Towards an Open, Secure, Decentralized and Coordinated Fog-to-Cloud Management Ecosystem

D5.2 mF2C reference architecture (integration IT-2)

Project Number	**730929**
Start Date	**01/01/2017**
Duration	**36 months**
Topic	**ICT-06-2016 - Cloud Computing**

Work Package	**WP5, PoC Integration and Demonstration Strategy**
Due Date:	*M36*
Submission Date:	*29/12/2019*
Version:	*1.9*
Status	*Final*
Author(s):	*Cristovao Cordeiro (SixSq), Sašo Stanovnik (XLAB), Matija Cankar (XLAB), Jens Jensen (STFC), Eva Marin Tordera (UPC), Denis Guilhot (WSL), Antonio Salis, Roberto Bulla, Paolo Cocco (ENG), Anna Queralt (BSC), Francesc Lordan (BSC), Marcin Spoczynski (INTEL), Jasenka Dizdarevic (TUBS), Francisco Carpio (TUBS)*
Reviewer(s)	*Jasenka Dizdarevic, Admela Jukan (TUBS)* *Jens Jensen (STFC)*

Keywords
Implementation, demonstration, integration

Project co-funded by the European Commission within the H2020 Programme		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

This document is issues within the frame and for the purpose of the mF2C project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730929. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

This document and its content are intellectual property of the mF2C Consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the mF2C Consortium or the Partners' detriment and are not to be disclosed externally without prior written consent from the mF2C Partners.

Each mF2C Partner may use this document in conformity with the mF2C Consortium Grant Agreement provisions.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	11/11/19	XLAB's contribution	Sašo Stanovnik, Matija Cankar (XLAB)
0.2	20/11/19	SixSq's contribution	Cristovao Cordeiro (SixSq)
0.3	20/11/19	STFC's contribution	Jens Jensen (STFC)
0.4	21/11/19	UPC's contribution	Eva Marin Tordera (UPC)
0.5	21/11/19	UC1 description and input	Denis Guilhot (WSL)
0.6	22/11/19	First contribution for UC3	Antonio Salis (ENG), Roberto Bulla, Paolo Cocco (ENG)
0.7	22/11/19	BSC's inputs	Anna Queralt (BSC), *Francesc Lordan (BSC)*
0.8	22/11/19	Intel's integration tests	Marcin Spoczynski (INTEL)
0.9	25/11/19	TUBS''s inputs	Jasenka Dizdarevic (TUBS), Francisco Carpio (TUBS)
1.0	26/11/19	Fix style, metadata and references	Cristovao Cordeiro (SixSq)
1.1	26/11/19	Update assignments and review	Cristovao Cordeiro (SixSq)
1.2	26/11/19	Re-write of section 3.6	Antonio Salis (ENG), Roberto Bulla, Paolo Cocco (ENG)
1.3	26/11/19	Address BSC's comments	Anna Queralt (BSC), Francesc Lordan (BSC)
1.4	27/11/19	INTEL's additions	Marcin Spoczynski (INTEL)
1.5	02/12/19	UPC fixes tables	Eva Marin Tordera (UPC)
1.6	02/12/19	Add conclusion	Cristovao Cordeiro (SixSq)
1.7	16/12/19	Merge reviews	Cristovao Cordeiro (SixSq)
1.8	18/12/19	Final review	Cristovao Cordeiro (SixSq)
1.9	29/12/2019	Quality Check comments assessed. Document ready for submission	Cristovao Cordeiro (SixSq), María Teresa García (ATOS)

Table of Contents

List of figures

List of tables

Executive Summary

This deliverable presents the functionalities of the mF2C [1] components in IT-2, the interactions and relations between components, and the joined-up functionalities of the whole platform. This document thus becomes a guide to the IT-2 release, for people who wish to understand how it works and for people who wish to reuse all or parts of the release.

It is important to note that this deliverable is a follow-up of D5.1 "mF2C Reference Architecture (integration IT-1)" [2], and thus builds on top of the content and knowledge already described in that deliverable.

This deliverable also takes into consideration all the architectural changes that have been added in past deliverables throughout IT-2.

At the time of writing this deliverable, all the technical development in mF2C is under a code freeze, which in practice means that all the information described in this document is final, and only subject to minor and non-disruptive changes.

Just like it was the case with the previous iteration, also in IT-2, source code and documentation are generally publicly available, components are loosely coupled using web services, and components are individually “packaged” and deployed in containers which can, in turn, be “composed” to form more complex “super-components” which can be tested as if they were a single component. A better understanding this loosely coupled architecture through the integration tests described in this document.

The ultimate validation of the platform is through integration tests and practical use cases. For that reason, besides the integration tests we've also put together a validation team, composed of mF2C project members who are not actively participating in the technical mF2C development, to assess the readiness of the platform and, together with the use case owners, provide practical feedback to the developers.

1. Introduction

This document presents the final integration of the mF2C [1] components for IT-2 to describe the final prototype of the mF2C platform, and to document the results of integration tests, functional validation and security audits.

The deliverable *D2.7 mF2C Architecture (IT-2)* [3]presented the updated architecture for the IT-2, thus this document is building on top of that knowledge plus the concepts and lessons learned from *D5.1 mF2C Reference Architecture (integration IT-1)* [2].

The architecture and respective workflows have been refined and updated since the release of deliverables D2.7 [3] and D5.1 [2]. Therefore, this document also reports on the modifications of functionalities supported by the mF2C blocks for IT-2.

1.1. Purpose

The purpose of this deliverable is to demonstrate and document the final state of the mF2C platform, indicating its final architecture, workflows, and instructions on how to install, use and test it. With this document, we also assess the IT-2 PoC in terms of provided functionality, security, scalability and relevance to the mF2C use cases.

1.2. Structure of the document

This document is structured as follows:

- Section 2 gives an insight on the goals of the integration work in IT-2, including the updates on the mF2C architecture and workflows that have been added since D5.1 [2] and D2.7 [3];
- Section 3 takes the final mF2C prototype and explains how to install, use and test it, including the description of the testbed used both for these tests and for the validation of the use cases;
- Section 4 provides a scalability analysis of the final platform.

1.3. Glossary of Acronyms

Acronym	Definition
AC	Agent Controller
ACL	Access Control List
API	Application Programming Interface
CA	Certificate Authority
CAU	Control Area Unit
CIMI	Cloud Infrastructure Management Interface
CSR	Certificate Signing Request
DER	Distributed Execution Runtime
ECC	Elliptic Curve Cryptography
EMS	Emergency Situation Management
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HDD	Hard Disk Drive
HTTP	HyperText Transfer Protocol
IMA	Industrial Management Application
IoT	Internet of Things
IT	Iteration
JSON	JavaScript Object Notation
JWT	JSON Web Token

Acronym	Definition
LAN	Local Area Network
MAC	Media Access Control (network address)
MQTT	MQ Telemetry Transport
NIC	Network Interface Card
OS	Operating System
PKI	Public Key Infrastructure
PM	Platform Manager
PoC	Proof of Concept
QA	Quality Assurance
QoS	Quality of Service
REST	Representational State Transfer (web service)
RSA	Rivest/Shamir/Adelman (public key system)
SIEM	Security information and event management
SLA	Service Level Agreement
UC	Use Case
URL	Uniform Resource Locator
VPN	Virtual Private Network

Table 1. Acronyms

2. IT-2 Scope

For IT-2, the main goal is to finalize the prototype developed during IT-1, in such a way that the overall end solution can be adopted by external users to mF2C. To achieve this goal, several milestones need to be met:

- practical demonstration of the architecture and workflows defined throughout IT-2 and described in past deliverables – through functional demonstrations;
- validation of all mF2C features and user functionality – through the validation team tests on functionality and security;
- full integration of mF2C components – through integration tests;
- suitability for use in real life scenarios – through the use cases.

These apply to the architecture design of the mF2C Agent and Microagent. Section 3 below addresses all of these milestones.

2.1. Final Architecture

In the final mF2C architectural design for IT-2 a modified architectural solution has been agreed upon and described in deliverable D2.7 [3]. The modification of certain components, interfaces and functionalities is a natural and necessary step in the architecture evolution, based on experiences with the IT-1 practical implementation. The final IT-2 architecture includes two, instead of one, main mF2C architectural entities - Agent and Microagent, with their respective architectural design presented in Figures 8 and 15 of D2.7 [3]. It also defines a slightly modified version of the standard mF2C Agent, adapted as described in Section 5.1 of D2.7 [3]. The differentiation between Platform Manager (PM) and the Agent Controller (AC) as the main building blocks of the Agent remains in the final architecture. The final design of the PM and integration of its modules has been provided in deliverables D4.4 [4] and D4.6 [5], respectively, and the final design of the AC and integration of its modules in deliverables D3.4 [6] and D3.6 [7].

2.2. Workflows update

After the design for IT-2 of the Agent Controller and the Platform Manager described in D3.4 [6] and D4.4 [4], respectively, some minor updates were made during the implementation phase. In this section, we summarize these updates.

2.2.1. Workflow updates for Agent Controller

Regarding the Agent controller, one of the workflows that has been slightly modified is the workflow for user registration using the mF2C website (corresponding to Figure 8 in D3.4 [6]).

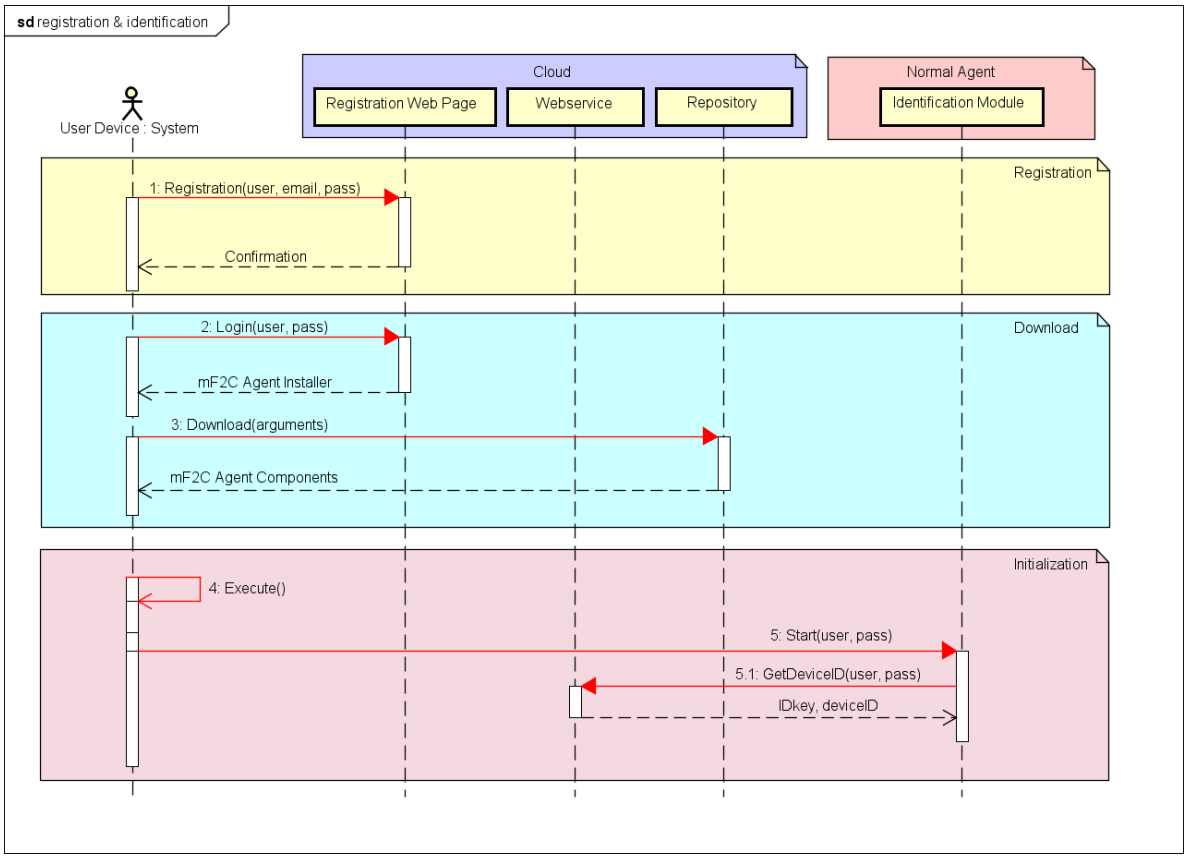

Figure 1. Registration and download from webpage (workflow updated)

As it is shown in Figure 1, in the download section, the arguments of step 2 are user and password and the response does not include any argument. Also, for the steps 5 and 5.1, the parameters sent from the user to the identification module, and from the identification module to the webservices are user and password.

The arguments for the third and last red solid arrow are also user and password.

Finally, the last response, from the webservice to the identification module includes both deviceID and IDKey.

Other workflows that are slightly modified, are the related to the Agent and leader initialization, corresponding to Figures 9 and 10 in D3.4 [6]. We only show the changes of the Agent initialization in Figure 2, because the changes for the leader initialization are the same. Basically, the changes are related to security issues and can be summarized as follows:

- Policy calls CAU-client with only 2 parameters: leader(device)Id and deviceID. The CSR is generated by the CAU-client
- The CAU stores the Agent's deviceID and the RSA Public Key associated with the new certificate in its key-value cache. The key is used in security-related cryptography functions.

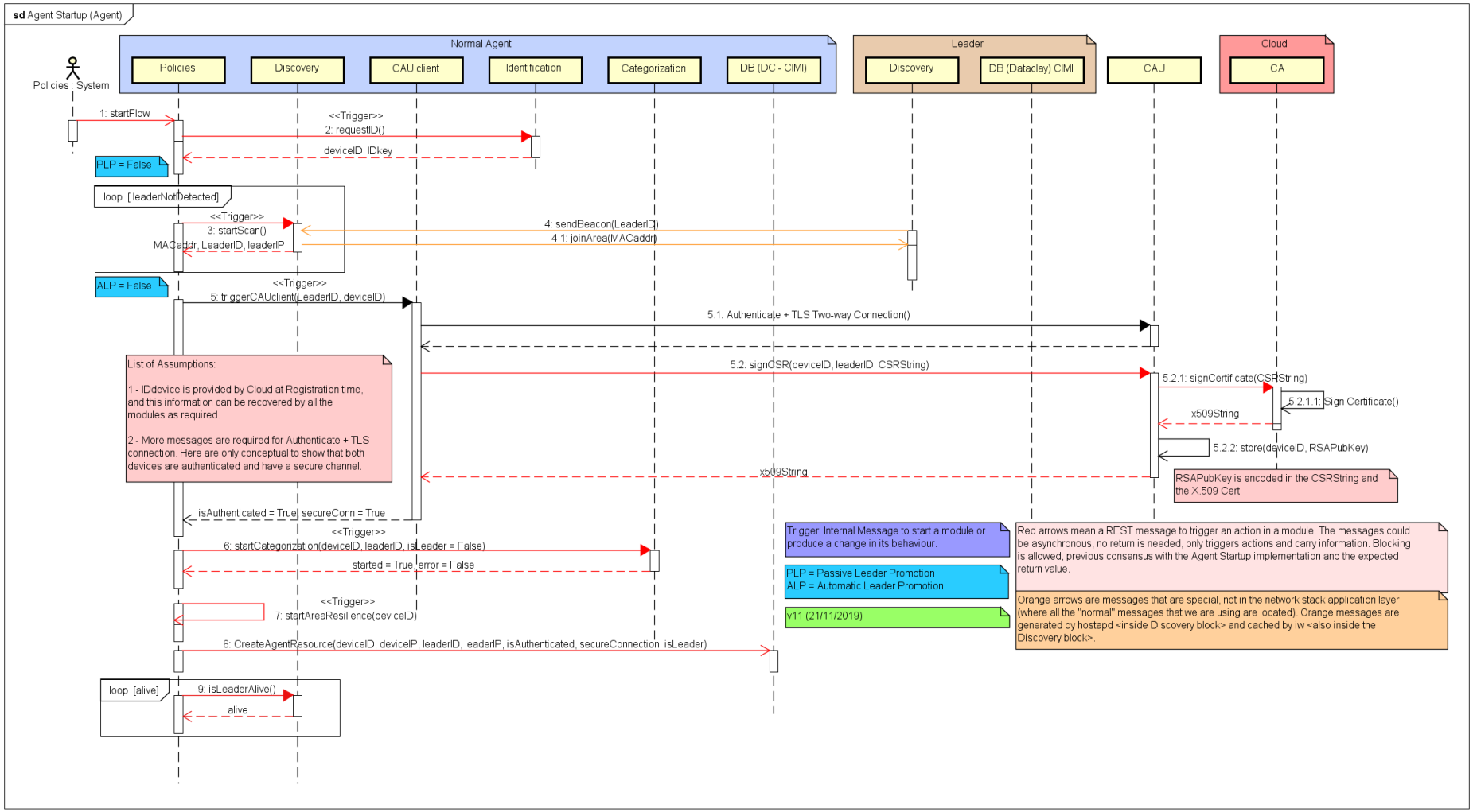

Figure 2. Agent initialization (workflow updated)

2.2.2.Workflow updates for Platform Manager

Regarding the Platform Manager, two of the workflows for Service Management module have been modified, QoS Providing and QoS Enforcement (corresponding to Figures 13 and 14 in D4.4 [4]).

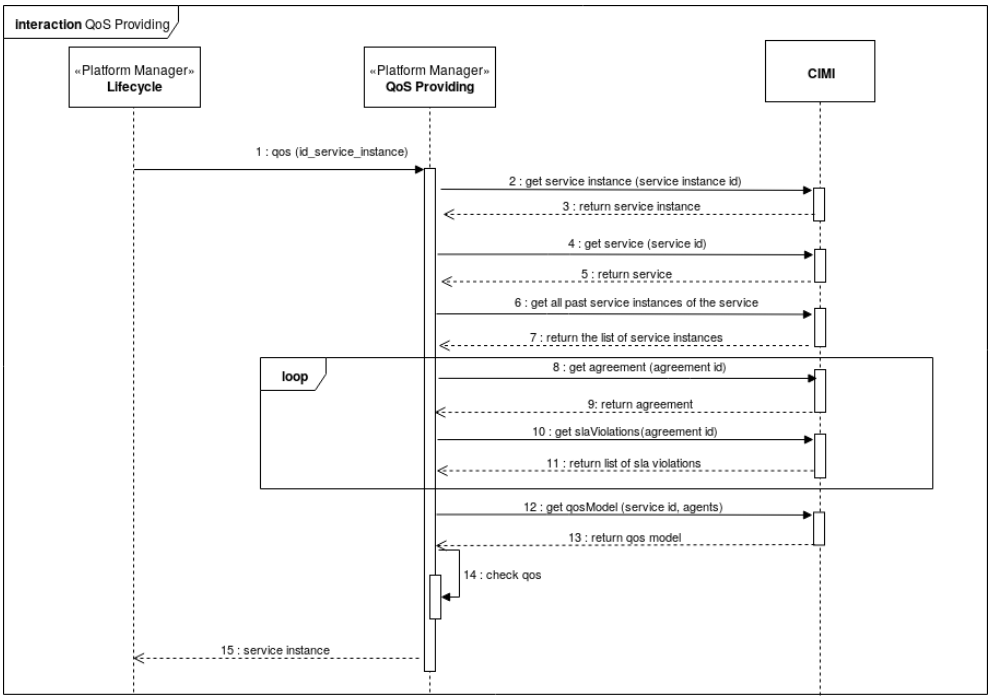

Figure 3. Service Management - QoS providing (workflow updated)

The role of QoS Providing has not changed from D4.4 [4], which is in determining which Agents from the chosen set of Agents for service execution can actually be used for the execution. The steps from 1 to 5 remain the same as before. In step 6, QoS Providing now requests all past service instances from CIMI, which returns the list (step 7). The step 6 from D4.4 [4], where based on the agreement id in the service, QoS Providing gets SLA violations information from CIMI is now done in step 10 and is preceded with step 8 in which QoS Providing requests the agreement from CIMI, which is returned in step 9. The remaining steps remain unaltered.

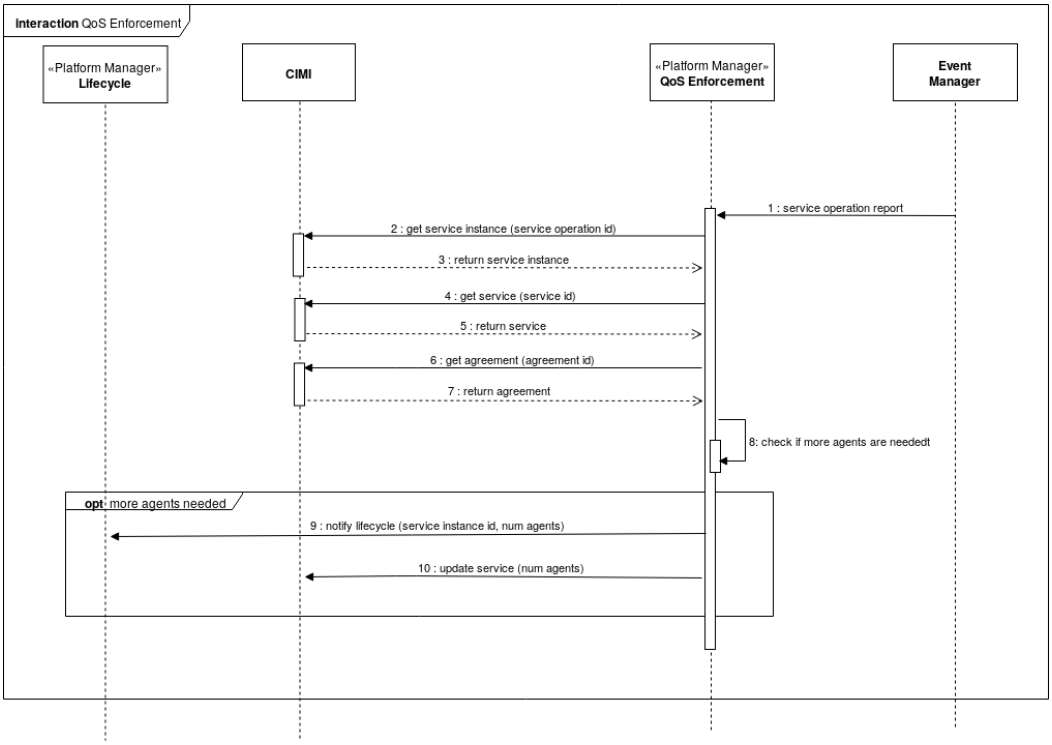

Figure 4. Service Management - QoS Enforcement (workflow updated)

The main functionality of QoS Enforcement remains the same, as the module which ensures the expected QoS of a service operation is unchanged. However, since this functionality was implemented after submission of D4.4 [4], the workflow has changed, with the new one shown in Figure 4.

- After a new service starts execution, COMPSs stores the service operation report (with the information on service execution time) in CIMI. The Event Manager sends a service operation report to QoS Enforcement.
- [2-5] After this report is received, QoS Enforcement requests first the service instance and then the service from CIMI, which returns it.
- [6-7] QoS Enforcement requests service level agreement from CIMI so it can compare the expected service operation time with the values in the SLA agreement.
- [8-10] If the value from the service operation report does not satisfy the SLA constraint, the QoS Enforcement requests the Lifecycle to add more Agents for the execution of the service.

3. mF2C Agent and Microagent validation

The adopted methodology for validating both the mF2C Agent and Microagent is:

- set up a controlled QA environment (testbed) for testing all new software updates to the mF2C Agent and Microagent before merging the codebase into a releasable state;
- following a step by step guideline, install the mF2C Agent and Microagent onto the testbed and/or personal user devices;
- run the integration tests against the installed mF2C Agent (and report on the failed tests);
- run a functional test (known as "Hello World") against the installed mF2C Agents, to make sure all user functionality is working as expected;
- perform a security audit against the running Agents to make sure all security components are working properly;
- run the use case applications in all the installed mF2C Agents.

3.1. Testbed

In IT-2 we introduced a closed and controlled environment addressed to the project developers. The objective was to avoid problems related to hardware (missing components, non-proper configuration, etc.) in the early stages of integration tasks. Once the integration of the components was close to stable, the testbed has been used for debugging and testing purposes.

It consists of four physical machines (without virtualization) with an Intel Xeon E5504 2.00GHz CPU, 8GB of DDR3 1333MHz RAM, and 160GB of HDD disk (16GB dedicated to Docker). The machines are connected to each other by using NetXtreme II BCM5709 Gigabit Ethernet network card (up to 1Gbps), connected to a Gigabit network switch. Additionally, each machine has a wireless NIC with the 802.11b/g/n Ralink RT5370 chipset that operates in the 2.4GHz band. The network is isolated from any traffic other than that generated by the machines themselves, and the external access to the testbed is done using a VPN connection. The OS installed is a Debian 9.3 (kernel version 4.9.0-8-amd64), with Docker version 18.09.2 (build 6247962), and Docker Compose version 1.23.2 (build 1110ad01).

Additionally, the testbed also contains two RaspberryPi 3 Model B connected to the same network, used to test the Microagent. The processor inside the board is a 1.2GHz Quad-Core ARM Cortex-A53 CPU, with 1GB LPDDR2 RAM, and microSD up to 16GB for the disk. The Raspberry is attached to the network via its 10/100 BaseT Ethernet (up to 100Mbps), and it also contains a wireless NIC 802.11 b/g/n Wireless LAN (Broadcom BCM2387 chipset). The OS is a Raspbian (Debian 9.11 kernel version 4.19.66-v7+), with Docker version 19.03.3 (build a872fc2), and Docker Compose version 1.23.2 (build 1110ad0).

The testbed is physically located at the UPC facilities, and to access it the academic networks are used, including GÉANT, RedIRIS, and Anella Cientifica.

3.2. Installation

In the next two subsections, we describe how to install the two versions of the Agent:

- Agent → in the roles of Agent, leader Agent, and cloud Agent
- Microagent → for the ARM architectures.

3.2.1. Agent

3.2.1.1. Prerequisites

- Ubuntu 16.04 / Debian 9 based distribution (Others may work as well).

- More than 8GB of RAM and 20GB of available disk.
- `docker` and `docker-compose` installed.
  - How to install docker, https://docs.docker.com/install/linux/docker-ce/ubuntu/ -install-docker-ce.
  - How to install docker-compose, https://docs.docker.com/compose/install/.
- Additional packages are required to run the installation and test scripts:
  - `iw`: Tool for configuring Linux wireless devices. More info https://wireless.wiki.kernel.org/en/users/Documentation/iw.
  - `jq`: Command-line JSON processor. More info https://stedolan.github.io/jq/download/.
  - `git`: Version control system. More info https://git-scm.com/book/en/v2/Getting-Started-Installing-Git.
- An Internet connection.
- Ports `80`, `443`, `1034`, `46000`, `46300`, `46040`, `46020`, `8086`, `8181`, `7474`, and `9001` must be free.
- Disable the network-manager service or tag the wireless NIC as unmanaged. More information: https://github.com/mF2C/ResourceManagement/tree/master/Discovery#prerequisites-1

3.2.1.2. mF2C Agent installation

Before the installation, to run the mF2C Agent is necessary to be registered as a valid mF2C User. Register a new user in the mF2C registration dashboard: http://dashboard.mf2c-project.eu:800/main.html.

Validate the registration on the validation mail sent to the specified email. This step can be omitted if you are already registered. One user can have multiple agents registered.

Installation script

1. Clone the main mF2C repository and enter into the directory or download the `.zip` file from the registration dashboard

   ```
   git clone https://github.com/mF2C/mF2C
   cd mF2C/agent
   ```

   If the agent is downloaded using the registration dashboard, see Figure 5, unzip the file that contains the agent configuration files and a `json` setup file with the sensors/actuators information.

   ```
   unzip mF2C.zip && unzip agent.zip && rm agent.zip && mv mF2C-master/
   mF2C
   mv setup.json mF2C/agent/
   cd mF2C/agent
   ```

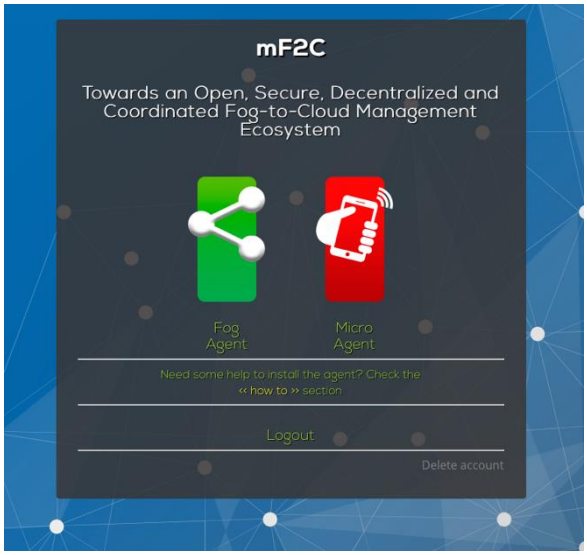

Figure 5. mF2C dashboard

2. Execute the mf2c.sh script:

   ```
   ./mf2c.sh
   ```

   type `mf2c.sh -L` if you want to deploy a Leader Agent or `mf2c.sh -C` to deploy a Cloud Agent.

   - Follow the instructions inside the script.

3. Once installation is completed, wait until all components are healthy (a component's health status is defined by the corresponding health check which is incorporated in the installation Compose file).

   ```
   ./mf2c.sh -s
   ```

   NOTE: Running an Agent with an unhealthy component may cause unexpected errors in the whole stack. Run `docker ps` to quickly check the health of every component.

4. To verify if the installation has been successful, run the hello-world.sh test.

   ```
   ./hello-world.sh
   ```

   add `--include-tests` argument to run all the component integration tests automatically.

Manual install

NOTE: Manual installation does not configure the wireless interface.

1. Clone the main mF2C repository and enter into the directory

   ```
   git clone https://github.com/mF2C/mF2C
   cd mF2C/agent
   ```

   If the agent is downloaded using the registration dashboard, see Figure 5, unzip the file that contains the agent configuration files and a `.json` setup file with the sensors/actuators information.

   ```
   unzip mF2C.zip && unzip agent.zip && rm agent.zip && mv mF2C-master/ mF2C
   mv setup.json mF2C/agent/
   cd mF2C/agent
   ```

2. Create a new `.env` file:

```
isLeader=False
isCloud=False
MF2C_CLOUD_AGENT="213.205.14.15"
PHY=
WIFI_DEV_FLAG=
usr="<username>"
pwd="<password>"
agentType="<type>"
targetDeviceActuator="<actuators>"
targetDeviceSensor="<sensors>"
```

   - Replace `<username>` and `<password>` with your mF2C credentials (from registration dashboard).
   - Replace `<type>` with `"2"` for a Normal or Leader Agent, or with `"1"` for the Cloud Agent.
   - Set `isLeader=True` to deploy a Leader or `isCloud=True` to deploy a Cloud Agent.
   - Replace `<actuators>` and `<sensors>` with the ones from the `setup.json` if the file exists.

3. Launch the Agent.
   `docker-compose -p mf2c up -d`
4. Wait until all components are healthy.
   `docker-compose -p mf2c ps`

   NOTE: Running an Agent with an unhealthy component may cause unexpected errors in the whole stack.

5. To test if the installation has been successful, run the `hello-world.sh` test.
   `./hello-world.sh`

   add `--include-tests` argument to run all the component integration tests automatically.

3.2.1.3. Uninstall the mF2C Agent

Installation script

1. Inside the `mF2C/agent` directory, run the `mf2c.sh` script with the following arguments:
   `./mf2c.sh -S`
2. Check if the `.env` file has been removed, the script finished without errors, and all the component containers are stopped
   `./mf2c.sh -s`

Manual uninstall

1. Inside the `mF2C/agent` directory, run the following command:
   `docker-compose -p mf2c down -v`
2. Check if the mF2C Agent has been successfully uninstalled, and all the component containers are stopped
   `docker-compose -p mf2c ps`

Clean-up

The mF2C stack uses a considerable amount of resources, specially disk and RAM. Once the Agent is uninstalled, is highly recommended to remove the downloaded docker images. More information https://docs.docker.com/config/pruning/.

```
docker image prune -a
```

3.2.1.4. Topology formation

The mF2C Agent automatically builds the topology (connect to the Leader/Cloud Agent, resource information sync, leave detection, etc...) using primarily the Discovery mechanism, https://github.com/mF2C/ResourceManagement/tree/master/Discovery - mf2c-discovery.

However, if the device does not have a Wireless NIC or a Leader is not detected, a secondary mechanism is used to connect the Agent to the Cloud Agent. This connection is done by joining to the mF2C VPN, https://github.com/mF2C/vpn - establishing-credentials-for-authentication, using the mF2C user credentials.

Both mechanisms are automatically triggered at the Agent installation, except for the Cloud Agent, which is considered to be at the top of the hierarchy and thus does not have any mF2C leader.

Manual topology

If the scenario or the device does not have the required capabilities regarding the wireless NIC, the mF2C Agent also allows a static topology, by specifying the IPs in each Agent.

To create the static topology, in each device modify the `docker-compose.yml` file as follows:

- Inside the file, at the `policies` service definition, add two environment variables:

```
- "leaderIP=192.168.4.44"
- "deviceIP=192.168.4.4"
```

  - Replace both IPs with the Leader IP and the device's own IP for the Agent. The Leader will automatically connect to the specified Cloud IP at the installation (if VPN is available).

By default, static variables will only be active if `Discovery` has failed or a Leader is not detected. If static IP variables are not set, VPN IPs are used.

It should be noted that IPs must be numerical and valid. If a wrong IP is specified, the topology may fail as well.

3.2.2.Microagent

The mF2C Microagent is a smaller and simpler IoT-ready distribution of the mF2C Agent, which is compatible with ARM architectures and thus fits into single-board computers like Raspberry Pi. The Microagent also runs with a loosely coupled microservice-based architecture and can also be deployed via Docker.

Due to its simplified nature (as described in D2.7 [3]), the Microagent only contains a small set of critical mF2C components that will ensure that the corresponding device can run mF2C services. In practice, this means that the mF2C Microagent **does not offer the following functionalities**: leadership capabilities, local storage, local resource management API, local device management, local service management and SLA and QoS features.

Therefore, the mF2C Microagent shall be seen purely as a worker node, incapable of managing other Agents and only executing whatever workloads its Agent leader mandates.

Prerequisites

- ARM-based device (preferably a Raspberry Pi);
- Docker Engine (version 18 or higher) and Docker Compose (version 1.23.2 or higher);
- with (only needed for an installation from Nuvla) Docker Swarm mode enabled.

Connectivity

When installed for the first time, the Microagent requires an internet connection so that the device can be validated with mF2C.

After that, if there are no other mF2C Agents in the vicinity that can act as leaders, the Microagent will default its leader to the mF2C Cloud Agent via the VPN client component. It should be noted again, that Microagents cannot be leaders.

Note: in order to connect to mF2C leaders in the vicinity, the Microagent relies on the existence of an external WiFi dongle

Installation for the Raspberry Pi

- From Nuvla[1]

1. go to nuvla.io
2. create an account and login
3. confirm that you can see your Docker Swarm infrastructure from Nuvla
    a. if your Raspberry Pi is running in Swarm mode, then you can add it and see it from the Infrastructures tab, or
    b. if your Raspberry Pi is already a NuvlaBox[2], then you can see it (or transform it into one) from the Edge panel
4. go to the App Store and search for "mF2C Microagent"
5. click launch and select the corresponding credential for your infrastructure
6. set your mF2C username and password in the environment variables
7. click launch and wait for the application to be ready

- Manually

1. *ssh* into your Raspberry Pi
2. make sure you have Docker and Docker Compose installed
3. download the latest Microagent compose file from GitHub[3]
4. set your mF2C credentials as environment variables: *export MF2C_USER=<youruser>* and *export MF2C_PWD=<yourpassword>*
5. in the same directory as the downloaded compose file, run docker-compose up -d
6. wait a few minutes. The Microagent will be ready once you can access *http://localhost:46000/api/v2/lm.html*

Uninstall

- From **Nuvla**

1. go to nuvla.io
2. go to the dashboard
3. find the Microagent app you want to uninstall and click stop

[1] https://nuvla.io
[2] https://sixsq.com/products-and-services/nuvlabox/overview
[3] https://github.com/mF2C/mF2C/blob/master/Microagent/docker-compose.yml

- Manually

1. *ssh* into your Microagent's device
2. find the installation folder, where your original compose file is
3. run "docker-compose down –v"

3.3. Integration Tests

The following tables comprise all the integration tests that have been developed for the mF2C Agent.

Target Modules	CIMI, DataClay		
Integration Test Description	This integration test verifies that the API server CIMI is correctly working by issuing a series of standard HTTP requests to test the connection between CIMI and DataClay. These requests target all the CRUD operations (CREATE, READ, UPDATE, DELETE) plus the authorization mechanisms enforced by CIMI.		
URL	https://github.com/mF2C/mF2C/blob/master/agent/tests/cimi.sh		
Owner	SixSq		
Initial Condition(s)	• The mF2C Agent needs to be running and in a healthy state		
Number of Tests	11		
Overall Status	Passed		
Steps	Simply execute the script from the URL above		
Results	**Check if cloud-entry-point exists**	**SUCCESS**	
	Check if user authorization is working properly	**SUCCESS**	
	Verify that 405 error code is given on invalid resource collection	**SUCCESS**	
	Verify that 404 error code is given for querying non-existent user	**SUCCESS**	
	Try to create new user	**SUCCESS**	

	Try to create the same user	**WARNING**	The requested was rejected, as expected, but the return error code is 400 instead of 409 (conflict). Issue is being tracked in https://github.com/mF2C/dataClay/issues/55
	Create a user profile resource	**SUCCESS**	
	Read a resource by its ID	**SUCCESS**	
	Filter for a set of resources	**SUCCESS**	
	Update the user profile created above	**SUCCESS**	
	Delete the user	**SUCCESS**	
Problems	Only one test showed up as a warning, related with the right propagation of error from the database to CIMI. This is considered as a warning and not an actual **FAILURE** because the system still behaves as expected.		
Required changes	Fix the propagation of HTTP errors in case of conflicts (code 409)		
Comments			

Table 2. CIMI-DataClay integration tests

Target Modules	DER, DataClay, Lifecycle Manager, ServiceManager, SLA enforcement
Integration Test Description	The test deploys and executes a COMPSs-based service. When the main method of the es.bsc.compss.agent.test.Test class is invoked selecting as core elements the methods from the interface es.bsc.compss.agent.test.TestItf, the DER generates a workflow with 3 tasks per iteration (the number of iterations is passed in as a parameter of the method, by default 20). The first task has no parameters, the second task generates an object, and the third task uses the value generated by the previous.
URL	https://github.com/mF2C/mF2C/blob/master/agent/tests/der.sh
Owner	Barcelona Supercomputing Center
Initial Condition(s)	• At least one regular mf2c Agent to be running on the same machine as the test

Number of Tests	4		
Overall Status	Passed		
Steps	• Execute the der.sh script		
Results	Validate that the SM and the LM properly deploy the service	Y	
	Validate that the LM properly launches the execution of the method	Y	
	Validate that ServiceOperationReports are properly published so SLA Enforcement can read them	Y	
	Validate that Dataclay works properly and allows object sharing among multiple Agents	Y	
	Validate that operation ends and execution time is properly published on the ServiceOperationReport	Y	
Problems			
Required changes			
Comments			

Table 3. DER integration tests

Target Modules	Landscaper, CIMI
Integration Test Description	This integration test verifies that Landscaper is correctly collecting mF2c specific data from CIMI and Docker API. The test consists of a series of HTTP request calls to be made against the Recommender API and the expected responses.
URL	https://github.com/mF2C/mF2C/blob/master/agent/tests/landscape.sh
Owner	Intel
Initial Condition(s)	Both the CIMI Device and Device Dynamics data need to be populated
Number of Tests	3
Overall Status	Passed

Steps	Execute the script from the URL above		
Results	**Test if hwlock host metadata is properly stored in the Landscaper**	**SUCCESS**	
	Test if mf2c_device_id entry is stored with proper value in Landscaper	**SUCCESS**	
	Test if docker virtual layer metadata is properly stored in Landscaper	**SUCCESS**	
Problems	When device or device dynamic entries in CIMI are not populated the test will return a failure. This is considered a warning and not an actual FAILURE because the system still behaves as expected.		
Required changes	None Required		
Comments			

Table 4. Landscaper-CIMI integration tests

Target Modules	Recommender, Landscaper
Integration Test Description	This integration test verifies that the topology delivered in Landscaper is correctly parsed and can be used by the Recommender. The test consists of a series of HTTP request calls against the Recommender API and the expected responses.
URL	https://github.com/mF2C/mF2C/blob/master/agent/tests/recommender.sh
Owner	Intel
Initial Condition(s)	Both the Topology and Device Dynamics data need to be populated
Number of Tests	3
Overall Status	Passed
Steps	Execute the script from the URL above

Results	**Execute optimal Recommender pipeline**	**SUCCESS**	
	Execute analysis endpoint for average analytics	**SUCCESS**	
	Execute optimal Recommender pipeline with additional telemetry filters	**SUCCESS**	
Problems	When device or device dynamic entries in CIMI are not populated the test will return a failure. This is considered as a warning and not an actual FAILURE because the system still behaves as expected.		
Required changes	None Required		
Comments			

Table 5. Recommender-Landscaper integration tests

Target Modules	Service Manager (Categorizer, QoS provider, Qos Enforcement), Lifecycle Manager, COMPSs, CIMI, Event Manager, SLA Manager		
Integration Test Description	This integration test checks that services are correctly categorized, that the QoS provider is checking service instances properly and that QoS Enforcement works properly with Lifecycle Manager, Event Manager and SLA Manager, for COMPSs applications.		
URL	https://github.com/mF2C/mF2C/blob/master/agent/tests/service-manager.sh		
Owner	TUBS		
Initial Condition(s)	One regular Agent connected to a leader Agent (all containers healthy in both cases).		
Number of Tests	11		
Overall Status	Passed		
Steps	• Execute the script from the link above in the leader Agent.		
Results	1) submit an SLA template into CIMI with guarantees “execution_time<5” for COMPSs operations	**SUCCESS**	

	2) submit a service into CIMI with previous SLA template that requires COMPSs	**SUCCESS**	
	3) retrieve the same service from CIMI and verify that the has been categorized by the Service Manager	**SUCCESS**	
	4) call the Lifecycle Manager to create a service instance of the service and verify that it was ok	**SUCCESS**	
	5) check QoS (provider) of the service instance	**SUCCESS**	
	6) check if agreement is created	**SUCCESS**	
	7) check if qos-model (for provider) is added to CIMI	**SUCCESS**	
	8) check COMPSs Agent availability	**SUCCESS**	Sometimes this test can fail when Lifecycle cannot contact COMPSs for some reason
	9) start an operation with duration 60 seconds	**SUCCESS**	
	10) check for service-operation-reports	**SUCCESS**	
	11) finally, check if new devices were successfully added to the service-instance. This implies that Event Manager successfully sent the service-operation-reports to QoS Enforcement and this one notifies to Lifecycle Manager.	**SUCCESS**	This is successful if it is possible to add the regular Agent to the service-instance, otherwise it fails.
Problems	Only two issues can make this test fail, but are not considered failures, but warnings. Make sure that both Agents are not running other containers or tasks.		
Required changes	Not required changes.		

Comments	If the same test is executed in the regular Agent, the last test would fail, which is normal because the regular Agent cannot add other devices to the execution of a service.

Table 6. Service Manager-Lifecycle Manager-Event Manager-SLA Manager integration tests

Target Modules	Policies, Resource Categorization, Discovery, Lifecycle Manager, CAU client, Identification, CIMI		
Integration Test Description	The main objective of the integration tests for this component is to check the start-up workflow. In other words, if the tests performed are successful, implies that the triggers have been correctly executed, all involved components are running and the Policies module can work together with them. This integration checks that: • the internal API is attending requests. • the Agent Resource has been created and stored into Dataclay. • the Agent has a role. • the workflow has started. • the Identification trigger has been successful and replied with the deviceID. • as a normal Agent, the trigger to find a leader has been successful, as leader, checks if the trigger to start broadcasting beacons has been successful. • the CAU client trigger has been successful. • the Resource Categorization trigger has been successful.		
URL	https://github.com/mF2C/mF2C/blob/master/agent/tests/policies.sh		
Owner	UPC		
Initial Condition(s)	All the implied components are running and healthy (they are not starting).		
Number of Tests	8		
Overall Status	Passed		
Steps	Execute the script from the URL above.		
Results	API	**SUCCESS**	
	Agent Resource Creation	**SUCCESS**	
	Role Selected	**SUCCESS**	
	Agent Start Workflow	**SUCCESS**	
	Identification	**SUCCESS**	
	Discovery	**SUCCESS**	

	CAU client	SUCCESS	
	Resource Categorization	SUCCESS	
Problems	This test only checks if the Agent resource has been created, not the content nor the validity of the data stored, causing a false positive (e.g. an IP that is no longer reachable). The health check of the component marks the state as unhealthy if the data has incorrect values.		
Required changes	None Required		
Comments			

Table 7. Polices integration tests

Target Modules	Resource Categorization, CIMI, DataClay, Landscaper, Recommender, Analytics, Sensor Manager		
Integration Test Description	This integration checks that device, device-dynamic and Agent resources are properly stored in DataClay and reachable through Landscaper, Analytic, Sensor Manager and Recommender.		
URL	https://github.com/mF2C/mF2C/blob/master/agent/tests/resource-categorization.sh		
Owner	UPC		
Initial Condition(s)	CIMI and DataClay are up and running.		
Number of Tests	4		
Overall Status	Passed		

Steps	• Execute the script of the URL above, which will perform:
	`curl -X GET "http://localhost/api/device" -H "accept: application/json" (checking the device is exist of not)`
	`curl -X GET "http://localhost/api/device-dynamic" -H "accept: application/json" (checking the device-dynamic is exist of not)`
	`curl -X GET "http://localhost/api/agent" -H "accept: application/json" (checking the Agent is exist of not)`		

	Landscaper can reach device and device dynamic resources	SUCCESS	
	Recommender can reach device,	SUCCESS	

	device dynamic and Agent resources		
	Analytic (LM) can reach device, device dynamic and Agent resources.	SUCCESS	
	Sensor Manager can reach device, device dynamic and Agent resources.	SUCCESS	
Problems			
Required changes	None Required		
Comments			

Table 8. Resource categorization integration tests

Target Modules	Identification		
Integration Test Description	This integration checks that after the registration, the deviceID and the IDkey are stored and reachable by the identification module.		
URL	https://github.com/mF2C/mF2C/blob/master/agent/tests/identification.sh		
Owner	UPC		
Initial Condition(s)	The user must be registered in the system and must have valid credentials		
Number of Tests	1		
Overall Status	Passed		
Steps	• Execute the script of the url above		
Results	Identification can access to the IDkey and deviceID	SUCCESS	
Problems			
Required changes	None Required		

Table 9. Registration Integration test

3.4. Functional Tests

We now describe the set of functionalities provided by mF2C. These tests can be run either individually or as part of some other test:

1. User Sign up/in
   - The registration of users can be done through the GUI of mF2C by specifying a valid email address, selecting a user name and a password. After the user is registered, he/she will receive a validation email. Once the address is validated, the user can download and/or install the Agent or the Microagent.
   - TEST: Using a web browser, the user can access the GUI in the cloud, with his/her validated credentials. To install the Agent or Microagent software the user must use valid credentials, otherwise the user is not identified, and the software cannot be successfully installed.

2. Auto clustering (agents leaving/joining)
   - The discovery component provides mechanisms to allow a leader to send 802.11 beacons so that a regular Agent could scan for those beacons and discover the leader.
   - TEST: Assuming a leader running. On the child (non-leader) Agent side, perform the scan REST call GET /api/v1/resource-management/discovery/scan/<interface_name>. A JSON with the detected leader information will be returned.
   - The discovery component provides mechanisms to detect the events of an Agent joining/leaving based on events monitored by the hostapdtool.
   - TEST: One leader running. Start the logs of the discovery container on the leader using docker logs mf2c_discovery_1 -f.
   - For an Agent joining, perform the join REST call POST /api/v1/resource-management/discovery/join/, with DATA {"interface":"wlan0" , "bssid":"aa:bb:cc:dd:ee:ff"} where aa:bb:cc:dd:ee:ff should be replaced with the MAC address of the wlan0 interface. In the leader discovery logs, a message will appear saying “At time_of_the_event aa:bb:cc:dd:ee:ff has joined” where aa:bb:cc:dd:ee:ff is the MAC address of the Agent.
   - For an Agent leaving, perform the event of an Agent leaving by either (i) shutting down the machine where the Agent is running (ii) shutting down the mF2C Agent via ./mf2c.sh -S or (iii) performing an explicit REST call to “unjoin” from the Agent side (i.e. PUT /api/v1/resource-management/discovery/join/). In the leader discovery logs, a message will appear saying “At time_of_the_event aa:bb:cc:dd:ee:ff has left” where aa:bb:cc:dd:ee:ff is the MAC address of the Agent.
   - For an Agent leaving/die-out, perform the device-dynamic REST call GET, on the leader side, as follows:

     curl -X GET "http://localhost/api/fog-area" -H "accept: application/json"

     to check the “status” field for the corresponding Agent. If the “status” is “disconnected” or “unavailable”, it means that either the Agent has left the mF2C fog-area or it has stopped participating in the mF2C network. Otherwise, if the Agent is participating to the mF2C network, the “status” remains “connected”.

3. API-based resource management
   - Users are given a rich RESTful HTTP API that can be used to perform any kind of CRUD (CREATE, READ, UPDATE, DELETE) operations on system and infrastructure resources. The

API is based on the CIMI specification and it has been extended to support ACL-based authorization. All mF2C operations and resources are managed via this API.

4. User sharing resources
   - The device's user can specify if this device is allowed to run mF2C services and the amount of services that can run on it.
   - TEST: Using the dashboard, the user can access two forms where these properties can be set.

5. Service registration, initialization and execution
   - The registration of services can be done through the GUI of mF2C specifying the name, type of executable, ports, sensors, etc. The initialization and execution of services can also be done through the GUI.
   - TEST: Using a web browser, the user can access the local GUI in localhost. This process has already been tested and verified.

6. Support N layers
   - mF2C supports the deployment and execution of services in a multilayer architecture. A user can launch a service from a child Agent. If the Agent's Lifecycle component does not find enough resources to launch this service, the request is forwarded to the Agent's leader.
   - TEST: At least two Agents connected; one leader, one child. Deployment of a service (with num_agents > 1) from the child Agent

7. Support to Docker Swarm
   - The Lifecycles supports the deployment of services in Docker Swarm environments.
   - TEST: One Agent with Docker Swarm configured is needed. Deployment of a service in this environment.

8. Automatic generation of SLAs from templates
   - The mF2C UI allows a service operator to specify in an SLA Template the QoS of an mF2C service (e.g., executionTime of an operation < 1000ms). When a user instantiates a service, the SLA agreement is created and assessed at runtime.
   - TEST: A template must be created in the UI and associated to a service. Now deploy a service instance and check the SLA agreement on the UI.

9. Assessment of non-COMPSs services
   - During IT-2, availability assessment was added to all types of services. For execution time assessment on non-COMPSs services, the application itself must populate the monitoring data in the database (i.e., CIMI)
   - TEST: See below

10. Detect violations (exec. time and availability)
    - mF2C's SLA Management supports availability (for all types of services) and execution time metrics (for COMPSs services). Availability refers to the availability of the containers that need to be running in order to consider the service available. Execution refers to the time that takes a given operation to complete (e.g., a call to a REST endpoint)
    - TEST: A template must be created in the UI with difficult-to-fulfil thresholds, and associated to a service. Now deploy a service instance. To detect availability violations, manually kill the container running the service and wait for the violation to happen.

Similarly, to detect execution time violations, make the service execute the operation, and wait for the violation happens. Violations can be retrieved in the UI.

11. Classify services for Recommender
    - Every new service is classified upon registration into the system. Since the algorithm for service classification is based on clustering, all services registered in the system are periodically reclassified, updating the category of the service. This category is later used to call the recommender for returning Agent recommendations.
    - TEST: When a new service is registered into the system, the classifier adds an initial category to the service. After adding more services, it can be seen how this category is updated.

12. QoS Providing
    - To determine in which Agents a certain service is going to be launched, first the Recommender returns a lists of Agents to the Lifecycle Manager and then the QoS provider determines which of these Agents are going to be finally used or blocked. This decision is based on a learning algorithm already explained in past deliverables.
    - TEST: Since the QoS provider learns from past executions, this block cannot be quickly tested, but many executions of the same service over a cluster of devices has to be performed. Tests in controlled environments have been successfully performed.

13. QoS Enforcement
    - Once a service instance is running, DER periodically stores service operation reports with the expected duration time of the service into CIMI. When a new report is stored or updated, the Event Manager notifies the QoS enforcement block. Then, the QoS enforcement checks if the expected duration is longer than the one specified in the SLA agreement. If so, it notifies the Lifecycle Manager to add more Agents to the current service execution.
    - TEST: This mechanism has already been tested and properly works for COMPSs services.

14. DER Add/remove Agents from an execution
    - A service instance is deployed on two Agents.
    - TEST: a long-length (e.g.; 20 seconds) operation is submitted to one of the agents hosting the service, indicating as available only the resources within the agent. Once this first operation finishes, the tester submits a second execution of the same operation passing in the same arguments and using only the resources within the same agent. After that, the tester invokes the addResource operation on the agent REST API to indicate that resources within the second agent are available to run the job. The execution should last less time since part of the processing is offloaded onto the other node. After running this second execution, the tester submits a third execution of the operation with the same arguments but indicating as available resources in both agents. The execution time for this third operation should be shorter than the first two. Finally, a fourth execution is submitted with the same parameters, using the resources within both agents. In the middle of the execution, the tester invokes the removeResources operation of the REST API of the agent to remove the resources within the second agent from the resource pool. The execution of this fourth operation should still be successful, but the time should be higher than the previous one (which runs on two agents all the time) but still lower than the execution that uses only one agent.

15. Refine recipe (INTEL)

- The user registers a new service into the system. The Service is deployed on node(s) with the mF2c Agent. Data representing the system's compute capabilities, topology and existing service footprint are stored in the Landscaper. The Recommender, using the information available in the Landscaper, refines the deployment recipe in terms of the resource allocation, comprising the number of CPU cores, Memory and Disk sizes. In some deployment scenarios, refinement of the deployment recipe also considers the availability of certain sensor types which are required by the service being deployed.
- TEST: Recommender learns from past executions. This block cannot be quickly tested, however iterative execution of the same service over a cluster of devices has been performed. Tests in controlled environments have been successfully performed.

mF2C Hello-World

The Hello-World test is implemented trying to test some of the previous described functionalities in mF2C with or without the integration tests. After having the Agent running, the *hello-world.sh* script can be simply executed in order to directly test the functionalities listed above.

3.5. Security Audit

This security section is split into three subsections: a high-level overview of security features; an as-brief-as-possible re-review of the IT-1 audit, and a summary.

3.5.1. High Level Security Features

This section describes the high level achieved – and open – security features of the mF2C platform.

3.5.1.1. Security by Design

☑ Public Key Infrastructure: all Agents/Microagents secured with X.509 certificates
- Agents joining a fog go through a bootstrap process, implemented through a cau-client, to obtain a X.509 certificates. There is a strong case for using PKI for IoT (e.g. [21] section F).

☑ Secure private key principles
- The private key never crosses the wire: the key is generated by the entity that will be using it, and is never held by anything else (cf. [21], section M.9). There is no key escrow (and none is needed.)

☑ Implemented policy-driven security
- A simple but sufficient policy is defined, classifying data as PRIVATE, PROTECTED, and PUBLIC.

☑ Agent security extended to DataClay and CIMI
- The Agent certificate security extends to all parts of the platform.

☑ GDPR compliance
- Support for user-requested deletion; and data protection through the security policies.

☑ Fixed infrastructure secured through long-term certificates
- There is a separate PKI for infrastructure, which can issue long-lived, revocable certificates, and the trust anchors can be pre-distributed with Agents.

☒ Security Incident Event Management (SIEM)
- With the diversity of logs and some components producing extremely verbose logging, it was not feasible to develop a SIEM system.

☒ In-depth security implementation, devsecops
- The principal focus of the project has been IoT research and implementation features, rather than security code reviews and hardening software.

☒ Trust anchor distribution
- The project never implemented a trust anchor distribution protocol, meaning that someone else reusing mF2C would need to reconfigure trust anchors throughout components. However, as mitigation, we are converging towards a single location for trust anchors, thus

making it easier in a re-deployment to implement a distribution mechanism on top of the platform.

3.5.1.2. Connectivity

☑ No internet access for unauthenticated Agents
- An Agent that arrives in a fog will not have internet access by default; it will need to obtain a credential first. Although the CA is running in the cloud, a special gateway called the CAU enables the client to request a certificate.

☑ Additional interconnect through lightweight VPN
- A very lightweight VPN implementation – no python, no bash – provides a secured network interface for both platform (child to leader) and application communications.

☑ Secure infra-container communications
- Communications between containers is implemented on the host (such as through a Docker bridge network.)

☑ Secure inter-container communications
- Communications between containers on different hosts is secured through a reverse proxy (Traefik), and secure VPN.

☑ Policy-aware communications library
- A library called AClib has been implemented to support message handling according to security requirements. The library can be used with multiple protocols (e.g. HTTP/REST, or MQTT.)

☑ Agents and Microagents mutually authenticate
- All Agents and Microagents have certificates and authenticate (client to server, peer to peer) to each other using these certificates. The downside is that general broadcasts are difficult: secured broadcasts will currently need to be relayed through the leader.

☒ Partial – not full – support for horizontal communication
- While the decision was made to carry control messages hierarchically, applications could still communicate horizontally, using the VPN or an external network such as LoRa or 4G. However, they then need to implement their own neighbor/peer discovery.

☒ Elliptic curve cryptography
- mF2C's PKI was never switched from RSA (the most widely supported public key algorithm) to ECC (which is considered more lightweight for limited devices.) However, even Microagent devices are capable of handling RSA operations, so the impact on this is negligible.

☒ Hardware assisted edge crypto
- It was intended to use hardware assisted crypto at the edge (e.g. elliptic curve chips, giving even an Arduino the ability to handle PKI), or Microsoft's Azure Sphere. However, this was not a high priority and had to be postponed.

3.5.1.3. Users of applications running on the mF2C platform

☑ No plaintext passwords
- No passwords pass across the wire without being end-to-end encrypted.

☑ Token-based authorisation/delegation (using JWT)
- There is support for handling fine-grained authorisations and delegations of rights through JSON Web Tokens (JWT). However, this is not implemented throughout the platform.

☒ No external identity providers in the portal.
- For some use cases (UC2, UC3, or people reusing mF2C), it would improve usability to let users bring their own identities (Facebook, Google, Microsoft Live), instead of the old-fashioned username/password registration. However, username/password is better than nothing, and people are used to it, so most applications can live with the limitation.

3.5.1.4. Use Cases

☑ EMMY

- The emmy library provides an implementation of a cutting-edge security schema, enabling users to authenticate without revealing their secrets (such as a password).

☑ Use case-specific PKI

- The service in the cloud that hosts the CA for the mF2C infrastructure and Agents can also host PKIs for use cases, thus enabling them to have distinct policies and trust anchors, and to decouple the PKIs from each other for added protection in case of compromises.

☒ Tamper proof edge hardware

- It was considered to investigate tamper evident edge hardware – as a proof of concept, or at least use a security-hardened microcontroller, but we did not have enough time – it would have required a port of much of the mF2C code, and dropping all Java and python components.

☒ CA as a service

- Providing CA as a service would make it easier to run use-case specific PKIs, but in the end, the extra convenience was not worth the significant effort. A redeployment of mF2C might decide to bring their own external CA.

3.5.2.Review of Security Audit

For IT-1 we conducted a rather extensive security audit, including penetration testing (*pentest*) of all relevant components. The results of this audit can be seen in Appendices 2, 3, 4, 5, 6, 7, and 8 of D5.1 [2]. This work is still extremely useful, as we can now save time and effort by updating the previous work. Appendices 7 (security architecture) and 8 (privacy) were both discussed in D2.5 [8] and need not be covered here.

It would not be possible to repeat all these tests for IT-2; to do so would require a 2-3-month extension to the project, through to at least February 2020, to set up a separate testbed, deploy the software, run the tests, and write up a report. However, it is still worth comparing the present state with the state found in the IT-1 test, focusing on the risks that were identified as most severe.

Vulnerability	Reference (D5.1 appendix)	Mitigation	Result and residual risk
MQTT open for unauthenticated clients	2	MQTT is not used in the standard platform. AClib available to encrypt messages	If MQTT is used, AClib should be used for all messages. Clients should be configured to discard unsigned messages.
Denial of service	3	Monitoring is now better than in IT-1. Client authentication required for more endpoints.	Still a general vulnerability for services that do not require authenticated clients.
Vulnerabilities in Docker network setup	3	Limited success in securing Traefik (#282).	There is still a mix of bridged, host, VPN, and proxied networking, which could lead

Vulnerability	Reference (D5.1 appendix)	Mitigation	Result and residual risk
			to unintentionally exposed endpoints
Check for network misconfigurations	3	The new potential weakness is probably reliance on wifi. VPN requires Agent keys for authentication and key agreement, so is less likely to be misconfigured.	As previous
Outbound connectivity from compromised component	3	This is potentially still an issue for some components	Known risk
Compromise of neighbour containers and Host protection of Docker daemon	3	A new potential vulnerability has been introduced in IT-2 through exporting the daemon socket to the container – for a limited set of containers	Managed risk
Use of environment variables	3	The use of passwords and other secrets in the config has been much reduced.	Reduced risk
DataClay	3	Security has been added to dataClay	Risk eliminated

Table 10. Vulnerability comparison table with respect to IT-1

Appendix 4 relates to logs and parsing logs. The main risk is that most components generate a huge amount of log output continuously, as seen when running an Agent in the foreground, thus risking that log events related to issues being lost. Additionally, we have not had time/priority to develop a SIEM system.

A secondary risk identified in section 4 is about the physical security of the edge devices. Again, to an extent, it is out of the scope of the platform itself, and more related to the use cases (see D4.1 [9]).

Appendix 5 relates to business level security, i.e. backups and recovery, and GDPR. It is noted that the compliance with GDPR is improved; with defined privacy policies and support for account deletion. Backups of the main service is out of scope; for the essential services like CAs, etc., these can be provided on cloud-hosted resources with as much durability as desired. The risks identified in this section are fully controlled.

Appendix 6 revisited the STRIDE vulnerabilities, originally discussed in detail in D2.4 [10], section 2.1.1. The risk of Spoofing is much reduced, thanks to the device ID and certificate being used more consistently on the platform level. The risk of Tampering remains low, although no cryptographic integrity checking is implemented for data at rest. Repudiation is mainly a question of logging user

actions, as identified in D5.1 [2], but remains a low risk. Information disclosure is like Tampering, with the highest protection in flight; AClib is available for clients requiring tamper or disclosure protection for stored data, so again the risk can be managed. Denial of service is a risk as discussed above; the main mitigation is that we now have more endpoints requiring client authentication. Finally, Elevation of privilege remains a low risk, with the main residual risk through the volumes and access to the Docker socket.

3.5.3. Summary of Open Issues

Since mF2C is a research project, there was never time to implement a full DevSecOps cycle (as opposed to the successfully implemented DevOps cycles, agile development, sprints). A full DevSecOps scenario would include code review for security, checking that all errors are handled, managing the dependency risks of external components and libraries. Thus, with enough resources, one of the most beneficial activities would be a code review.

3.6. Use Cases Validation

In order to validate the technical functionality and implementation of the mF2C Agent and Microagent in IT-2, the mF2C Use Cases have separately conducted their own tests and evaluation. Their aim was to verify that the implemented features provided enough coverage to fulfil their requirements.

Use case 1

UC1 (Emergency Situation Management - EMS) for IT-2 is an evolution from the use case developed for IT-1 and described in D5.1 - mF2C reference architecture (integration IT-1) in M16. The evolution of this use case has allowed the project to optimise the main services for IT-1 (decision making according to an inclination sensor and EMS in a Smart City context. The service still triggers the intervention of the relevant emergency services but in an optimised manner. The services now consist of five functionalities which cannot be demonstrated using only the Fog or the Cloud versions but require the mF2C coordination to perform all services optimally. These services are:

1) Monitoring
2) Detection
3) Positioning
4) Scanning
5) Intervention.

The architecture has also been modified to include more layers and Agents. The storyline starts with behind the scene actions such as the indoor positioning of the workers. Their position is calculated and stored in an anonymised way in the location concentrator. When the tiltmeter reports an incident, the EMS tool calculates which is the closest worker and sends a message to the industrial application for the worker to locate the incident and either validate the event or report a false positive on the monitoring software. If validated, the emergency is displayed on the visualisation software, the local intervention is initiated by starting a siren and a beacon, whilst the global emergency intervention process is initiated by contacting the relevant emergency systems, calculating the optimum route and changing the traffic lights on the way for a faster resolution of the incident. The positions of the workers are not reported on the visualisation software to ensure GDPR compliance. The different components are described, together with their interaction, in the following figure:

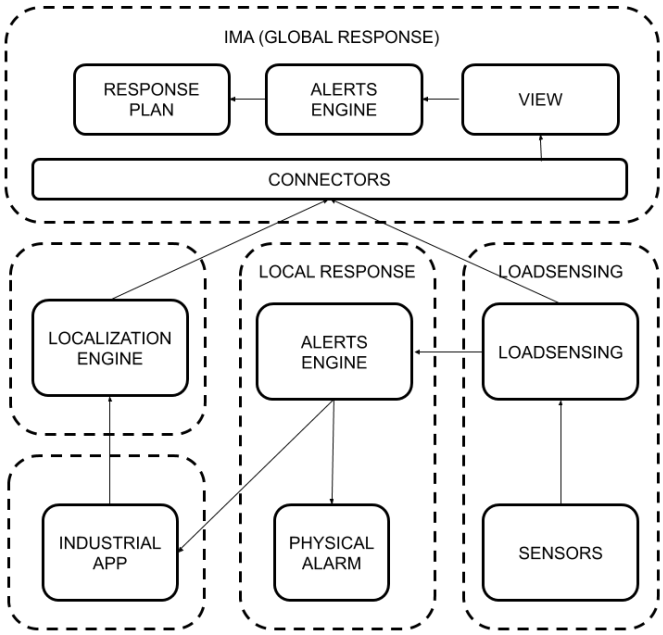

Figure 6. Technical components of the EMS service for IT-2 with their interaction

The tiltmeter sensor that reports the incident when there is an inclination of the building beyond a pre-defined threshold is the Loadsensing sensor. The Loadsensing system (meaning the Loadsensing communication system) transmits the incident to the Industrial Management application (IMA) software, that includes the visualisation and part of the monitoring functionality, as well as the global alerts engine and response plans. Loadsensing is a commercial suite of devices available from the Worldsensing catalogue. The incident is also reported directly to the local response alerts engine, which reports to the industrial app and, if necessary, activates the local response plan including the physical alarms. The industrial app feeds on the localisation engine that stores the position of the workers to act as the interface with the workers, in which they can validate or cancel the existence of the emergency.

A more detailed description of the use case and its validation is available in D5.4 - mF2C solution demonstration and field trial results (validation IT-2). This use case has been used to validate the Agent capabilities and performance. The results of these tests have been used to make improvements both on the Agent and on the use case.

Use case 2

UC2, Smart Boat, has continuously validated the Agent's capabilities through the deployment of the SmartBoat platform, both on the regular Agent and on the Microagent. Fixes and improvements were provided through direct feedback to core mF2C platform developers. Throughout the development and integration efforts of IT-2, this has shown to be valuable information for constructing a working system.

Specific testing was also executed by experienced developers with no prior knowledge of, or interaction with, the mF2C platform. The purpose of these tests was to obtain pure, tabula rasa feedback on the experience of the developer. Testers were provided with the mF2C documentation as a starting point and were left to their own devices to achieve a self-imposed list of tasks using features described in the documentation. Any hard blockers were discussed with developers familiar with the inner workings of mF2C, all the while noting down inconsistencies and other failure points of both the documentation and general usability of the mF2C platform.

A report was produced by the testers that contained a comprehensive list of suggestions, notes and detailed error cases. Through it, the mF2C development consortium then took actions to rectify the

inconsistencies, fix bugs, improve usability and, most importantly, improve documentation such that new developers would have a better first experience when using and integrating their application with the mF2C platform.

Use case 3

The UC3 progressed in the evolution of features, using an agile approach to software development. The continuous deployment of newer software releases in the internal testbed and finally in the airport testbed enabled to check the correct functioning of all features and interaction with the mF2C agent.

A contribution to the integration tests sessions have been provided with a dedicated developer that produced a detailed report for each test case.

To validate the performance of the mF2C platform within the Use case 3, the following performance measures have been defined:

- **Latency:** measured from the smartphone to fog and cloud devices
- **Response Time**: the time measured at the client premises, from the time of the request to the reception of the answer.

A laptop with wi-fi connection has been chosen as the client, and Jmeter has been used to launch batteries of increasing numbers of simultaneous client proximity requests per second to the server, corresponding to real world scenarios, collecting data on response time under different loads.

The following tests have been defined:

- **Smartphone to Fog** (PCbox)
- **Smartphone to Cloud** (OpenStack)
- **Smartphone to Fog2Cloud** (PCbox & OpenStack)

In terms of metrics, the response time for every request is collected. In the third scenario, the number of requests processed at the fog level are summed separately, thus determining the percentage of requests processed at fog level against the total.

In the "**Smartphone to Fog2Cloud"** test the first run has been run with only one fog available node (PCbox), while in the second run an additional PCbox has been included.

Results

The picture of Figure 3 shows the average response times collected in the various runs.

The "**Smartphone to Fog"** test performs well with low number of requests, but as the number of requests grows, the response time increases significantly, not fulfilling the real-time requirement.

The "**Smartphone to Cloud"** shows a quite stable performance that reflects the base latency between the peer nodes. So, with the increase of the number of requests, the performance is better than the "**Smartphone to Fog"** test**.**

The "**Smartphone to Fog2Cloud"** shows the best of the two previous tests: it gets the advantage of the runtime distribution capabilities that uses the fog nodes with small number of requests, and for larger numbers distributes between both layers, thus maintaining a good real-time response time. With 10000 samples, we measured an improvement of about 25% compared to the "**Smartphone to Cloud"** approach**.**

The same test with an additional fog nodes gives an even better performance, using more the fog resources and distributing to the cloud only with higher number of requests. An additional 15% of improvement has been tracked, with a total improvement of about 40% vs the Cloud-only approach.

In the following Figure the performance under all settings are shown, the X-axis represents the number of concurrent proximity requests run, while the Y-axis represents the response time expressed in milliseconds.

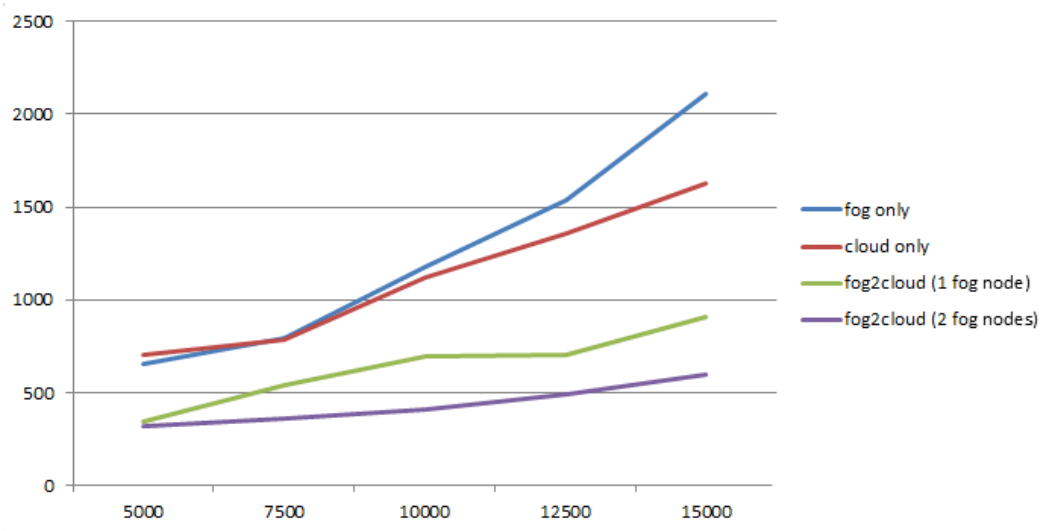

Figure 3. Performance comparison between different settings

The following table shows the latencies of fog and cloud from smartphones.

Node element	Latency (avg)
Fog node (miniPC)	<1 msec
Cloud node (OpenStack)	35 msec

The Figure 4 summarizes the average response times and variance in the four test cases with different number of samples.

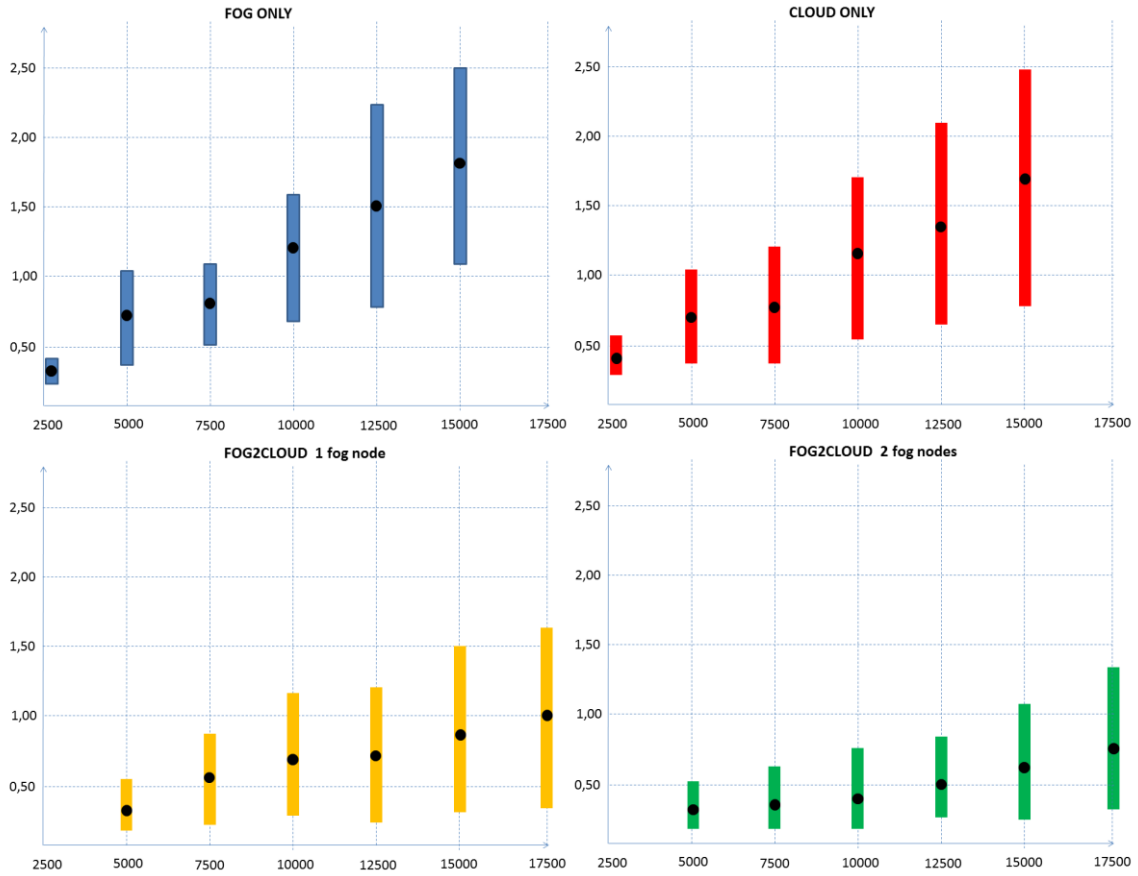

Figure 4. Response times and variance in the four settings

The described tests validated the **Fog2Cloud** setting: the intelligence embedded in the mF2C Agent runtime enable the proficient distribution of processing optimizing the response time, even under severe processing conditions. Adding more nodes near the edge has proved to be a simple way to scale the system, keeping processing near the edge and saving latency time to access the cloud, and only off-load part of the processing as the fog nodes get overloaded.

4. Scalability

The purpose of this section is to evaluate the scalability of the mF2C platform in terms of the number of devices participating in an mF2C deployment. First, we introduce the evaluation we conducted, focusing on those mF2C architectural components that are potential source of bottlenecks either in the management of the platform and its devices or in the execution of services, and detail the setting where the experiments were run. Thus, subsection 4.1 focuses on the scalability analysis from the point of view of the management of the platform, evaluating how the number of devices to be synchronized affects the performance of requests from the mF2C components. Then, Subsection 4.2 describes the evaluation of the scalability from the perspective of the execution of a service, analysing how the number of devices to be coordinated affects the performance of a service that runs on top of mF2C.

To demonstrate the scalability in terms of number of devices participating in the mF2C platform we will focus on those mF2C components that may cause bottlenecks as the number of devices increases.

In a distributed platform such as mF2C, the main bottleneck that limits the scalability are communication interfaces, which with an increasing number of end devices may overload the network and drastically reduce performance of the system. In other words, individual system components may become blocked due to communication delays.

To address this bottleneck, in mF2C most components operate locally in an agent, where they can access the data they require without the need to communicating with other agents across the networks. This is possible because the Data Management component is in charge of replicating and synchronizing the required data between agents. However, this beneficial characteristic does not remove potential scalability issues to come up, rather it shifts them to the synchronization and replication features within the Data management component, thus making this component to become a potential bottleneck.

Another functional component where communications between agents cannot be avoided is the distributed execution of services or applications. Here, the Distributed Execution Runtime (DER) needs to coordinate and send execution requests to the different devices that have been selected for the execution.

Thus, the evaluation of the scalability of mF2C can be reduced to evaluate the scalability of both the Data Management and the Distributed Execution Runtime components, which are the potential bottlenecks either in the management of the platform or in the execution of applications or services, respectively. More precisely, it is the dataClay object store and the COMPSs runtime, at the core of these components, that are the ones in charge of the interactions between the agents in an mF2C cluster.

In order to evaluate the scalability of these components, we have performed the tests in an environment where network is not the bottleneck, and we found that these components themselves scale. In a real environment, the effective scalability that can be achieved will depend on the quality (performance and stability) of the available network infrastructure, and on the computing capacity of the devices deployed, which will vary in each particular setting. Thus, although the components have been successfully tested in regular devices such as laptops, Raspberry Pi and Nvidia Jetson boards, the scalability analysis has been performed in the MareNostrum 4 supercomputer hosted by the Barcelona Supercomputing Center, with high throughput and low latency communications. This will highlight the possible scalability problems in the implementation of the components.

The MareNostrum 4 consists of 3.456 nodes, each of them equipped with 48 cores. This infrastructure is made available to European scientists as a facility belonging to different supercomputing networks, so the resources that can be used in an individual experiment are limited according to a set of policies.

We were able to use at most 33 nodes to run the experiments, each of them simulating a device, which should allow us to detect scalability problems by stressing the system at a higher degree than what would happen in the normal operation of an mF2C infrastructure. We ran several experiments where one of the nodes plays the role of leader, and the rest of nodes are their children, from 2 to 32.

It is worth noticing that the number of nodes where the experiment is executed does not represent the total number of agents in an mF2C deployment, but the degree of concurrency within a single mF2C cluster, that is, the number of concurrent synchronization requests in the case of Data Management, and the number of agents concurrently executing a service in the case of the Distributed Execution Runtime, in both cases managed by a single leader. Thanks to the mF2C hierarchical architecture, designed for scalability, an mF2C deployment can consist of several such clusters, so the results of the experiments proves the scalability of a deployment where each leader, in any layer, handles a maximum concurrency degree of 32.

4.1. Scalability in management

As explained above, the potential bottleneck in the management of mF2C is the synchronization of data between Agents, performed by dataClay within the Data Management component. We have developed a benchmark that evaluates this scenario, with a leader and an increasing set of children (from 2 to 32, each of them in a different node) that are updating data simultaneously, which is the worst-case scenario regarding synchronization of data in mF2C. More precisely, each child updates a piece of data that it holds, and the update is synchronized to the leader, which holds data from all the children according to the mF2C architecture. Recall that the children represent the concurrency degree, and not the total number of children in an mF2C cluster.

This benchmark already covers the potentially problematic situations regarding synchronization, since data is only synchronized bottom-up and each agent has at most one parent. Thus, increasing the number of layers in the mF2C hierarchy would not negatively affect scalability.

The test consists in each children performing 500,000 updates on a piece of data stored locally. Each of these updates is synchronized to the leader, who receives the simultaneous requests from all the children and updates its local data accordingly. We show the results of this evaluation in Figure 7, for 2, 4, 8, 16, and 32 children updating data at the same time. For each setting, we depict the time of a single update (in nanoseconds), calculated as the mean of all the updates executed, which includes the time of the update in the child itself plus the time spent in the synchronization with the leader.

As can be seen in the figure, when using all the available resources in each node (i.e. 48 CPU cores), a concurrency degree of 32 is not enough to stress the system and the time to synchronize remains almost constant (between 0.117 and 0.123 milliseconds), independent of the number of concurrent requests. This shows that the Data Management component is able to handle all the requests simultaneously, taking full advantage of the available resources in the device.

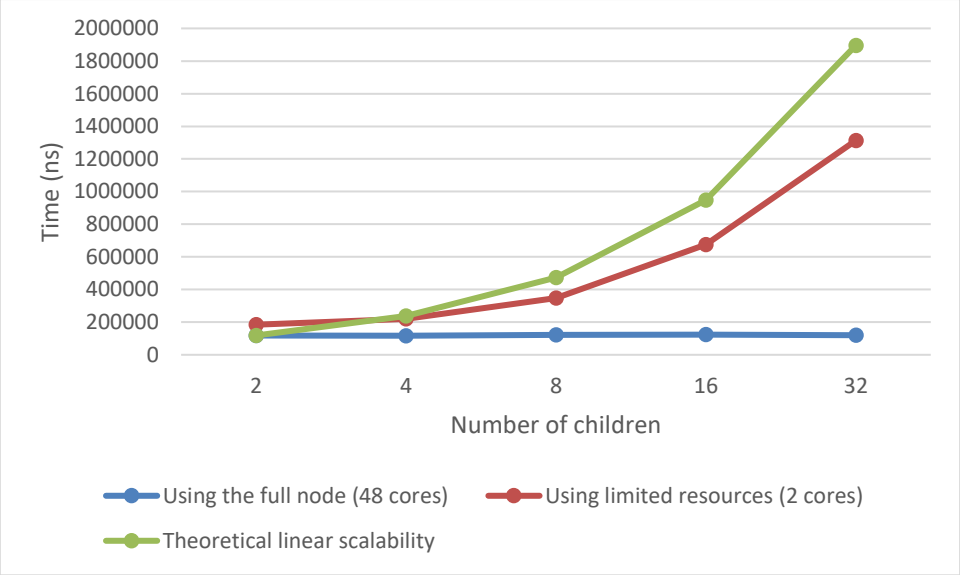

Figure 7. Scalability of the Data Management

To simulate a scenario with more limited resources and try to make possible problems show up, we reduced the cores to be used in each node to 2 (less than the number of cores in a Raspberry Pi), which implies that the leader will be able to handle at most 2 simultaneous requests, due to the limited capacity of the device. In this case, we can see that the synchronization scales better than linearly according to the available resources (see the "Theoretical linear scalability" line for comparison, which represents the perfectly linear behaviour, starting from the first synchronization time obtained): doubling the number of requests almost doubles synchronization time, not causing additional delays that may end up generating a bottleneck. Thus, the potential bottleneck caused by synchronization of data has not shown up with the maximum degree of concurrency that we were able to test, and there is no indication that this would happen with a higher degree of concurrency.

In light of the good scalability results shown, if the expected performance is not met with the available resources in an mF2C deployment, the administrator should either scale the system vertically by increasing the capacity of the leader device so that it can handle the load of the system, or scale horizontally by organizing the Agents in smaller clusters, so that the load of each leader is reduced.

4.2. Scalability in execution

Regarding execution, the potential bottleneck is the distribution of tasks among the devices, performed by COMPSs as part of the Distributed Execution Runtime (DER) component.

To test the scalability of mF2C services developed using the DER, we executed a machine learning algorithm to create a classification model: Random Forest. The purpose of such a model is to return the probabilities for a new observation to correspond to an observation of an element belonging to a set of predefined classes. For achieving such a goal, the model consists of a set of estimators each of which returns the probability of belonging to each class given an input observation. The prediction of a Random Forest model for a new observation corresponds to the aggregated result of all the estimators belonging to such model. Therefore, the training of the model only consists of creating N estimators. Each estimator is created by reading a training dataset and generating a decision tree based on the feature values within that data set; thus, the creation of each tree is totally independent from the creation of other estimators. In order to avoid overfitting, each estimator is trained using a random combination of samples of the dataset.

For distributing the training of the Random Forest model, the application generates two tasks for each estimator. The first one, generateRandomSelection, creates the combination of the samples within the dataset that will be used as the input training dataset for the estimator; and the second one,

createEstimator, that actually generates the estimator using the output of the first task. Figure 8 illustrates with a directed acyclic graph the dependencies at task level that exist during the training with 5 estimators. Blue and red nodes respectively represent generateRandomSelection and createEstimator tasks; and the graph edges depict data dependencies where the start of the edge is the node generating the data value and the end of the arrow the task consuming such value.

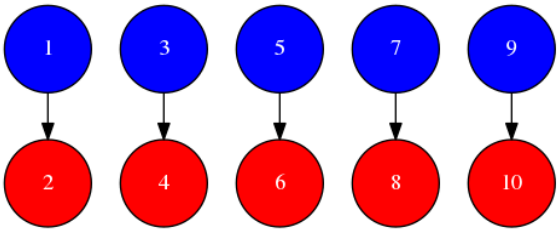

Figure 8. Task-dependency graph task generated by the Random Forest model training with 10 estimators

The chart in Figure 8 shows the execution time needed to train a Random Forest model with a variable number of estimators (from 1 to 4096) that classify observations among 200 classes. The training dataset used for the test is randomly generated for each execution and contains 30,000 samples with 40 features each. To run the tests, each available node in the MareNostrum 4 holds a single DER Agent deployed on it that oversees executing tasks on the node or offloading them onto other Agents of the infrastructure. On this experiment, the computation of the model training is distributed among 2, 4, 8 16, and 32 Agents (respectively, 96, 192, 384, 768, and 1536 CPU cores).

Observing the chart, we can see that the more agents are used, the lower the observed execution time is for the same number of estimators in the Random Forest model. When doubling the resources from 2 to 4 agents, the application achieves a 1.8x speed up; from 4 to 8, the speed up reaches 1.75x; and from 8 to 16, up to 1.71x. When using 2 and 4 agents, we can see that the execution time increases linearly according to the number of estimators. However, with 8 agents, the behaviour changes due to two issues. The first one, load imbalance, is originated by the application nature and the number of estimators generated. The more resources are used, the less workload is assigned to each resource. Since tasks are coarse grained, the indivisible computation unit is the training of one estimator (about 6 seconds), when larger infrastructures are used, the more common it is to have idle resources when no more estimators need to be computed. The second observed problem is the time needed to schedule the tasks. When 4000 estimators are generated, the application is composed of 8000 tasks; and the scheduler needs to evaluate on which of the available resources is better to run each task. Thus, when only 2 agents are used, the scheduler needs to consider 16000 different options; when 16 agents are used, 128000. At some point, the time needed to decide which tasks that will be executing at a specific time is larger than the actual time to process the task; and the scheduling time becomes the bottleneck of the execution. Notice that this problem is related to the execution conditions. When the application runs on top of an Edge infrastructure, both the network and the computing power are lower. Therefore, tasks take longer to execute and the problem needs to have a higher number of tasks to impact significantly on the performance of the application.

Another solution to reduce the impact of the scheduling on the performance is to cluster several Agents and selecting one of them as a representative which might run the task on its resources or forward it to another Agent within the cluster. This technique can be applied recursively generating an N-level infrastructure and the number of resources that the scheduler needs to consider is drastically reduced on each level. Figure 9 shows the impact of these technique on the 32 Agents (1536 cores) case, where instead of organizing the workers in a master-worker schema, 3 layers are defined where each Agent directly manages up to 8 Agents. Although 256000 options should be considered to plan the execution, the application speed-up is similar to the one obtained on the 8 Agents case.

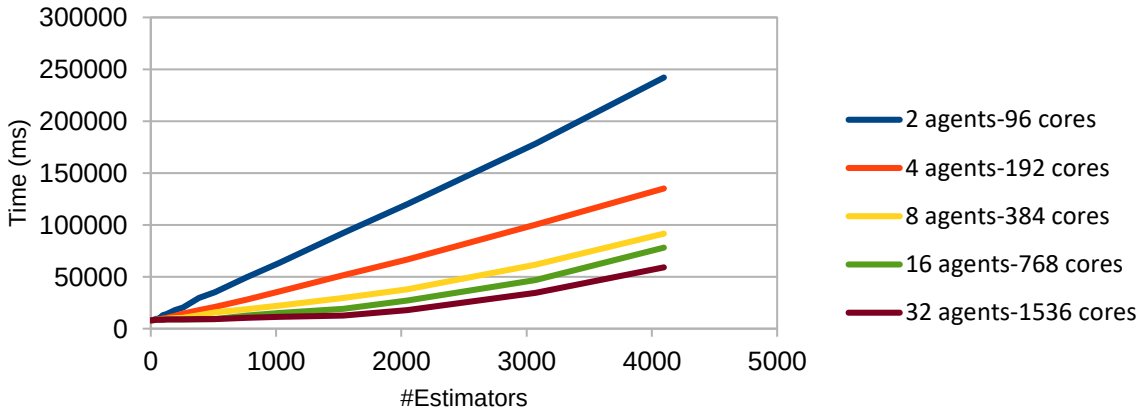

Figure 9. Scalability of the Distributed Execution Runtime

Summarizing, the scalability analysis shows that some issues start to be noticed in a high-performance infrastructure with 8 agents selected for an execution, although performance is not yet affected. Interestingly, in an edge to cloud infrastructure with more limited capabilities, the detected scalability problem will have a smaller impact and, thus, more agents would be needed before reaching a performance degradation. In any case, the problem is solved by the DER by clustering agents in order to keep the potential scheduling combinations under control, as shown for the case of 32 agents, to avoid decreasing application performance. Also, it is worth noticing that the application selected for the benchmark stresses the potential bottlenecks, and other kinds of applications would not be affected by this problem, thus being able to keep scaling linearly, as happens in this test with 2 and 4 agents.

5. Conclusions

In its final form, the IT-2 mF2C prototype has been exhaustively tested from several ends:

- The functionality tests show that the designed architecture is able to provide the expected behaviour according to the functionality workflows presented in past deliverables.
- The security update shows that many of the issues identified in the extensive IT-1 security assessment have been addressed. In most cases, there is a reduced risk which can be managed through the introduction of additional controls. In a few cases, new risks have been introduced. Overall, the project platform provides an extensive set of security features, primarily focused on individually keyed Agents with credentials from a cloud-based certification authority. This security underpins most of the platform-based guarantees. Additional work provides much improved support for user privacy and GDPR.
- While staying in line with their requirements, the three mF2C Use Cases have joined the testing efforts, providing unique feedback from the users' perspective, concerning the mF2C Agent's performance, installation procedures and general user experience. A more detailed description of the use case and its validation will be available in D5.4.
- The integration tests provide continuous and generic mechanisms to test the inter-component dependency, thus contributing to the code integrity throughout future development. Not all the integration tests have successfully passed, and this is a good indicator of the improvements that can still be done.
- The scalability tests have shown that due to its loosely coupled architecture, mF2C is able to scale alongside its core components, DataClay being the most important one, due to its inter-agent synchronization mechanisms. Here, good results for scalability have been demonstrated. The physical resources and physical networking capability are some of the topics to be considered by the mF2C user and/or provider, when scaling the platform.

All the user-specific material (documentation) is summarized in this document whilst it is being further developed for public online documentation.

In summary, the final IT-2 PoC implementation meets the expected design. Its features can fulfil the use cases' demands. There is however room for improvement, not only for assuring a better and more consolidated integration of the components during execution, but also to better automate the full lifecycle of the mF2C Agent.

References

[1] mF2C, “mF2C Project,” [Online]. Available: http://www.mf2c-project.eu/.

[2] mF2C , “D5.1 mF2C reference architecture (integration IT-1),” [Online]. Available: https://www.mf2c-project.eu/d5-1-mf2c-reference-architecture-integration-it-1/.

[3] mF2C, “D2.7 mF2C Architecture (IT-2),” [Online]. Available: https://www.mf2c-project.eu/d2-7-mf2c-architecture-it-2/.

[4] mF2C, “D4.4 Design of the mF2C Platform Manager block components and microagents (IT-2),” [Online]. Available: https://www.mf2c-project.eu/d4-4-design-of-the-mf2c-platform-manager-block-components-and-microagents-it-2/.

[5] mF2C, “D4.6 Platform Manager blocks and microagents integration (IT-2),” [Online]. Available: https://www.mf2c-project.eu/d4-6-mf2c-platform-manager-blocks-and-microagents-integration-it-2/.

[6] mF2C, “D3.4 Design of the mF2C Controller Block (IT-2),” [Online]. Available: https://www.mf2c-project.eu/d3-4-design-of-the-mf2c-agent-controller-block-it-2/.

[7] mF2C, “D3.6 mF2C Agent Controller block integration (IT-2),” [Online]. Available: https://www.mf2c-project.eu/d3-6-mf2c-agent-controller-block-integration-it-2/.

[8] mF2C, “D2.5 mF2C Security/Privacy Requirements and Features (IT-2),” [Online]. Available: https://www.mf2c-project.eu/d2-5-mf2c-security-privacy-requirements-and-features-it-2/.

[9] mF2C, “D4.1 Security and privacy aspects for the mF2C Gearbox block (IT-1),” [Online]. Available: https://www.mf2c-project.eu/d4-1-m6/.

[10] mF2C, “D2.4 mF2C Security/Privacy Requirements and Features,” 2017. [Online]. Available: http://www.mf2c-project.eu/d2-4-m4/.

[11] m. Consortium, “D5.1 mF2C reference architecture (integration IT-1),” [Online]. Available: https://www.mf2c-project.eu/d5-1-mf2c-reference-architecture-integration-it-1/.