which elements of the product backlog should enter the sprint's backlog and therefore be implemented during the current sprint. A sprint planning meeting will be held at the beginning of event in order to track the progress and solve any issues that might arise. At the end of the sprint, the sprint review and sprint retrospective event takes place in order to review the product increment as well as review how everything went and take actions for improving the process. After that meeting, the sprint planning event is repeated to plan the next sprint

until the required milestones are met and the product increment reaches the envisioned AFarCloud product.

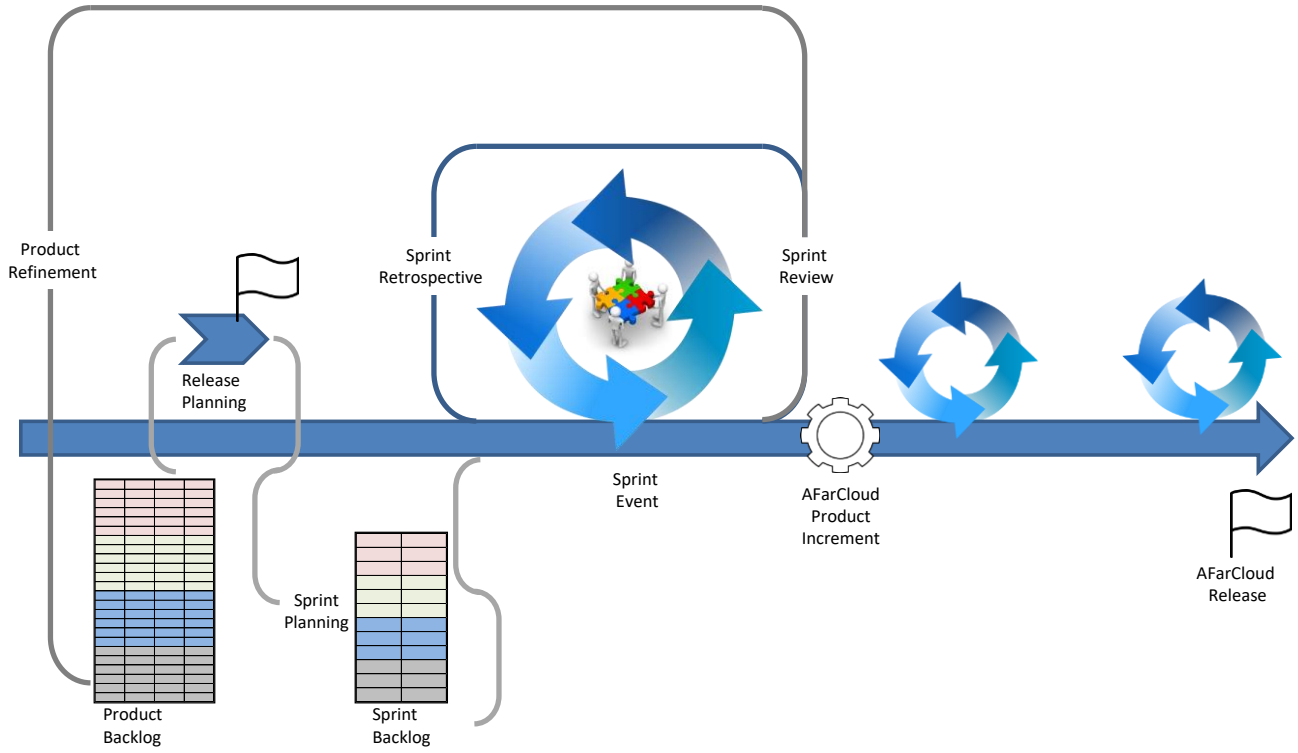

Figure 5. Sprint vs. release planning

3.3.1. Software components testing procedure

Testing activities of the software components are expected to be done before their deployment into the holistic scenarios. Consequently, a procedure has been established to make sure that the software components developed in the project are tested thoroughly, so that they can be deployed onto the vehicles and devices that are mounting them with the guarantee that they will not produce any kind of failure on them that will jeopardize those devices in any way. The procedure that has been created works as follows:

1. Software components **are first tested in the local premises and with the local equipment of the partner that is developing them**. In order to know how a software component to be tested works with the ones supposed to interact with, **the developers of the other software components must provide a description of how their own components are to be interfaced** by the developed one, so there is a clear idea on how to interact with them.
2. When those components have fulfilled that stage, they will be moved to an **abstract laboratory**, where a generic iteration of what a demonstrator will look like will be available. This generic iteration will consist of: a) the Farm Management System, where all the high level parts used for decision making, user interfaces or applications are contained as any other component (DSS, MMT, etc.), b) the cloud infrastructure, composed by the Semantic Middleware components and the cloud repository infrastructure and c) a simulation of the most widespread hardware components that are going to be used in the project (WSN, collars, other sensors, UGVs, UAVs). This simulation can be based on the software components used on those devices running in several computers, or in the vehicles themselves if it is possible.
   Management of this abstract laboratory and its testing activities will be closely linked to the partner where they are located. Access to software and debugging can be done remotely when required via programs like TeamViewer.
3. When testing activities are deemed as satisfactory in the abstract laboratory are deemed satisfactory, the tested software will be moved to **partial demonstrators** where its performance and overall behaviour can be assessed before they are moved to the holistic, final demonstrator. This will be the opportunity to install the component in the actual hardware device that is going to use it in the holistic demonstrator used for the project evaluation. There might be more than one partial demonstrator in use, in case it is required to test how information is shared among several demonstrators.
4. Finally, all the components will be moved to the **Holistic demonstrator**. The last debugging works and simulations will be done there.

The latter stages have been graphically summarized in Figure 6. Note that for the first integration stage, components involved may vary greatly depending on what each partner is working on.

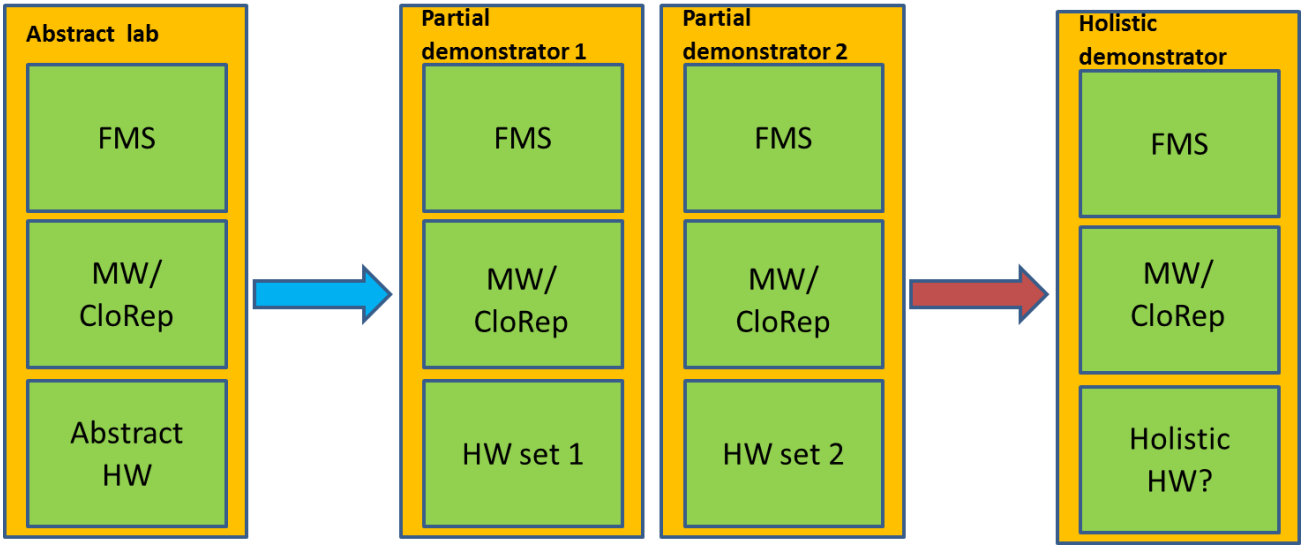

Figure 6. Deployment stages for testing

4. Validation strategy

4.1. Objectives and overall approach

In AFarCloud validation is intended to ensure that the final AFarCloud platform delivered during each planned release meets the operational needs of the AFarCloud users. A **release-based** validation will be performed according to the Release Plan described in the following section.

- *Storyboard based validation.* The release-based validation will be performed by WP7 partners assisted by technical WP leaders. A specific release validation form, containing all the user requested functionalities, will be used to validate the AFarCloud platform during the holistic demonstration phases. The release validation will produce a list of features not implemented, partially implemented or fully implemented related to the components verified as fully working listed in the product backlog and integration maps. The list of the incomplete features and an analysis of the issue(s) will be presented to the technical partners to assist them solving the problem(s).
- Documentation should be done per feature validated. The documentation per feature will be collected and reviewed by demonstrator owners, which will follow a documentation template. The missing documentation will be reported to the WP leaders of technical feature. The documentation provided to WP leaders will serve as a basis for the validation process. After each release-based validation, the validated documentation will be incorporated into platform release package.

- Validate the Key Performance Indicators (KPI's). Each release will present a list of KPI's that will be validated using measurable characterization, such as: not reached, partially reached, fully reached. A KPI validation report will be shared with technical partners to assist them reaching the goal. A first list of KPIs is available in subsection 4.1.1.

4.1.1. Key Performance Indicators

A specific strategy is followed in the AFarCloud project in order to monitor the project progress and the impact. Three levels of KPI were identified in the DoW:

- KPIs "level 1" is related to specific demonstrators.
- KPIs "level 2" is related to the AFarCloud overall platform.
- KPIs "level 3" measures the contribution for exploitation and the general innovation in modern agriculture, a novel ecosystem.

Level 1 KPIs are mentioned in D.7.1 and are mainly Business KPIs for the demonstrators' planning.

KPI's, related to each release of the AFarCloud platform, are defined by the WP7 partners a) based on the contents of the Description of Work document as well as on the demonstration plans provided by the rest of the Tasks in Work package 7 mainly D.7.1. These KPIs are "Level 2" KPIs or Technical.

Table 6: AFarCloud Technical KPIs

KPI ID	Technical KPI	Description
T1	Capacity	Capacity is the size of the workload compared to available infrastructure.
T2	Periodicity	The frequency of demand and supply activity The amplitude of the demand and supply activity
T3	Reliability	Mean Time Between Failures (MTBF) and Mean Time To Repair (MTTR). For non-repairable components, it is expressed as Mean Time To Fail (MTTF).
T4	Response Time	Response Time gives a clear picture of the overall performance of the cloud. It is crucial, as it has an impact on application performance and availability.
T5	Scalability	Degree to which the service or system can support a defined growth scenario.
T6	Security	The level of security clearance required to access service/data. (We should consider open data)

T7	Service/System Availability	This metric is the percentage of time that a service or system is available. It is the ratio of time a system or component is functional to the total time it is required or expected to function. An uptime of 99.9% means 42 minutes of downtime per month during which you cannot provide service to your customers.
T8	Throughput	The latency of transactions The volume per unit of time throughput An indicator of the workload efficiency
T9	Service and Helpdesk	Level of support. Need of a service level agreement
T10	Cost per customer	Estimation per year

Table 7 shows the mapping among Level 1 and 2 KPIs applied in each scenario.

Table 7: Level 1 and 2 KPI mapping

Scenario	Business (Level 1 KPI)	Technical (Level 2 KPI)
AS01: Demonstrator for cranberry protection	KPI1; KPI2	T1; T3; T4; T5; T10
AS02: Silage / Cereal monitoring and control	KPI2; KPI4; KPI10	T1; T4; T5;
AS03: Environmental monitoring for sustainable crop production and livestock welfare"	KPI2; KPI4; KPI6; KPI7	T1; T3; T4; T5; T7
AS04: Vineyards monitoring	KPI1; KPI5	T2; T3; T4; T5; T7; T10
AS05: Farming based on permaculture principles	KPI2; KPI3; KPI4; KPI5	T1; T2; T3; T4; T5; T7
AS06: Livestock health and movement	KPI7; KPI8; KPI9; KPI10	T1; T3; T5; T7
AS07: Measurement of health status through ruminal probes On Pasture Monitoring	KPI7; KPI8	T1; T3; T5; T7

AS08: Measurement of health status through dairy robotics and gas monitoring"	KPI7	T1; T3; T5; T7
AS09: Cow nutrition management	KPI1; KPI2; KPI10	T1; T3; T5; T7; T10
AS10: Sustainable livestock farming	KPI1; KPI2; KPI10	T1; T3; T5; T7; T10
AS11: San Rossore Park	KP1, KPI3, KPI4, KPI5, KP6, KPI10	T1; T3; T5; T7; T10

4.1.2. Security Assessment Process

The cybersecurity assessment reflects the actual achived security level (SL_A) of the Austrian – Use Case (AT-UC) implementation. For detected security gaps, recommendations of security countermeasures are given how to fulfil the necessary security levels.
In general, the AFarCloud architecture specification, defined in T2.2, will be assessed for cybersecurity properties. For example, to identify both the security assets and the security weak points of the overall design. The assessment is done in a general way to act as a base for security assessments for a dedicated demonstrator architecture.
In detail, the AT-UC will be investigated for a concrete cyber security assessment. In the first-year only a first and conceptual security analysis report will be worked out, based on the system description of the AT-UC.
The security assessment process will be done according the security standard IEC62443.

4.1.2.1. Cybersecurity Assessment Process Flow

Figure 7 shows the cybersecurity process flow predetermined by the security standard IEC62443.
The standard descripts the main steps to perform the cybersecurity assessment of a given system.

1. The security assessment starts with a detailed description of the System under Consideration (SuC). This step defines what shall be assessed and what are the system borders. This process step must be done very carefully, because an extended system definition would produce unnecessary analysis effort on the one-hand side. On the other hand-side, a narrowly limited system definition will overlook important system parts which finally are never included in the overall security analysis. In general, in the AT-UC the SuC includes all components beginning with the sensors on the field and ends with functionalities in the cloud.

2. The detailed system description is the input for the high-level security analysis. Methods like FMVEA (Failure Mode and Vulnerability Effect Analysis) and TARA (Threat Analysis and Risk Assessment) are used to identify all possible security attack vectors in the given operation environment.
   The main assessment criteria are the following protection consideration to evaluate the necessary security level SL0 to SL4

SL 0	No specific requirements or security protection necessary

SL 1	Protection against casual or coincidental violation
SL 2	Protection against intentional violation using simple means with low resources, generic skills, and low motivation
SL 3	Protection against intentional violation using sophisticated means with moderate resources, IACS specific skills, and moderate motivation
SL 4	Protection against intentional violation using sophisticated means with extended resources, IT specific skills, and high motivation

A Security level (SL) will be expressed as a security vector. The vector contains seven SL numbers (0-4). Each number of the vector represents the selected security level number for the appropriate Foundational Requirement (FR).

FR1 – IAC	Identification and Authentication Control
FR2 – UC	Use Control
FR3 – SI	System Integrity
FR4 – DC	Data Confidentiality
FR5 – RDF	Restricted Data Flow
FR6 – TRE	Timely Response to Events
FR7 – RA	Resource Availability

3. Not all parts of the system must have the same security level. In this case the SuC is divided in dedicated security zones and conduits, for a detailed analysis. The conduits define the data communication paths between the diverse zones.

4. *(→ Will be done after Year 1*) The analysis output shows a positive assessment result when the identified security risk can be tolerated by the asset owner. When not, a detailed cybersecurity analysis for any zone and any conduit must be performed to identify the necessary cybersecurity counter measures to harden the system.

5. *(→ Will be done after Year 1*) Finally, all new found security recommendations and new requirements are documented as part of the CyberSecurity Requirement Specification (CSRS). This report reflects only the current security status.

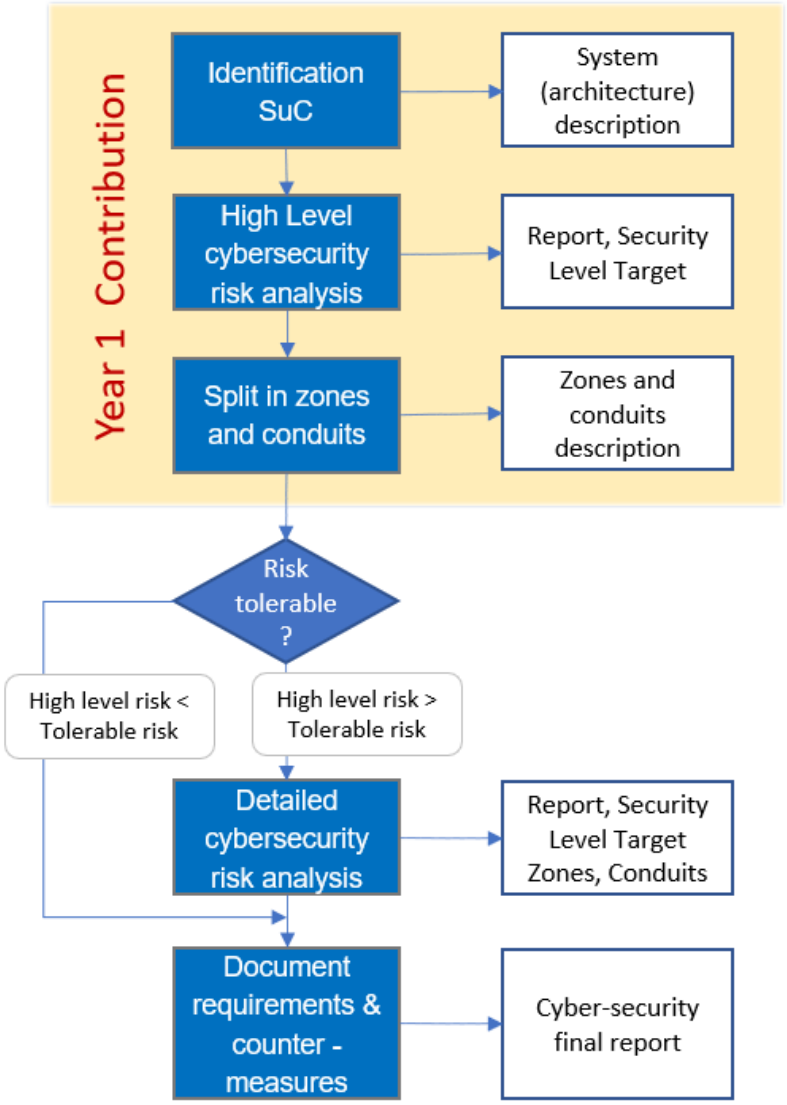

Figure 7 Security assessment process diagram (IEC62443)

4.1.2.2. Asset Owner Approval

The asset owner (a person who contribute or use the system – e.g.: The security expert for the farm plant) reviews the results of the security assessment. The security assessment expresses the security capability of the system with the SL_C (Security Level Capability). In general, the SL_C documents what the zone or the conduit can maintain with the actually implemented security measures. The SL-C must be equal or better than the SL-T (Security Level Target). The comparison of SL-T and SL-C can be expressed in a diagram like the following in Figure 8. This diagram is generated for each zone and conduit, individually.

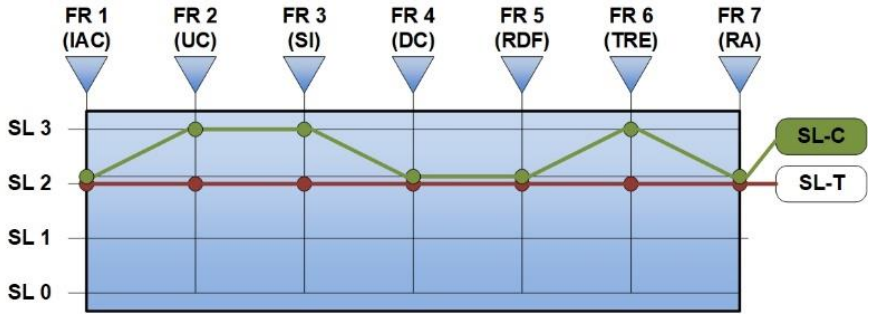

Figure 8 SL_C to SL_T comparison overview

4.1.2.3. CyberSecurity Requirement Specification (CSRS)

The resulting CSRS defines the implementation of the given security requirements and mandatory security counter measurements for the analyzed system (SuC), when a specific security level shall be achieved.
A CSRS shall contain the following additional parts for ~~a~~ complete documentation:

- SuC description
- Operating environment assumptions
- Threat landscape
- Mandatory security functions
- Tolerable risk
- Regulatory requirements

A security assessment and analysis must be done periodically to react to new cybersecurity threats early in time with an appropriate counter measure.

4.1.2.4. Security Assessment Data

Data	**Presentation concept**
Overall system definition	The system description is essential for a usefully and complete definition of the System under Consideration (SuC)
Operating environment assumptions, threat landscape	The operation environment defines the borders of the analysis and descripts unusable and unrealistic considerations or assumptions to keep the analysis within a limit.
Cybersecurity attack vectors	The possible cybersecurity attack vectors are identified from the operation environment and by a FMVEA and TARA analysis.
Required security level	A definition of the adequate security levels is essential for a usefully and complete cybersecurity assessment.

4.2. **Release plans**

The AFarCloud Release plan is made based on the evolution of the platform in three yearly stages originally foreseen in which feature development will be undertaken. Three demonstrators will be set up in a different country per year, in an incremental way, starting as early trials the first year and ending with a final demonstrator where the platform, functionalities and devices will be validated. Each release should demonstrate specific high-level features as defined during user requirements and storyboard collection phase.

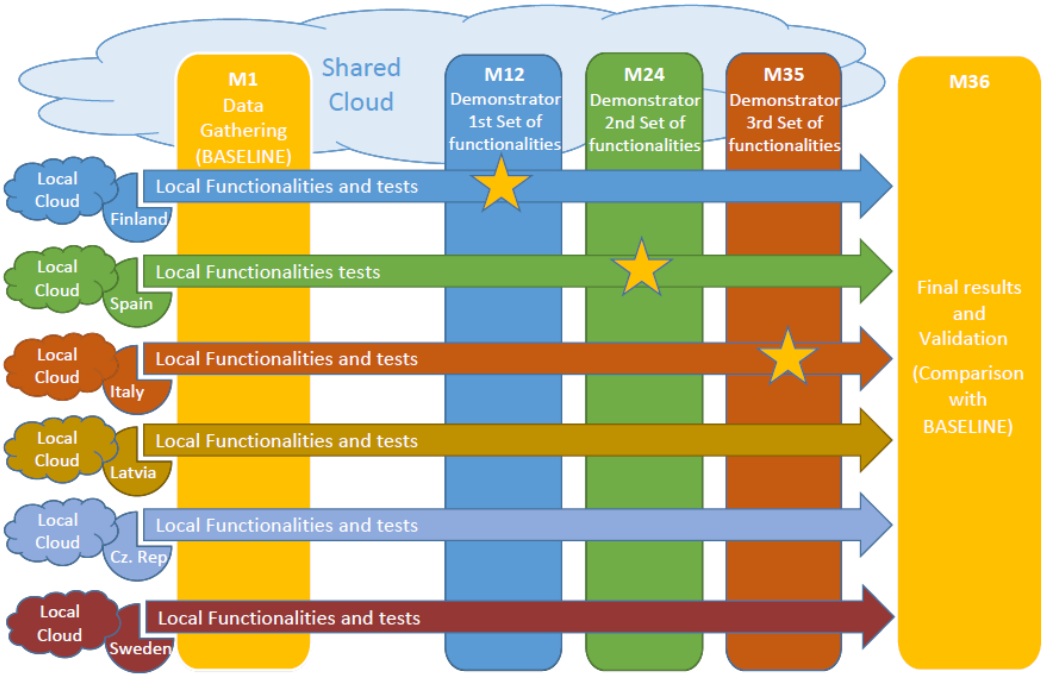

Figure 9. AFarCloud release phases

The AFarCloud demonstration strategy and planning delivered in D7.1 is gradual and incremental from two perspectives. First, all platform components are tested in local demonstrators before moving these components to the holistic demonstrator in a yearly basis. Second, demonstrator functionalities are implemented gradually through the lifecycle of the project. Each year, based on the results from the last years, the technologies supporting each functionality are improved and completed. Figure 10 shows the integration and validation planning of local demonstrators towards the holistic demonstrator. Each year a similar gannt will be executed. The holistic demonstrators will deploy all the functionalities that have been tested in local demonstrators and can be hosted by the holistic demonstration's farm.

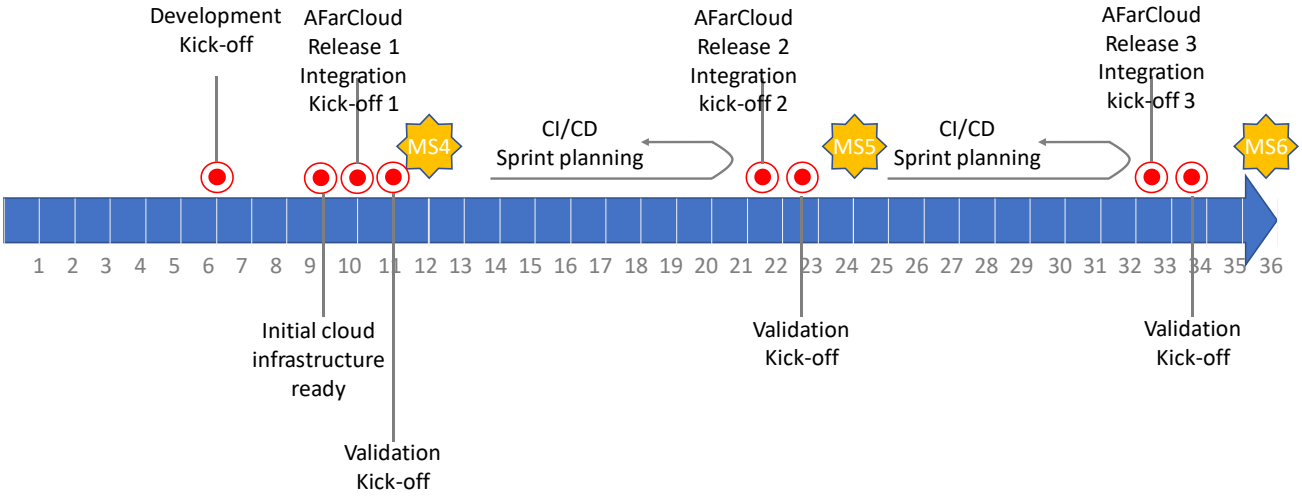

Figure 10. AFarCloud release planning

5. Technologies, tools and guidelines

5.1. Integration guidelines

Software and hardware development can be a very complex task. As a principle, it is highly recommended for AFarCloud developers to apply Integration Guidelines (IG) during the implementation of the required software and hardware components. AFarCloud developers, by following these IG, they would have positive effect on the code quality and ultimately, the integration testing of the developed product. IG defines coding standards to be followed, to ensure that the code is readable and understandable by different developers involved in the process.

5.1.1. Design Patterns

Design patterns have been derived as general reusable solutions to commonly occurring problems within a given context. They have been evolved by developers during a period of time, so that they describe best practices in common diagnosed problems. AFarCloud developers are encouraged to use design patterns, where applicable. Having a common standard terminology would significantly enhance communication quality among the partners.

5.1.2. Code comments and documentation

On the same context, code documentation is highly recommended within AFarCloud. Among the advantages one can be rewarded is the easier revision of developers' code in the future and at the same time minimize the required time for any source code updates. Thus, code comments and clear documentation benefits both the developer and the interested partners who wish to integrate with a specific component in AFarCloud.

5.1.2.1. OpenAPI specification

As the maintenance of API documents is getting a headache with the evolution of APIs functionality, online API documentation alleviates this burden. This is now achieved through the OpenAPI Specification (OAS).

OAS defines a standard, programming language-agnostic interface description for REST APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic. When properly defined via OpenAPI, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Similar to what interface descriptions have done for lower-level programming, OAS in calling a service. OpenAPI documents describe an API's services and are represented in either Yet Another Markup Language (YAML) or JSON formats. These documents may either be produced and served statically or be generated dynamically from an application.

In AFarCloud, developers are urged to adopt Swagger for online documentation of the developed APIs.

5.1.3. Programming languages and Software architecture

The applications and services that should be integrated and supported have vastly diverse specifications and, therefore, requirements, resulting in the selection of a single programming language as preferred to be an unrealistic consideration. AFarCloud has opted for orchestration at the level of APIs that are technology agnostic.

RESTful APIs and publish subscribe client-server architectures (e.g MQTT) have been considered across all the development activities of AFarCloud. By means of this approach, there is no programming language lock-in, allowing all interested parties to integrate by simply respecting the specifications of the APIs of interest.

REST relies on HTTP for the application of “Create”, “Read”, “Update” and “Delete” processes through the “POST”, “GET”, “PUT” and “DELETE” HTTP methods respectively. AFarCloud adopts REST for enacting synchronous communication among its components through common HTTP APIs and pub/sub messaging for asynchronous communications, enabling microservices architectures.

5.1.4. Interfaces and Data Models

Interfaces and Data Models are elements of major importance for a successful integration process among different components. Data models define classes, parameters and allowed methods whilst Interfaces define the communication channel between software components. Interfaces are specifying the information flow, the interacting entities and the way that this interaction is realized.

In AfarCloud where there is a significant number of partners and developers who would contribute in the development process of the platform, it is an absolute necessity for every contributor to provide the definition of Data Models and Interfaces for each component in a transparent and coherent manner, to the extent possible. Furthermore, this approach is even more prominent given the fact that likely a number of modifications over existing code might take place in the lifetime of AFarCloud project, thus a number of modifications will have to be made in the respective Interfaces and Data

Models. Therefore, in AFarCloud, given the great importance that descriptions of interfaces and data models play in the Integration process, it is mandatory for the partners to use the API description template that is described in section 8.1. Data models' descriptions shall be illustrated in a tabulated format.

5.1.5. Unit Testing and Source commits

Source control appears as a greatly helpful solution for software development, especially for partners or developers situated in geographically dispersed places who co-implementing software components. Furthermore, Source control facilitates the integration process by communicating the progress of the implementation to the interested parties. A good practice that is highly encouraged is to code commit in a frequent, meaningful and systematic way. This would improve the reviewability of the updated code by the interested partners. However, its benefits may be burdened by improper use of the control elements offered by the source code management platform of the project.

Serious integration problems might get introduced when "broken" code is committed. Such problematic behaviour will stall and ultimately put in jeopardy the integration process. Therefore, in AFarCloud developers are required to thoroughly test their code before uploading it in the common repository.

Lastly, a constructive approach for a newly implemented software component, should be to provide software commits starting from basic API skeletons and gradually construct fully functional code blocks.

5.2. AFarCloud DevOps development environment and procedures

5.2.1. DevOps infrastructure set up

DevOps (Development and Operations) is a software development phrase.

AFarCloud software components will be developed by all the technology partners of the consortium. In the context of the project, each technology provider will set up its own development environments, but a shared software repository will be set up for the integration of all the components to be implemented so that they can interoperate, communicate and work together in an integrated manner. Finally, specific environments will be created for each pilot with the integration and customization of the components that it uses.

DevOps is a set of practices that automate the processes between software development and IT teams, in order that they can build, test, and release software faster and more reliably. The concept of DevOps is founded on building a culture of collaboration between teams that historically functioned in relative siloes. The promised benefits include increased trust, faster software releases, and ability to solve critical issues quickly, and better manage unplanned work [5].

In AFarCloud, DevOps philosophy and the corresponding approach will be used and applied internally for the development and the operation (deployment and validation into the pilot's environments) of the software components in the project.

This section aims to present the DevOps tools planned to be used at project level for the development and operation of AFarCloud's implementations.

AFarCloud platform is composed of a set of software components that will be implemented by different partners following different technologies. To overcome the integration challenges, AFarCloud will use a DevOps based approach to be able to fully support the management of these implementations and the planed releases. DevOps integrates development and operations into a single-minded entity [6] with common goals: high-quality software, faster releases and improved users' satisfaction.

DevOps also incorporates a number of agile principles, methods, and practices such as continuous delivery, continuous integration, and collaboration[7].

For a successful implementation of a DevOps approach, it is required a set-up of a development and delivery pipeline that consists of the stages an application goes through from development through production, as shown in the figure below. This figure shows the environments that are envisioned in AFarCloud covering the different development stages:

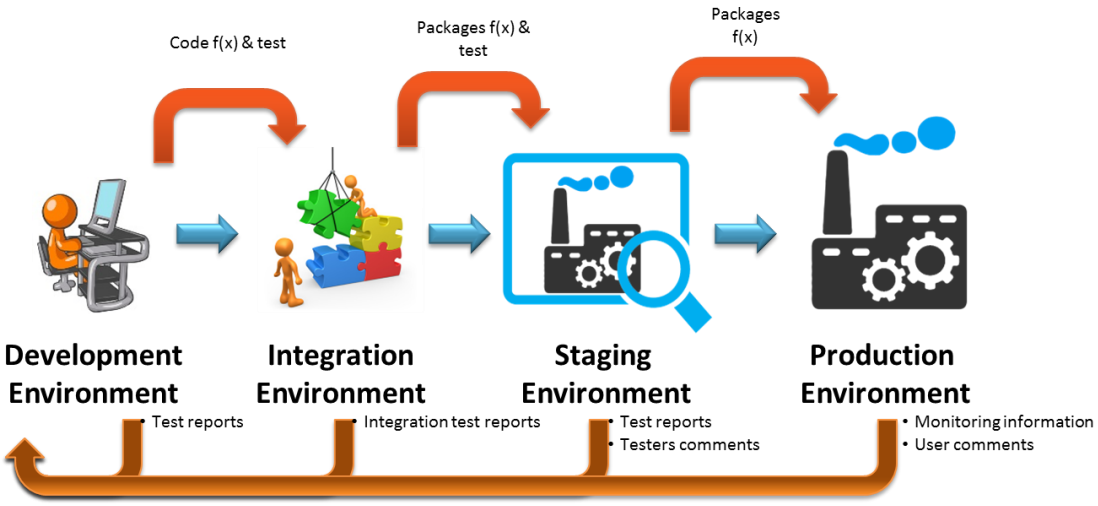

Figure 11. Envisioned stages for the development, integration and validation of the software components to be implemented in AFarCloud.

- The *Development* stage aims to provide a development environment where to write and test code, as well as to support collaborative environments (e.g., source control management, work-item management, collaboration, unit testing, project planning). Possible tools to cover these are: *Git* [8] as version control system, Jenkins *Jenkins* [9] to support continuous integration, *Apache Maven* [10] to manage project's builds, reporting and documentation, and containerization technology to have applications running in self-contained units that can be moved across platforms (e.g., Docker [11][11]). Tools such as GitLab also offer its continuous integration and continuous deployment pipeline.
- The *Integration* stage focuses mostly on compiling the code and performing the unit and integration tests.

- The *Staging* (pre-production) stage is where the Quality Assurance, user acceptance, and development/testing teams do the actual testing. Possible tools to support this stage are: *Jenkins*, *Apache Maven for* building and testing instructions, and *xUnit* as unit testing framework. Tools to automate the creation of the infrastructure such as *Chef* [12] (a cloud infrastructure framework that automates the building, deploying, and management of infrastructure) or *Puppet* [13] (for data centre orchestration by automating configuration and management of machines and software) are often used at this stage. The use of Containers technology [1] such as Docker or Kubernetes also play an important role to quickly deploy and port environments.
- The *Production Environment* focuses on the management and provisioning of the environment that the pilots and final users get to test. The tools mentioned above, such as Chef, Puppet, Docker or Kubernetes are used in this stage.

In the following sections the different tools (shown in figure 2) and how they are planned to be used in the context of AFarCloud are presented.

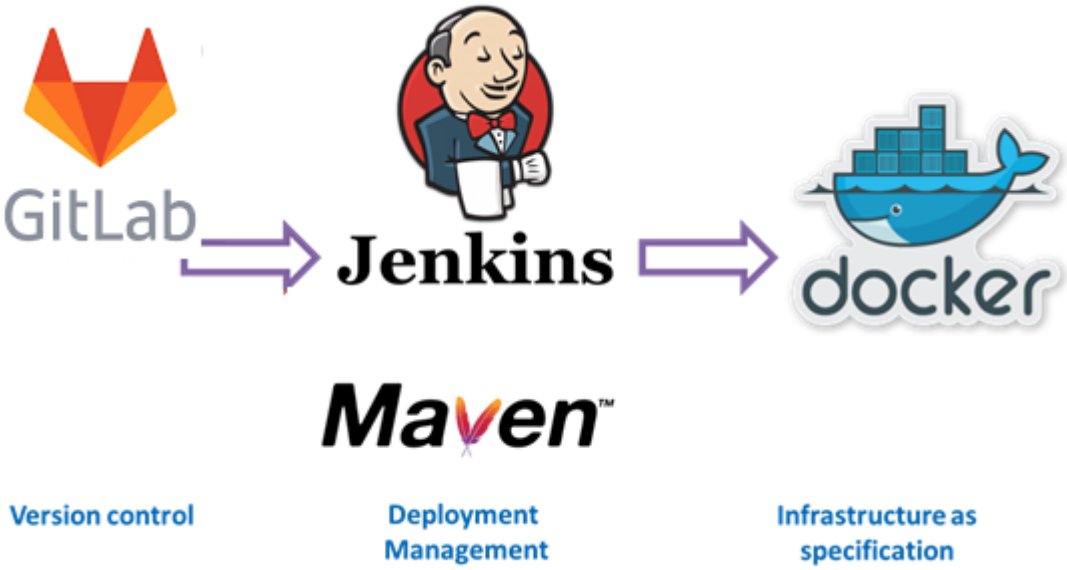

Figure 12. Tools for version controlling, deployment and infrastructure in AFarCloud.

AFarCloud is a considerably large project meaning that the AFarCloud platform consists of a large number of individual components. Arguably, there might be a case where some partner won't be able to comply to these CI/CD recommendations. It is acceptable to override these guidelines to an extent. However, integration guidelines, especially the ones that fall into Integration Testing category and refer to the testing objectives, testing reports etc. are mandatory and all AFarCloud partners are obliged to comply to these directives.

[1] A container is *"a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another"* (source https://www.docker.com/resources/what-container). This approach has the benefit of speeding up the testing process and building large, scalable cloud applications.

5.2.2. Version control and task management

5.2.2.1. Software repository

The technical work packages of AFarCloud will use GitLab [14] to manage source code and for version control. GitLab is an open source code management (SCM) system based on Git [8] but adding its own features covering for instance the DevOps pipeline.

The AFarCloud GitLab repository, hosted at TECNALIA, will store both private and public repositories. The private repositories will be used to host the initial stages of the different components of the project until they are mature enough to be deployed into the public domain. Besides, the private repositories will be also used to store repositories required by the pilots to develop their pilot-oriented specific source code and resources.

The different components of AFarCloud will become public, whenever appropriate, following the AFarCloud Description of Work (DoW) commitments. All public source code will only be made public once the licenses of the components have been agreed by all interested parties. Private repositories will host the partners' proprietary implementations as established in the different individual and collaborative dissemination strategies.

Gitlab, promotes conversational development to speed-up the coding activities, increase errors visibility and establish proper, controlled CI/CD operations. To achieve this, Gitlab exposes the conversational development status of a project using the following tabulated stages and calculated values to calculate the project's conversational development index (ConvDev Index), providing an overview of the project adoption of Concurrent DevOps from planning to monitoring.

Gitlab promotes the term "pipeline" to describe sets of sequential continuous integration (CI) and continuous delivery (CD) operations. In this course, CI pipelines include code building followed by automated unit and integration tests. Next, CD pipelines deploy the code to different environment, most usually for review, for actual user testing (staging phase) and, finally for production use. The above is depicted in Figure 13.

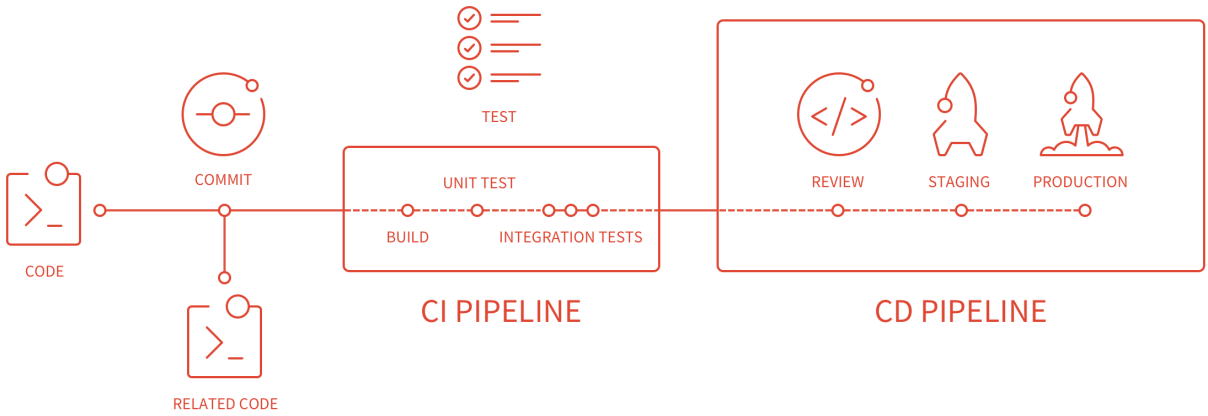

Figure 13: Continuous integration schema

5.2.2.1. GitLab Integration guidelines

Gitlab organizes the CI/CD pipelines by employing pipeline graphs as illustrated in the figure below. These are defined and described via sets of simple YAML scripts (with a static filename, like as: .gitlab-ci.yml) referred to as job files, where the various pipelines are organized in stages. Jobs are executed by designated Gitlab runners.

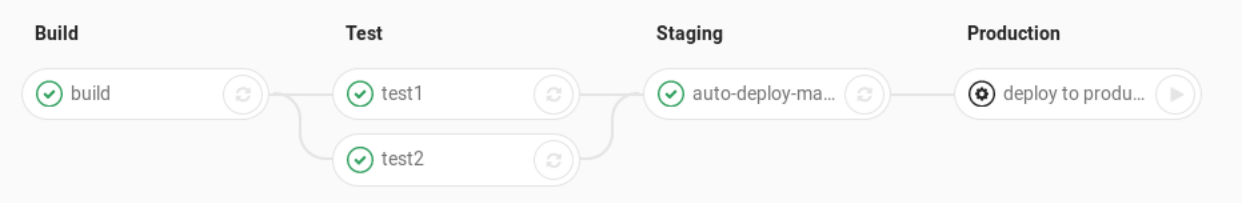

Figure 14: GitLab pipeline schema

It is worth mentioning, however, that the aforementioned Gitlab runners refer to services that connect to the project Gitlab instance -either the public or a private one- and listen for code changes so that they can actually perform the source code building/testing/deploying. The code execution is performed in sandbox Docker containers that are automatically generated by the Gitlab runners and get automatically deleted when all the stages have been successfully completed.

As already mentioned, GitLab has been selected as the CI/CD framework to host the project development. Some of the functionalities of the GitLab are tabulated below:

- Multiple projects are possible, grouped under groups and subgroups. So, the source code implemented the components belonging in separate architectural blocks can be organized in separate groups.
- Multiple private/public projects can be created, so during immature developments steps can be taken privately, while making available more stable versions of software as open source.
- GitLab provides significant flexibility in developing teams' collaboration. Different developing teams can work in parallel on different branches, developing, testing and evaluating their features at separate deployment environment without disturbing other teams. Then, code can be merged when features are mature enough.
- A single branch can be deployed at different environments, separating staging from production

5.2.2.2. Tracking development

An Issue Tracking System is a software package to maintain and manage a list of issues, usually within a collaborating team, until they are resolved. In software development environments, a problem or "issue" can be a feature, a bug or any other request desired to be tracked towards its resolution. Issue Tracking Systems usually support resource assignment, priority definition, time constraints, project planning, etc.

It is envisioned that the AFarCloud consortium will use a tool for managing the development project of the AFarCloud platform comprising the different software components. While it is not yet decided which tool could be used for this purpose, a good option could be the use of GitLab issues.

The issue functionality in GitLab has several goals such as proposing new features or functionalities, reporting bugs, or foster discussions among developers on a certain topic or idea.

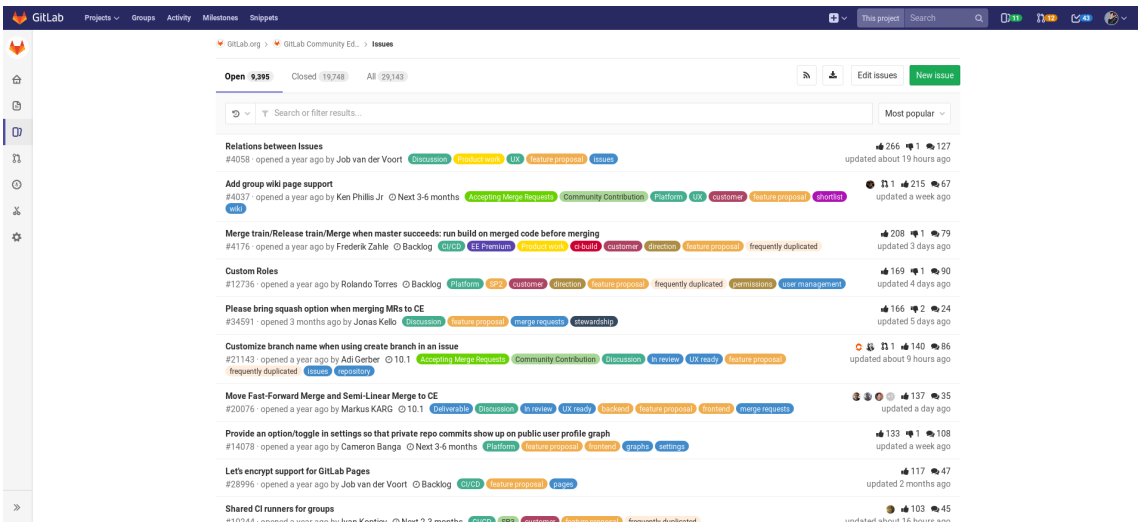

Figure 15. GitLab issues (adopted from GitLab https://docs.gitlab.com/ee/user/project/issues/)

A GitLab issue has at least the following fields to be completed:

- Issue description
- Assignee
- Milestone
- Due date
- Labels

5.2.2.2.1. Version release

The AFarCloud project release schedule will be as follows:

- Major version release at each milestone as originally planned in the AFarCloud Proposal DoA.
- Minor version release following the alpha/Beta integration strategy proposed in the following section

Versions are denoted using a standard triplet of integers: MAJOR.MINOR.INTERIM BUILD/PATCH (e.g. 1.2.51). In between Major or Minor version, partners can generate any number of interim releases.

5.2.3. Deployment management

In the context of AFarCloud one of the main objectives of the DevOps philosophy is to enhance the flow between the development stage and the operation stage, to decrease the production times. There are several ways of improvement such as agile methodologies and other approaches (i.e. Kaizen

[15], Lean [16], SixSigma [17]) but when the objective is to decrease the time of passing from development to production, one of the main resources is the systematic automation of repetitive tasks.

There exist multiple approaches for automating tasks. In AFarCloud, it is envisioned to use Jenkins.

5.2.3.1. Jenkins (automation server)

Jenkins, supports the execution of *post build tasks* to enable the *integration tests* of one or more artefact over a preconfigured environment such as a container. The usage of an automation server such as Jenkins, provides a lot of advantages when sharing the information about the status of the continuous integration tasks, both for the developers and for the users.

Main functionalities:

Jenkins provides many functionalities that can be extended through the more than 2000 already available plugins. In this section, only those relevant for AFarCloud are described to support namely the registration of a component, the continuous integration, debugging, modifying a component and deleting it.

During the continuous integration stage, the automation server should support different development strategies, such as the alfa-beta approach for integration complemented with manual testing. The functionalities that will be needed for such a development strategy are:

- Creation of the automatization tasks to systematize the DevOps cycle and to accelerate the incorporation of the changes and modifications into the production environment.
- Grouping the tasks into different groups to manage complex developments.
- Review the execution log so that mechanisms to identify the causes of a failure in the automation task are in place.
- See the status of previous executions to analyse failures.
- Keep the results of past executions to see the details of the past executions i.e. when a failure occurs.
- Notify the status of the executions for the automatic launching of some tasks.
- Recover the automation code from git, to include in configuration management, the integration tasks.
- Delete projects.

Apart from these functionalities, using Maven or previously installed plugins, Jenkins will be able to manage the infrastructure to execute the different components so that it can perform the integration tests against them.

Integration points:

With respect to the integration technologies, in AFarCloud it is envisioned to use mainly the REST API provided by Jenkins [18]. This API provides functionalities for:

- Recovering information from Jenkins
- Launching executions
- Creating/ Copying Jobs

5.2.3.2. Cloud infrastructure

A cloud infrastructure is set up to support the AFarCloud DevOps development methodology in the most automatic and efficient mode. Figure 16 depicts this infrastructure:

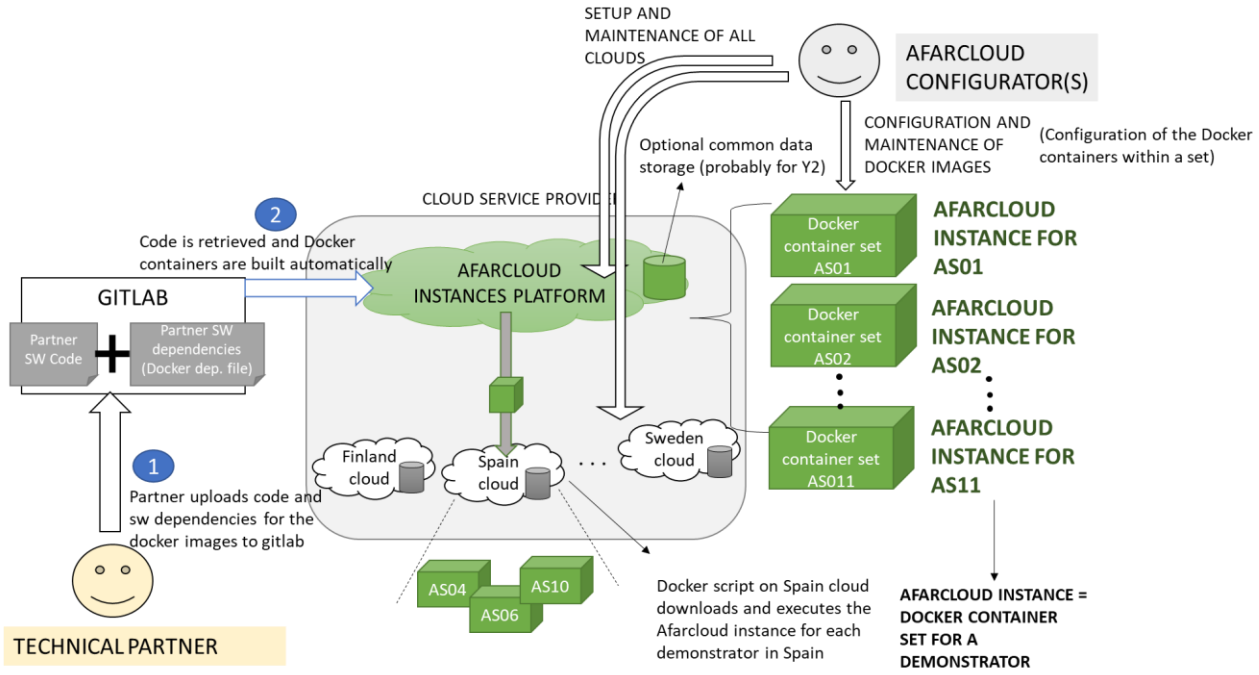

Figure 16 – AFarCloud infrastructure for deployment

In this infrastructure, the AFarCloud Instances Platform contains all the AFarCloud instances, that is, the platform instances for all demonstrators. An AFarCloud Platform instance is a set of Docker containers. Instances are automatically created by the AFarCloud Instances Platform that automatically retrieves the platform components from GitLab. Each instance is configured based on the needs and requirements of its demonstrator, which is established by a configurator partner in the Consortium. Each demonstrator is hosted in each own cloud and automatically downloads and runs its corresponding AFarCloud Instance from the AFarCloud Instances Platform. Demonstrator clouds may potentially be grouped in cloud accounts per country, in line with the AFarCloud release plan outlined in Figure 7. If possible, this design decision would make it easier to administrate demonstrator clouds based on each country's resources.

5.2.4. Infrastructure as code

The management of complex environments, such as the one foreseen in AfarCloud, where technical incompatibilities are easy to arise pose a huge risk, are error – prone and time consuming. This is so

because it may involve some of all the following activities: requesting and managing virtual machines, configuring the access to those machines, their operating system, access to the components, and so on.

To avoid these risk, AFarCloud proposes to use containers, whenever feasible. The containers technology allows the definition of separate spaces (both at communication level and at file system level) in the same virtual or physical machine, optimizing the computation resources.

At the same time, containers based technologies (such as Docker [11][11] or Warden[19]) allows the explicit provision of the configuration of the containers: baseline operating system, packages included, initial content, etc. This allows the instantiation of the same container with exactly the same initial characteristics.

Furthermore, some containers technologies support the usage of the containers registry where developed containers can be uploaded so that other team members can download and use /test them with a small set of instructions.

5.2.4.1. Containers

Containerization refers to Operating System (OS)-level virtualization for deploying and running distributed applications. Containerization is a lightweight alternative to a virtual machine that involves encapsulating an application in a container with its own operating system. A container takes its meaning from the logistics term, *packaging container*. When we refer to an application container, we mean packaging software.

In AFarCloud, the selected containerization tool is the Docker open source tool. Adopting Docker will allow AFarCloud to efficiently create virtual environments for integration tests and staging.

In AFarCloud it is envisioned to use Docker as containerization technology. These are the reasons for this election:

- Open source technology
- Professional support if required
- It provides a public registry for the containers or we can create our own one.
- It has an extensive and growing users' community.

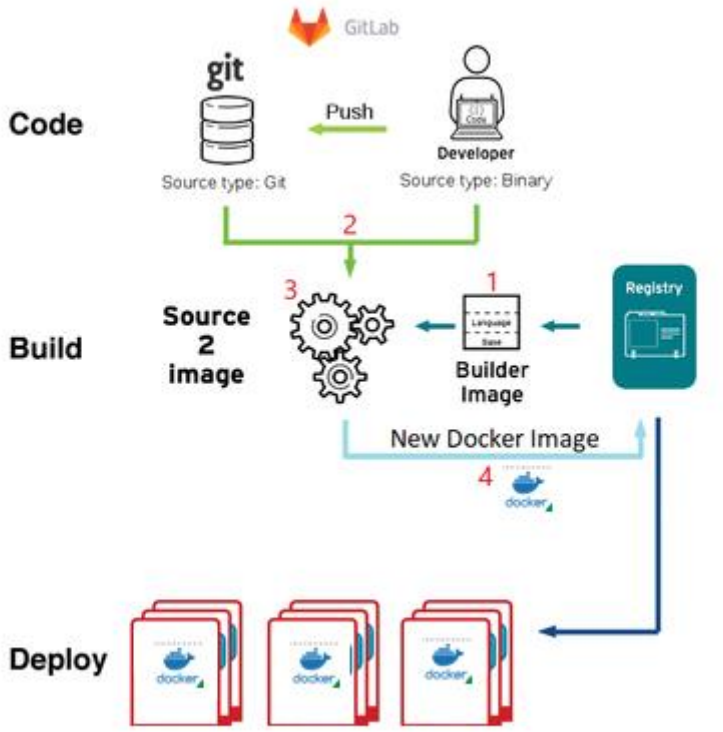

Figure 17: CI/CD schema

Main functionalities:

Docker provides a lot of functionalities. In this section, only those relevant for AFarCloud are going to be described. For this, we will focus on the following DevOps stages: continuous integration, publication, distribution and updating.

In AFarCloud it is envisioned to use the following capacities from Docker:

- Definition of the platform requirements for the components.
- Containers creation including both the component and the platform requirements for it.
- Configuration of the containers during its instantiation.
- Logs communication
- Containers instantiation
- Containers instances stopping and deleting
- Persistency definition

With respect to communication it is envisioned that AFarCloud project will use:

- Containers registration into the registry
- Project level registry creation

With respect to distribution, it is envisioned that AFarCloud project will use:

- Download containers from the registry

With respect to update, it is envisioned that AFarCloud project will use:

- Download containers versions from the registry

Integration points:

With respect to integration technologies, it is envisioned that AFarCloud will use mainly the Maven plugin for Docker [20]. This will allow the project to obtain the actions registry (log) of what is happening in the different actions that supports and that are needed in the DevOps cycle:

- Build
- Run
- Stop
- Pushing into the registry
- Log

This registry integrates perfectly with Jenkins and allows us to analyse what has happened during the execution of the different activities.

The proposed approach may seem complex however, this approach provides the project with an editable configuration and adjustable to the needs of every project. All the files are stored with the project and are accessible and modifiable by the team working on it.

5.2.1. Artefact repository

5.2.1.1. Nexus

In AFarCloud, Nexus will be used as an Artefact Repository. Nexus manages software "artefacts" required for development. AfarCloud developers' builds can download dependencies from Nexus and can publish artefacts to Nexus creating a way to share artifacts. With Nexus a developer can control access to, and deployment of, every artefact from a single location.

Nexus is a free repository manager with universal support for popular components. It supports maven artefacts but also Docker registries as the Docker repository format.

5.2.2. Proposed deployment and development conventions

5.2.2.1. Proposed Deployment

Along the development and integration lifecycle, a test and a production environment will support the AFarCloud framework's CI pipeline. Both environments will be identical, so their definition should cover the both test and demonstration phases of the project. AFarCloud components will be organized in groups mapped to the demonstrators that will use case their functionality. These groups will represent their specific deployment options, considering contextual and software dependencies among components. The proposed VM characteristics to host each of the above Test and Deployment groups will be tailor-made given the specific requirements of the demonstrators. The requirements of these VMs will be measured given the needs in vCPU, RAM and HDD (storage).

5.2.2.2. Naming conventions

This section introduces the set of naming conventions for the source code, which will be.

`eu.AFarCloud.modulename.componentname.subcomponentname.`

Endpoints naming convention is as follows:

`/AFarCloud/[group]/[componentname]/`

Domain names will be for the development environment: `***.dev.AFarCloud.eu`, while for the production environment this will be the name that it will be used: `****.AFarCloud.eu`

Source code files heading shall follow the following format:

```
/*

* Copyright (c) 201x <<Company_name>>.
* All rights reserved. This program and the accompanying materials
* are made available under the terms of the
* <<licensing_schema_to_be_decided>> which accompanies
* this distribution, and is available at
* <<link of the information of the selected licensing schema>>
*
* Contributors:
*
* <<Full Name of the contributor(s)>> <<(Organization Name(s))>>
**Initially developed in the context of AFarCloud EU project www.AFarCloud.eu
*/
```

6. Conclusions

In this deliverable we have outlined the development, integration and validation strategy and plan. The document serves as an initial point of reference for the technical teams during the preparation of the different AFarCloud platform releases and the work that will be carried out in the rest of the WP7 tasks up to M36 at the end of the project.

A realistic demo-centric methodology tailored to the AFarCloud project´s nature and complexity has been described as well as the collaborative tools that are being set-up to support the technical partners into successfully reaching the milestones as identified and detailed in the Integration Plan.

A high-level planning has been provided, describing how to assign the initial project´s user requirements to the different platform releases and demonstrators. The initial product backlog has been described, and the objective to expand this during project execution to a detailed integration roadmap mapping user stories, to detailed development tasks, their dependencies, responsibilities estimated duration and timeline. The backlog will be a live list of functionalities and tasks always being updated and containing all features to be developed while it will also serve as a tool for the validation activities.

7. References

[1] K. Schwaber y J. Sutherland, «"The Scrum Guide",» Scrum.org, October 27, 2017.

[2] LeSS, “less.works,” 2014-2019. [Online]. Available: https://less.works/. [Accessed 29 04 2019].

[3] M. Cohn, « Succeeding with Agile: Software Development Using Scrum.,» Addison-Wesley, Upper Saddle River, NJ, 2010.

[4] ISTQB, “istqb.org,” ISTQB, 2016. [Online]. Available: https://www.istqb.org/certification-path-root/agile-tester-extension/agile-tester-extension-in-a-nutshell.html. [Accessed 29 04 2019].

[5] Atlassian, “What is DevOps?,” [Online]. Available: https://www.atlassian.com/devops. [Accessed 11 11 2018].

[6] Aniket Deshpande, "’DevOps’ an Extension of Agile Methodology – How It will Impact QA?," Software Testing Help.

[7] New Relic, "“Navigating DevOps - What it is and why it matters to you and your business”," New Relic, 2014.

[8] Git, "Git," [Online]. Available: https://git-scm.com/.

[9] Jenkins, "Jenkins," [Online]. Available: jenkins-ci.org. [Accessed 9 5 2017].

[10] Apache, "Apache Maven," [Online]. Available: maven.apache.org. [Accessed 09 05 2017].

[11] Docker, “Docker,” [Online]. Available: www.docker.com. [Accessed 09 05 2017].

[12] Chef, "Chef," [Online]. Available: www.chef.io. [Accessed 9 5 2017].

[13] Puppet Labs, "Puppet," [Online]. Available: puppetlabs.com/puppet/puppet-open-source. [Accessed 2017 05 09].

[14] GitLab, "GitLab," GitLab, [Online]. Available: https://about.gitlab.com/.

[15] Wikipedia, “Kaizen-Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Kaizen. [Accessed 15 05 2017].

[16] Wikipedia, "Lean-Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Lean_software_development. [Accessed 15 05 2017].

[17] Wikipedia, "SixSigma - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Six_Sigma. [Accessed 15 5 2017].

[18] Jenkins, "Jenkins API," [Online]. Available: https://wiki.jenkins-ci.org/display/JENKINS/Remote+access+API. [Accessed 15 05 2017].

[19] Cloud Foundry, "Warden," [Online]. Available: https://docs.cloudfoundry.org/concepts/architecture/warden.html. [Accessed 15 05 2017].

[20] Github, “Docker, Maven plugin,” [Online]. Available: https://github.com/fabric8io/docker-maven-plugin. [Accessed 15 05 2017].

[21] GitLab, “GitLab convdev,” [Online]. Available: https://docs.gitlab.com/ee/user/instance_statistics/convdev.html. [Accessed 29 04 2019].

[22] GitLab, “GitLab jobs,” [Online]. Available: https://docs.gitlab.com/ee/ci/yaml/README.html#jobs. [Accessed 29 04 2019].

[23] GitLab, “GitLab pipelines,” [Online]. Available: https://docs.gitlab.com/ee/ci/pipelines.html. [Accessed 29 04 2019].

8. Annexes

8.1. API description template

Title	*This field holds the description of the API*

URL *This field holds the relative URL to the described API. For simplicit Root URL can be cut off from this description and can be placed as a hyper text apove the API template*	
Method *This field holds the type of the Method used*	
`GET | POST | DELETE | PUT`	
URL Params *This field holds the parameters (if exist). Separated based on the fields below into required and optional.*	
Required:	
`id=[integer]`	*parameter description*
Optional:	
`image_id=[alphanumeric]`	*parameter description*
Data Params *This field holds the body payload of a post request.*	
Required:	
`id=[integer]`	*parameter description*
Optional:	
`image_id=[alphanumeric]`	*parameter description*
Success response <What should the status code be on success and is there any returned data? This is useful when people need to know what their callbacks should expect>	

`200`
 	
`Content: { }`
	response description

Error response *This field holds the list of all possible error responses. Doing that, helps prevent assumptions of why the enpoint fails and saves a lot of time during the integration process.*	
`404`	*response description*
Sample call *This field holds a possible sample call to the described endpoint in a curl-like format. Please, choose the format wisely so that is clear and easy to read by the interested parties.*	
Notes *This field holds any additional helpful info related to this endpoint.*	