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

D5.1 mF2C reference architecture (integration IT-1)

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:	*M16*
Submission Date:	*11/05/2018*
Version:	*1.4*
Status	*Final*
Author(s):	*Román Sosa (ATOS), Roi Sucasas (ATOS), Anna Queralt (BSC), Daniele Lezzi (BSC), Antonio Salis (ENG), Alexander Leckey (INTEL), Cristovao Cordeiro (SIXSQ), Jens Jensen (STFC), Cheney Ketley (STFC), Shirley Crompton (STFC), Franscisco Carpio (TUBS), Eva Tordera (UPC), Xavi Masip (UPC), Sergi Sànchez (UPC), Jordi Garcia (UPC), Ester Simó (UPC), Alejandro Jurnet (UPC), Zeineb Rejiba (UPC), Souvik Sengupta (UPC), Alejandro Gómez (UPC), Denis Guilhot (WOS), Alejandro Lampropulos (WOS), Matija Cankar (XLAB)*
Reviewer(s)	*Jens Jensen (STFC), Admela Jukan (TUBS)*

Keywords
Implementation, PoC, demonstration, integration

Project co-funded by the European Commission within the Seventh Framework 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 project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730929. Any dissemination of results here presented reflects only the consortium view. The Research Executive Agency is not responsible for any use that may be made of the information it contains.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	21/03/2018	ToC and assignments	Cristovao Cordeiro (SIXSQ)
0.2	22/03/2018	Preliminary considerations for section 3	Jens Jensen (STFC)
0.3	11/04/2018	Section 4 restructured	All
0.4	17/04/2018	Initial inputs	All
0.5	18/04/2018	Further contributions	All
0.6	22/04/2018	First workflows added	All
0.7	23/04/2018	Missing workflows added	All
0.8	24/04/2018	UCs and security tests added	All
0.9	27/04/2018	Missing workflows added	All
1.0	01/05/2018	First integrated version	All
1.1	03/05/2018	Internal review	Admela Jukan (TUBS), Jens Jensen (STFC)
1.2	04/05/2018	Review and initial quality check	Lara López (ATOS)
1.3	08/05/2018	Comments addressed	All
1.4	11/05/2018	Further refinement and final quality check	Lara López (ATOS)

Table of Contents

List of figures

List of tables

Executive Summary

This deliverable presents interaction between the mF2C components in IT-1, their functionalities and relations, in order to provide the first version of the project reference architecture. The main focus of the document is to describe the functionalities that will be supported by the prototype in IT-1, the integration process to build a prototype and the tests used to validate the prototype.
The technical workflows defined in past deliverables will be reviewed and updated according to their role in IT-1. The infrastructures to be used as testbeds for IT-1 will also be proposed, together with the main functionality demonstrations and use cases' applications.
The outcome of this document is a description of the mF2C prototype that will be used by the use cases to demonstrate the mF2C approach.

1. Introduction

This document presents the integration of the mF2C [1] components for IT-1 to provide the first prototype of the mF2C platform.

The deliverable *D2.6 mF2C Architecture (IT-1)* [2] presented the initial architecture for the IT-1, while each of the architecture blocks were described in more detail in deliverables *D3.3 Design of the mF2C Controller Block IT-1* [3] and *D4.3 Design of the mF2C Platform Manager block components and microagents* IT-1 [4]. The architecture has been refined and updated since the submission of these deliverables. Therefore, this document also reports on the modifications of functionalities supported by the mF2C blocks for IT-1.

This document is structured as follows:

- Section 2 shows the list of updated functionalities for IT-1 and the ones planned for IT-2.
- Section 3 presents the project workflows update.
- Section 4 introduces the first mF2C prototype and the use cases to be used for validating the platform.
- Section 5 describes the testbeds where the mF2C prototype has been installed and the type of tests that have been performed to validate the platform.
- The appendices show the results of the tests described in section 5.

1.1 Purpose

The objective of this deliverable is to describe the first integrated prototype of the mF2C system, the process used to validate the prototype and its results.

1.2 Glossary of Acronyms

Acronym	Definition
API	Application Programming Interface
BT	BlueTooth
BW	Band Width
CA	Certification Authority
CAU	Control Area Unit
CIMI	Cloud Infrastructure Management Interface
CSR	Certificate Signing Request
GPS	Global Positioning System
IoT	Internet of Things
IR	Infra Red
JSON	JavaScript Object Notation
IT	Iteration
PoC	Proof of Concept
PoI	Point of Interest
QoS	Quality of Service
REST	REpresentational State Transfer
SLA	Service Level Agreement
UC	Use Case

Table 1. Acronyms

2. IT-1 Scope

For IT-1, the main goal is to demonstrate the feasibility of the foreseen components and functionalities of the mF2C system. The technical workflows and overall architecture defined in a previous deliverable (D2.6) are to be demonstrated and validated to some extent during this phase.

As a collaborative project, where many programmatic components are being developed from scratch, some functionalities have taken priority while others have either been discontinued after deeper analysis or postponed to IT-2. The following subsections will address which components belong to IT-1, which ones have been left out, and some of the technical decisions that have been taken in order to ease the development and integration process.

2.1 Obsolete and IT-2-only Components

Several components are now out of scope or now obsolete from the original design. The components identified as out of scope are being postponed to IT-2. These are highlighted in Figure 1.

The table below includes new functionalities that we've defined lately, but which might only be implemented for IT-2, such as the Event service. In some cases, entire components have been postponed, in other cases only previously proposed functionality of a component. In some cases where a component has been considered as obsolete, as replacement or alternative component has been introduced.

Module	***Component***	***Description***
Telemetry Monitoring	Intelligent Instrumentation	Module to analyze metrics output to allow for throttling publishing frequencies depending on e.g., anomaly detection, battery degradation, etc.
	Distributed Query Engine	Module to allow the query engine to provide a single API that abstracts access to multiple locations of metrics published. A single location will be used for IT-1 and extended for IT-2
	Event Service	An event queue used by each of the modules to publish/subscribe to events, e.g., service deployed, device added/removed, etc.
Service Orchestration	SLA Management	The automatic creation of an SLA agreement (probably based on templates) has been postponed to IT-2

Table 2. Platform Manager

Module	***Component***	***Description***
Resource Manage	Data Manager	Obsoleted. To be used to enable access to data from sensors from any node where the application using the component runs. DataClay will now manage this functionality. DataClay is already in IT-1.
Service Manager	Services Runtime	Obsolete. This has been replaced by the COMPSs engine
	Allocation	Allocates available resources to requests to meet security and privacy rules, cost models, guaranteeing overall optimal resources

	Mapping	usage. Sends a task to the categorization component to categorize it based on the attributes from the Lifecycle Manager (e.g., memory, network, CPU, storage, time Limit, priority)
User Management	Assessment	Component responsible for checking that the mF2C applications "respect" the sharing model and the profile properties defined by the device's user

Table 3. Agent Controller

2.2 IT-1 Core Components

Some IT-1 components have been designated as “core components” due to their crucial role in the validation of the main functionality workflows from WP3 and WP4.

The main mF2C interface and entry point for mF2C users will be the Cloud Infrastructure Management Interface (CIMI). This component will also provide every other internal component with the interaction layer for DataClay and the mF2C storage backend.

Both CIMI and DataClay will be deployed on every mF2C agent, alongside all the other active components highlighted in Figure 1. Besides CIMI, which obviously needs to be reachable from outside the device, the Lifecycle Manager service will also be exposed and reachable over the network, so that the Lifecycle Manager components from multiple mF2C agents can communicate directly with each other. Traefik [5] will be used as an auxiliary core component for doing the reverse proxying amongst the different blocks that need to be exposed over the network.

All components that are expected to interact with other components are equipped with a REST API which, unless intentionally exposed, will only be reachable by other blocks within the same local network inside the device.

The Service Orchestration block will offer the capability to deploy service instances (applications) through the Lifecycle Manager (with allocation assistance from the Recommender), to aggregate and display the infrastructure resources through the Landscaper and to create and validate service level agreements (SLAs), through the SLA Manager. The Distributed Execution Runtime will mostly be composed by COMPSs and will provide the execution of Java applications in a distributed environment (namely, the mF2C infrastructure). Finally, the Analytics component from the Telemetry Monitoring block will analyze monitoring data to assist the Recommender.

On the Agent Controller level, the Resource Manager block will be responsible for the mF2C agent start-up, registration and authentication, through the Discovery and Identification components. The Categorization component will then gather a dataset of static and dynamic information on the underlying device, its attached sensors, and the surrounding fog area. The Service Manager block will provide the catalogue of existing services that users can start applications from (like an App Store) and finally, the User Manager block will manage any additional information and provide it to the mF2C users’ profiles and accounts.

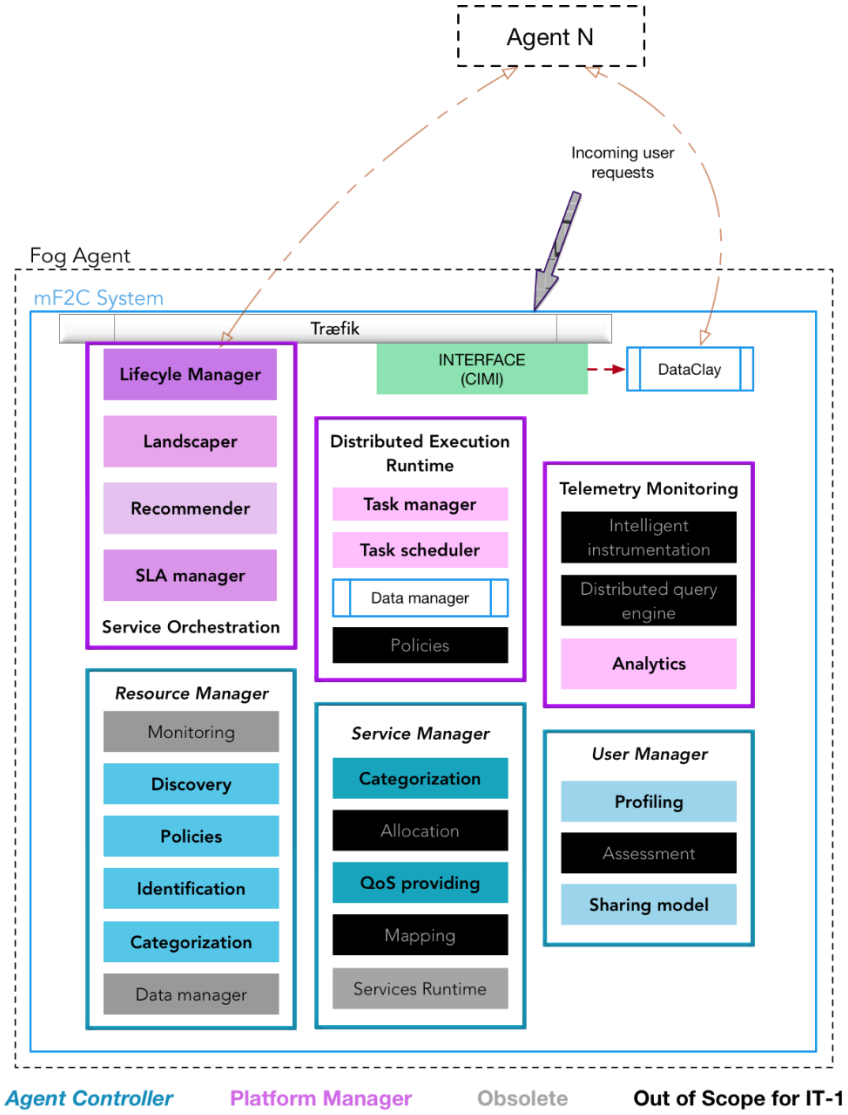


Figure 1 Components layout for IT-1

2.3 Conventions

To ease the collaborative development for IT-1[1], certain programming and organizational conventions have been adopted:

- **Code Versioning:** all components are to be hosted in GitHub [6], publicly available;
- **IT-1 Deployment Strategy:** individual component will be treated as separate services, being delivered in the form of Docker [7] containers, integrated via Docker Compose. See more details about the system orchestration in section 4.2;
- **Documentation:** each component repository shall have the respective technical documentation in GitHub, while a public general mF2C documentation page [8] shall be set up;
- **Components' Visibility:** only component which need inter-agent communication should be visible from outside the mF2C agent;

1 These conventions will be reassessed after IT-1 and may be relaxed or dropped altogether.

- **Port Mappings:** to ease and standardize inter-components' interactions, all components have been equipped with their own REST API which is exposed to the other components internally within the Docker network, according to the port mappings defined in Table 4..

Component	*Port Range*	*Module/Service*	*Proposed Port*	*Service Name*
Interface	-	CIMI	8201	cimi
		Traefik	443	proxy
Service Orchestrator	46000-46997	Lifecycle Manager	46000	lifecycle
		Landscaper	46010	landscaper
		Recommender and Analytics	46020	analytics_engine
		SLA Manager	46030	sla-manager
Resource Manager		Discovery	46040	discovery
		Policies	46050 (REST) 46051 46052	policies
		Identification	46060	identification
		CAU Client	46065	
		Categorization	46070	resource-categorization
DER		Task Manager	46100	COMPSs
		Task Scheduler		
Service Manager		Categorization	46200	service-manager
		QoS Providing		
User Manager		Assessment	46300	user-management
		Sharing Model		
		Profiling		
Security		Certificate Authority	51433 52433 53433 51022 52022 53022	
CAU		Control Area Unit	46400 46410	

Table 4. Ports and naming conventions

3. Workflows update

As the project evolves, the different workflows, presented in previous deliverables describing the whole set of functionalities envisioned for the mF2C system, are being updated to accommodate enriched features. Thus, in this section we present the revised workflows as previously foreseen for the IT-1 Reference Architecture, classified according to the different functionalities.

3.1 Registration and Identification

This workflow shows the initial process when a user (using his/her device) wants to join the mF2C system. The process includes the registration of the user, the download of the mF2C agent software, as well as the assignation of a user identification and device identification (userID and deviceID) to be used later for the security functionality. At this step, the user has not yet joined an mF2C coverage area.

Both the IDKey (also called user ID) and the Device identifier are calculated at the cloud side, the user ID during the registration and the Device ID once the user has downloaded and started the mF2C Agent for the first time.

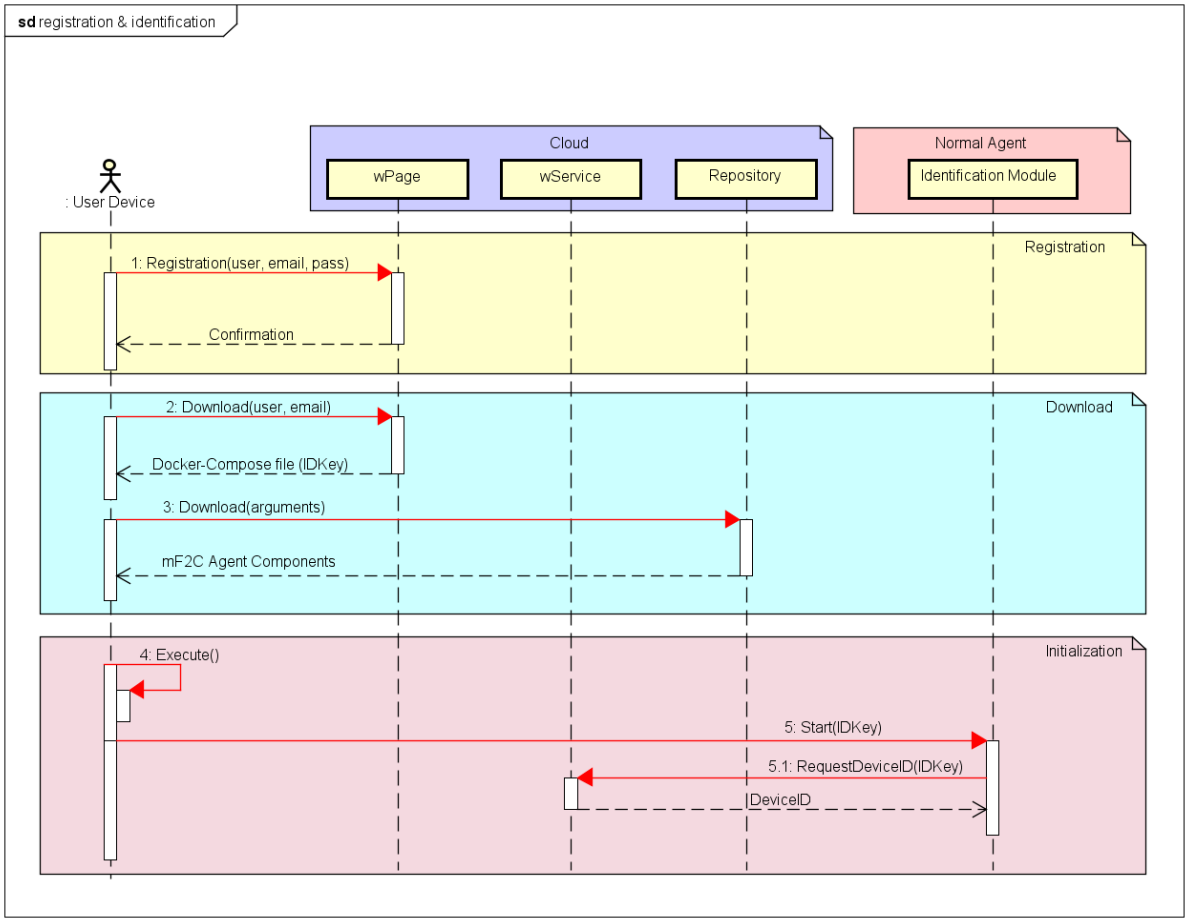


Figure 2 Registration and Identification Workflow

As shown in Figure 2, the registration process starts when the user connects to the mF2C provider webpage and enters his/her user name, email address and chooses a password.

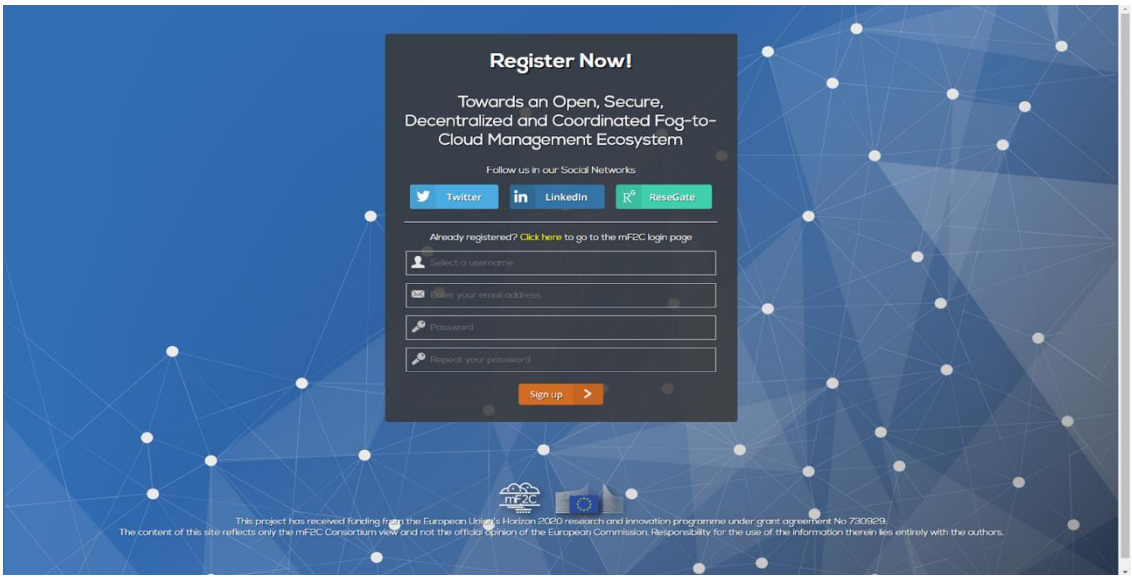


Figure 3 Registration page screenshot

Once the user is registered, the next times he/she enters in the mF2C provider webpage (for instance, for downloading the mF2C agent in a different device), he/she only needs to log in with his/her user and password, as it is shown in Figure 3.

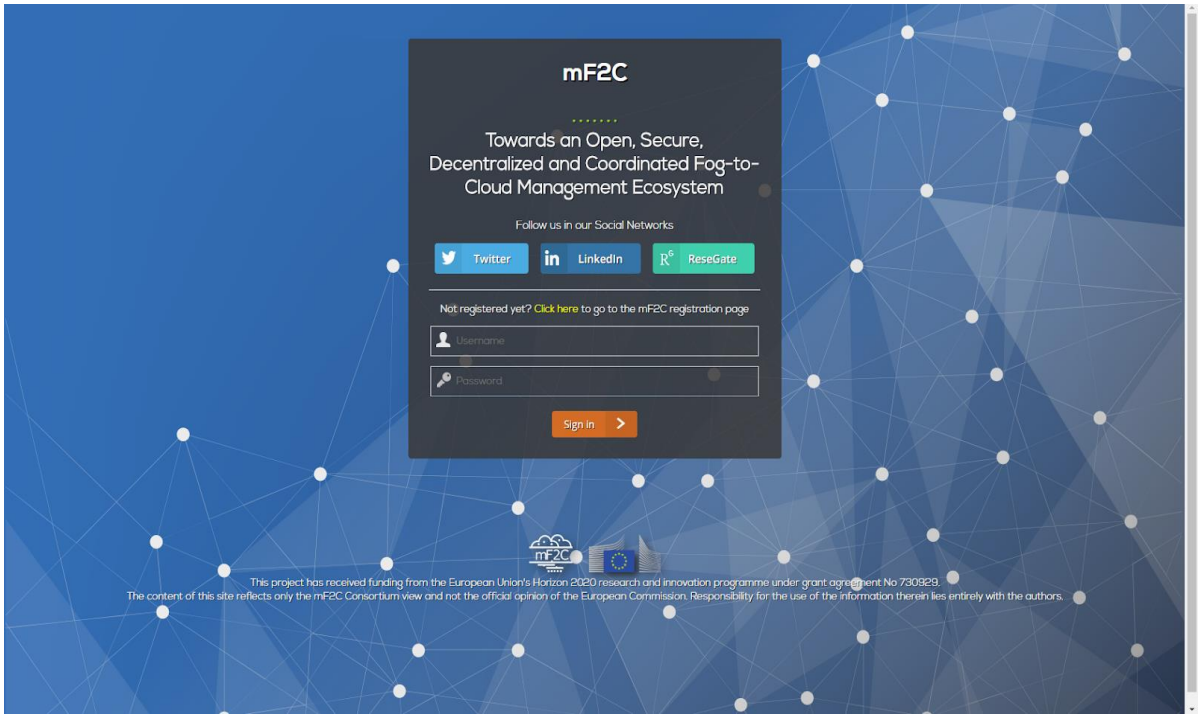


Figure 4 Login page screenshot

Regarding the download file, the system will provide every user with a compose file that includes the agent requirements and configurations, ports for each mF2C component and a unique IDKey. This IDKey will be generated as shown in the following formula:

$$IDKey = hash512(useremailaddress)$$

Figure 5 Download page screenshot

3.2 Discovery, Authentication (Security) and Categorization

This workflow (Figure 6) describes the process followed by a device (potential mF2C agent) approaching an mF2C area it may want to join. Certainly, the device is able to run an mF2C agent because it has the mF2C agent software installed as a result of the previous registration process. The first step for the device is to be discovered, and to that end, the device starts scanning the area looking for possible leaders. When it detects the beacons of a leader, a process of authentication involving the agent, the leader, the CAU and the CA is performed. After this process, and if the authentication is successful, the device becomes an mF2C agent, able to contribute to the mF2C system or to request the execution of services. It is worth mentioning that after a successful authentication of a new device, the categorization module is run, the information about device's available resources, if any, is stored in the agent's local database, and finally dataClay is requested to synchronize this local database with the leader's database.

In Figure 7 we show one of the steps of the previous workflow, where the agent, after detecting the beacon (step 3), decodes the information and sends the MAC address of the leader to the policies block. Currently (IT-1), there is no re-attempt if the process fails, but it should be added in next iteration.

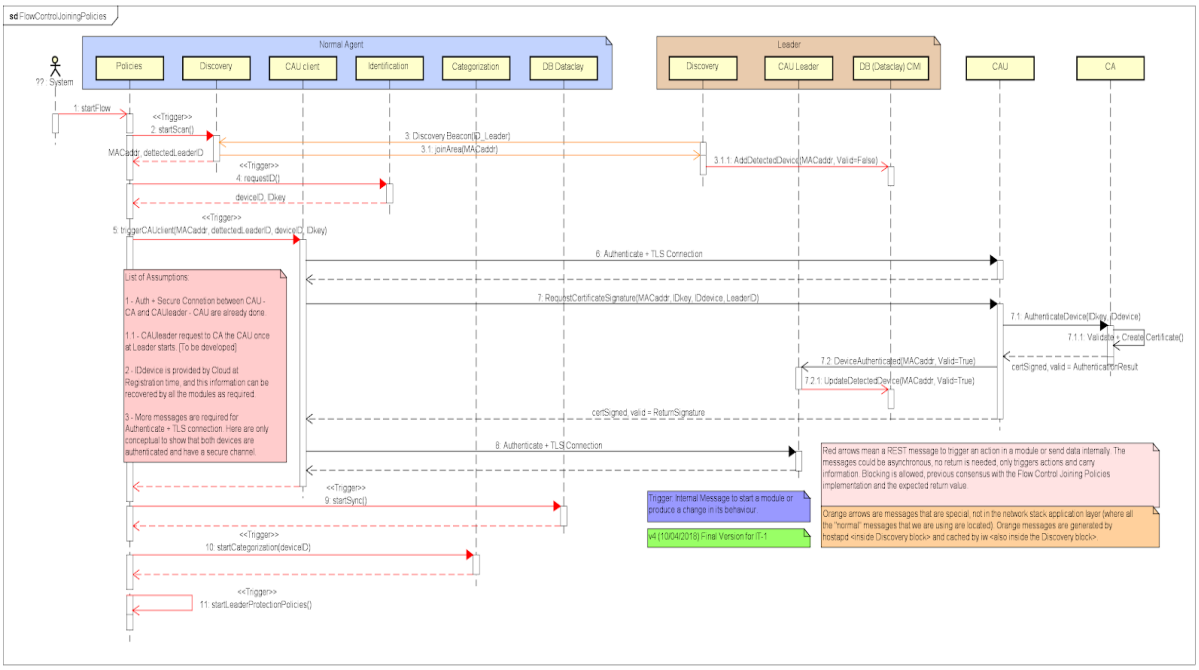


Figure 6 Discovery, authentication and categorization workflow

```
craax@craax-laptop:~$ curl -X   GET http://0.0.0.0:46040/api/v1/resource-management/discovery/scan/wlp2s0
{
  "error_message": "None",
  "found_leaders": [
    {
      "Bssid": "3c:a0:67:20:0d:67",
      "Leader ID": "a1b2c3d4a1b2c3dd",
      "Reward": "5EUR",
      "Service type": "High storage",
      "Urgency level": "MEDIUM",
      "vsie_data": "01 08 a1 b2 c3 d4 a1 b2 c3 dd 02 01 01 03 01 05 04 01 02"
    }
  ],
  "used_mac": "7c:b0:c2:bb:17:a0"
}
```

Figure 7 Screenshot of the discovery module execution

In Figure 8 the discovery, authentication and categorization flow (Figure 6) is executed according to the workflow communicating with all the modules involved.

```
[DEBUG]:  Sending request...
[FCJP]:  Discovery Scan Trigger.
127.0.0.1 - - [24/Apr/2018 15:11:31] "POST /api/v1/resource-management/policies/startFlow/ HTTP/1.1" 200 -
[DEBUG]:  request result {'started': True}
[FCJP]:  Discovery Scan Trigger Done.
[FCJP]:  GetID from Identification by Trigger.
[FCJP]:  Identification Trigger Done.
[FCJP]:  CAU client Trigger.
[FCJP]:  CAU client Trigger Done.
[FCJP]:  Categorization Start Trigger.
[FCJP]:  Categorization Start Trigger Done.
[FCJP]:  LPP Start Trigger.
127.0.0.1 - - [24/Apr/2018 15:11:46] "GET /api/v1/resource-management/policies/startLeaderProtectionPolicies/ HTTP/1.1" 200 -
[FCJP]:  LPP Start Trigger Done.
```

Figure 8 Workflow execution

Finally, in Figure 9 we show a screenshot of the categorization module execution, taken after step 10 in the workflow.

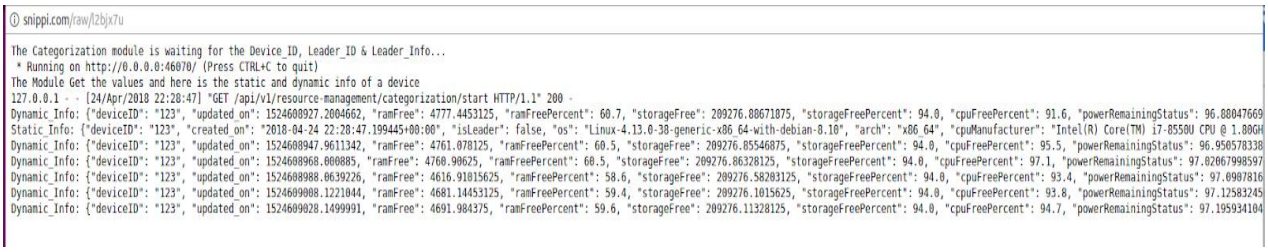


Figure 9 Screenshot of the categorization module execution

3.3 Leader failure

This workflow presents the leader failure scenario, that is when a leader fails (or is not accessible) it must be replaced by one of the agents acting as backup (assuming a policy exits to select such a backup, or a designated backup has been pre-selected). In IT-1, the leader is essential to providing the required functionality, so our options, in increasing order of complexity, are as follows:

1. Accept that a leader can fail - with severe loss of fog functionality, but this may be acceptable in some cases, e.g. a temporary loss of fog connectivity.
2. Designate a backup leader. Through a process of data transfer (from the failed leader's database, if possible) and/or rebuilding its database (re-replicated from the agents' local database), the backup takes over.
3. Implement a leader selection process whereby any agent which is capable of taking on the leadership role, can be elected leader. These can be done through simple processes, which have a small probability of failure, or more sophisticated but more time-consuming processes.

While all of these options were considered, our primary focus in IT-1 is option 2. The success of this process involves transferring the aggregated database (fogArea topology in Figure 10) from the leader to the backup node, as well as the selection of a new backup node (the agent on the right in the figure). Finally, once the backup becomes the leader it must start sending beacons and aggregating new agents in the area.

In Figure 11, we can see the log from the leader, showing the leader selecting a backup from the topology and receiving the *keepalive* protocol from the backup. Once the procedure is done, the backup keeps watching the leader.

Then, in Figure 12, we disconnect the leader and the backup detects that. The backup then becomes a Leader and perform some tasks (e.g. loading the topology, sending some internal communication to blocks...) before selecting another backup. It is important here only to select an agent which is capable of being a leader[2].

In Figure 13 we show how the new backup is selected.

2 For example, if the agent is implemented on a user's mobile phone, it can run out of battery power or disappear from the fog; thus an agent running on a mobile phone is not considered capable of being a leader. We could have relied on the backup process to recover in this scenario, but it seems prudent to minimize the need for recovery.

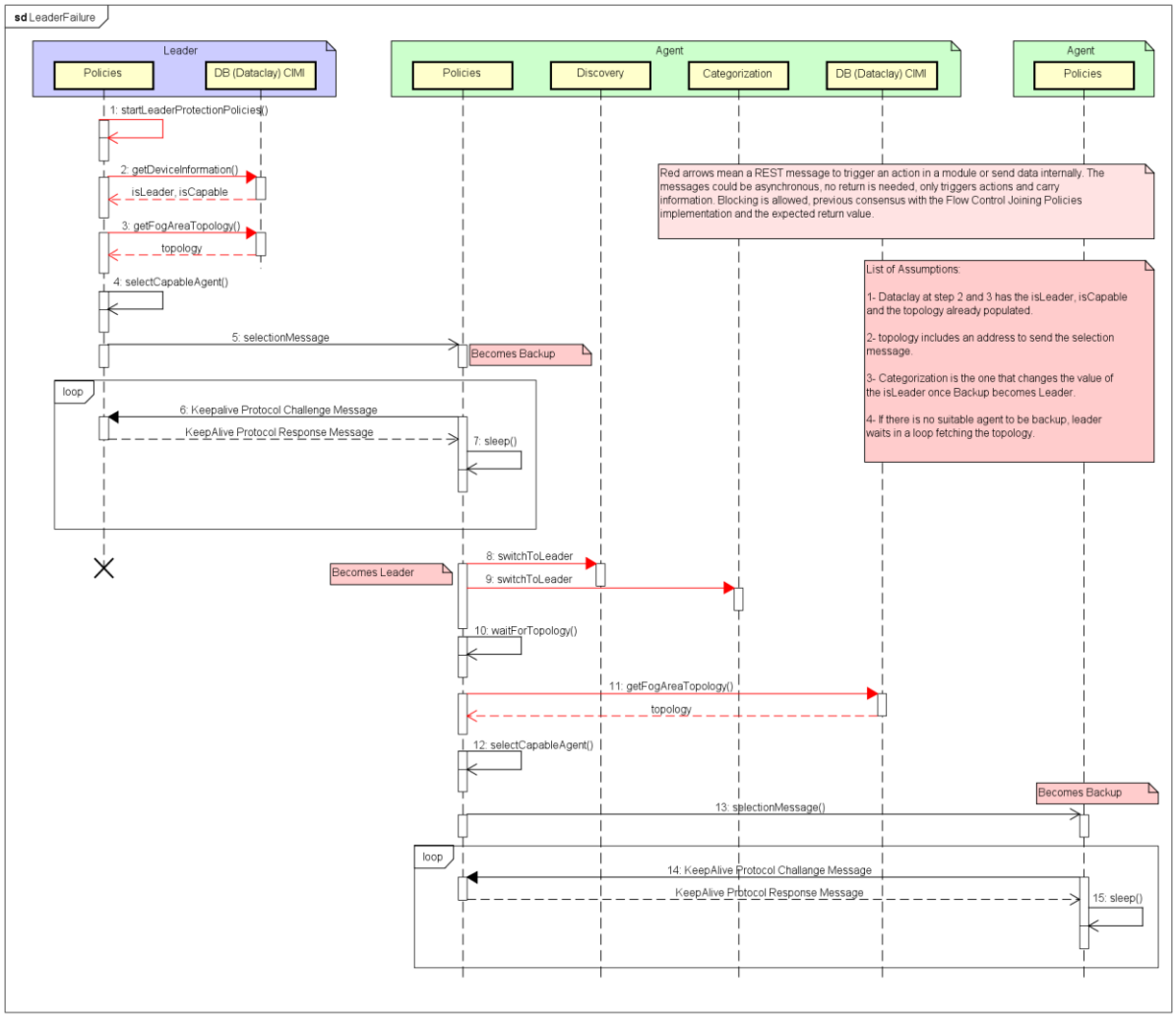


Figure 10 Leader failure workflow

```
[FCJP]:  LPP Start Trigger.
[LPP]:  Leader seting up
127.0.0.1 - - [23/Apr/2018 15:27:03] "GET /api/v1/resource-management/policies/startLeaderProtectionPolicies/ HTTP/1.1" 200 -
[LPP]:  Backup selected. Agent (123, '192.168.5.2')
[LPP]:  Elected Addr: 192.168.5.2 in Port: 46051
[LPP]:  Backup Welcome Server created.
[FCJP]:  LPP Start Trigger Done.
[LPP]:  Backup with addr ('192.168.5.2', 52190) connected to leader keepalive socket. Keepalive protocol started.
[LPP]:  KeepAlive Message from Backup received. Status: All Green
[LPP]:  KeepAlive Message from Backup received. Status: All Green
[LPP]:  KeepAlive Message from Backup received. Status: All Green
```

Figure 11 Screenshot of the backup selection and keepalive reception

```
[LPP]:  Reply received, Leader still alive [pong]. Sleeping  1 seconds.
[LPP]:  Keepalive sent [#14]
[LPP]:  Reply received, Leader still alive [pong]. Sleeping  1 seconds.
[LPP]:  Keepalive sent [#15]
[LPP] ERROR:  Message from leader received is not standar. Recv msg ''
[LPP]:  ## LEADER IS DOWN! ##
[LPP]:  Leader seting up
[LPP]:  Backup selected. Agent (123, '192.168.5.4')
[LPP]:  Elected Addr: 192.168.5.4 in Port: 46051
[LPP]:  Backup Welcome Server created.
[LPP]:  Backup with addr ('192.168.5.4', 59780) connected to leader keepalive socket. Keepalive protocol started.
[LPP]:  KeepAlive Message from Backup received. Status: All Green
[LPP]:  KeepAlive Message from Backup received. Status: All Green
[LPP]:  KeepAlive Message from Backup received. Status: All Green
```

Figure 12 Screenshot of leader disconnection

```
[FCJP]:  LPP Start Trigger.
127.0.0.1 - - [23/Apr/2018 15:26:42] "GET /api/v1/resource-management/policies/startLeaderProtectionPolicies/ HTTP/1.1" 200 -
[LPP]:  I'm not a Leader.
[FCJP]:  LPP Start Trigger Done.
[LPP]:  I'm capable to be Leader.
[LPP]:  Server created.
[LPP]:  Waiting to be selected.
[LPP]: New connection - ('192.168.5.3', 33002)
[LPP]: Becoming backup due leader selection.
[LPP]:  I'm selected to be a backup. Seting up
[LPP]:  Correct connection between backup and leader.
[LPP]:  Keepalive sent [#0]
[LPP]:  Reply received, Leader still alive [pong]. Sleeping  1 seconds.
[LPP]:  Keepalive sent [#1]
[LPP]:  Reply received, Leader still alive [pong]. Sleeping  1 seconds.
[LPP]:  Keepalive sent [#2]
```

Figure 13 Screenshot of the new backup selection

3.4 Monitoring

The IT-1 Monitoring workflow (Figure 14, below) is an update to figure 8 in D3.3 [3] (section 3.6, p.22), in that the *Intelligent Instrumentation* is pushed to IT-2. The probes would have registered with this module, and the module would have requested that the probes update their metrics collection frequencies. Each telemetry probe installed on the device will just register with the Distributed Query Engine for IT-1 to allow their metrics to be queried from any device in the mF2C cluster.

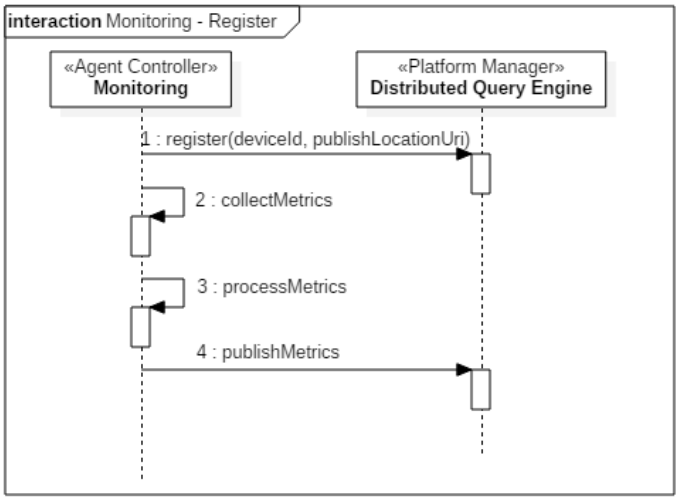


Figure 14 Monitoring workflow

3.5 User Profiling

The User Profiling module is responsible for managing the user's profile properties, and it is part of the User Management module of the Agent Controller.

The following workflow (Figure 15) presents some minor changes regarding the workflow presented in D3.3 [3]. In this workflow, the Profiling module interacts with CIMI instead of calling directly the database. First, it is called to initialize a user's profile. Then, it processes the request parameters, and finally it interacts with CIMI in order to store this information (profile's properties).

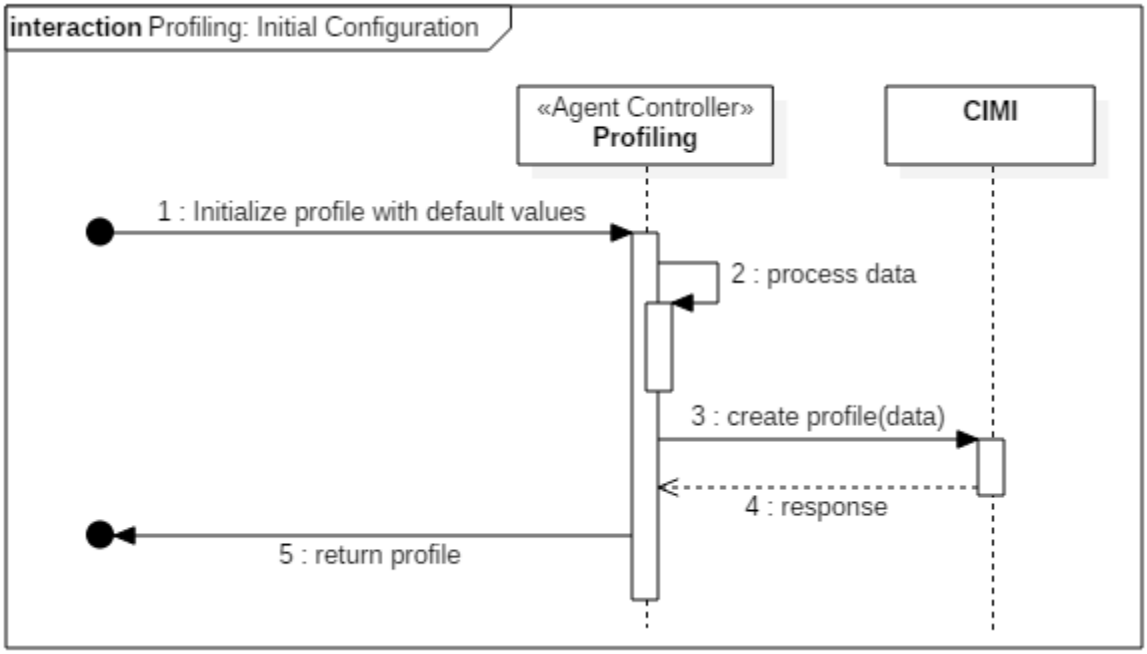


Figure 15 Profile properties configuration (initialization and update workflow)

3.6 Sharing model

The sharing model module is responsible for the management of the properties that define the device's shareable resources.

The initialization workflow describes the process that initializes these properties. It presents some changes with regards to the workflow described in D3.3 [3]. This (IT-1) workflow skips the calls to the Categorization module, and replaces the call to the database with a call to CIMI. Thus, after processing the request, this component calls CIMI to store the information.

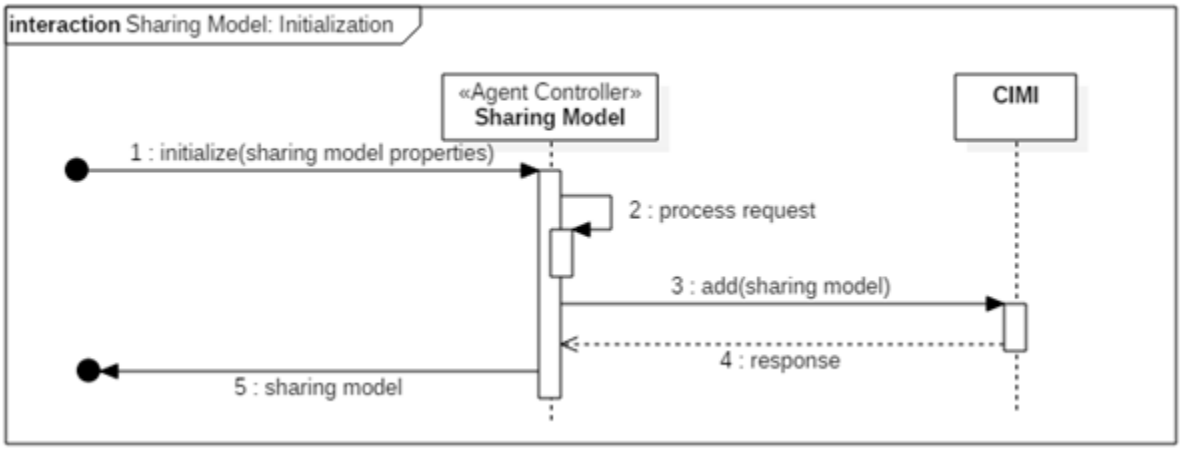


Figure 16 Configuration of shareable resources when installing mF2C software

3.7 Service registration and Service Level Agreement-based QoS Analysis

This workflow demonstrates the registration of a new service and the QoS-providing function in the Service Manager (see Figure 17). When a new service is registered into the mF2C system, and defined in a JSON format following a specific structure (shown later in Figure 42), the service definition is sent to the service manager through an interface in the Categorizer. Before to accept or reject the service, the categorizer updates its local repository of services from CIMI and, then, verifies if the service can be accepted for registration or rejected in case already exists.

When the Lifecycle Manager issues a request in order to check if a given list of agents can be used to execute a service, the QoS provider gets the service instance from CIMI by specifying the service instance id. Then, the QoS provider gets the SLA violations from the SLA Manager using the agreement id that is specified in the service instance. Similarly, it gets the service from the Categorizer using the service id specified, also, in the service instance. After making the decision on which agents should be accepted for the service execution, a modified service instance is returned to the Lifecycle Manager with the updated list of suitable agents.

The decision whether a certain agent can or cannot be used for a certain service instance is based on the number of SLA violations occurred in previous executions of that specific service. With this information, the QoS Provider uses reinforcement learning to allow or block the use of specific agents. The QoS provider design block in shown in Figure 18.

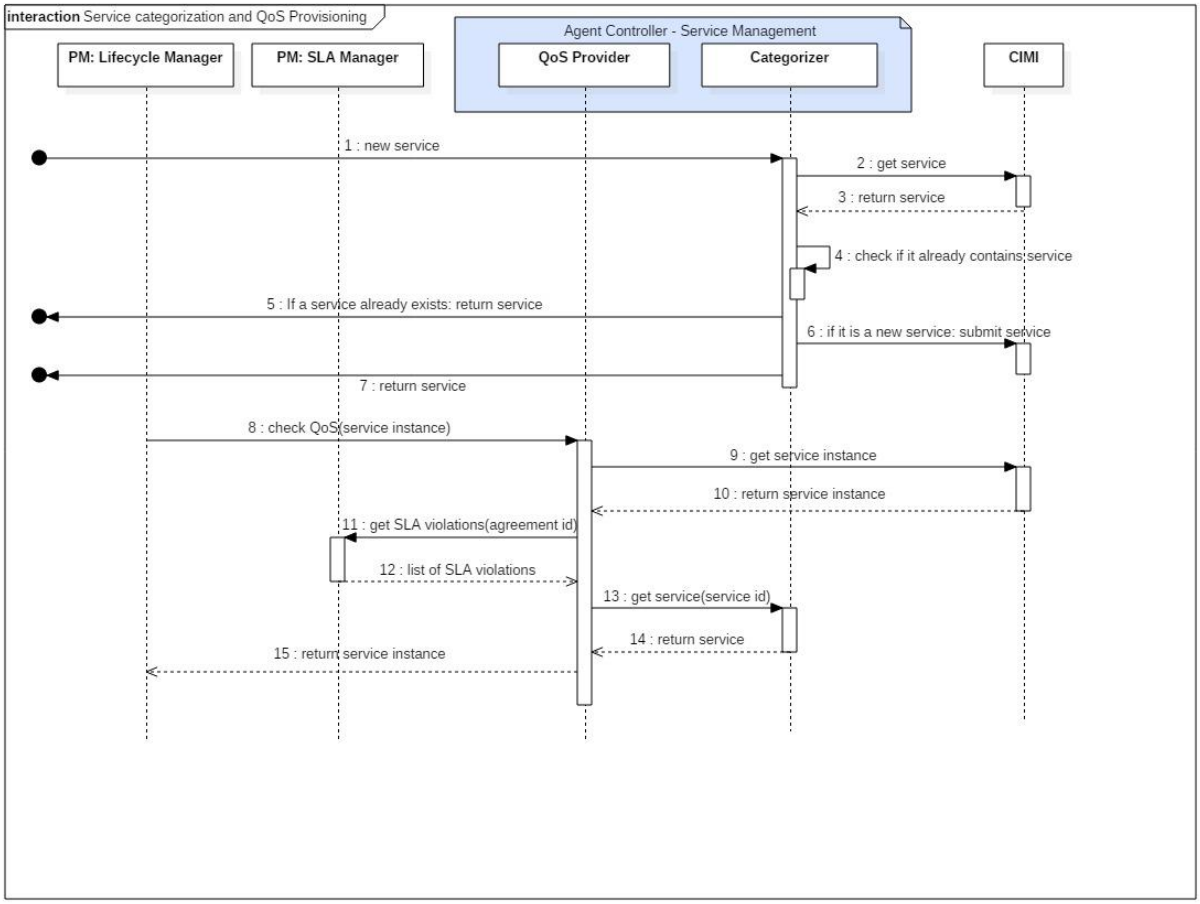


Figure 17 Service registration in the Service Manager and QoS provider

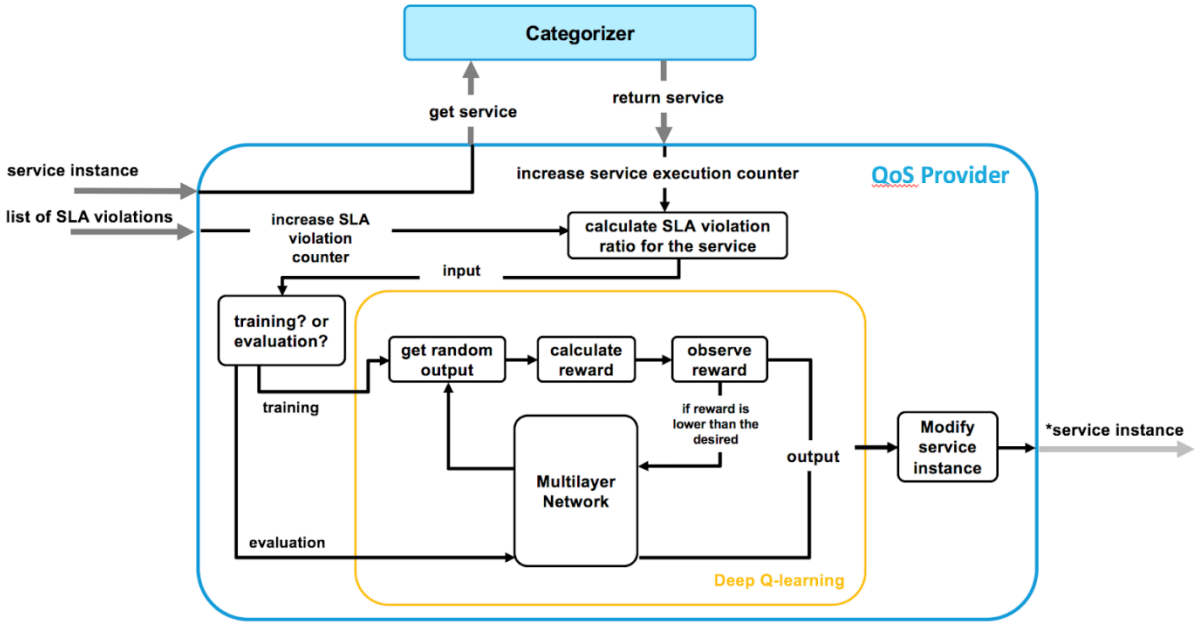


Figure 18 QoS Provider - Deep Q-learning algorithm design

In order to determine if the suggested agents by the service instance should be used, the QoS provider uses the number of service executions and the number of SLA violations, to calculate a ratio that is used as the input for the Deep Q-learning (DQL) algorithm. Then, it must be decided whether that input is taken for training or for evaluation (the decision process being described below). In the case of training, the DQL algorithm will initially get a random output, which determines which agents are accepted. Based on the output, a reward is calculated following the next function:

$$r_t = \sum_{n=0}^{N} y_n \left(-2x_s + 1\right) + (1 - y_n)(x_s - 1)$$

where the N is the total number of agents specified in the service instance, y_n is 1 when the agent n is chosen, 0 otherwise, and x_s is the input ratio. The calculated reward is observed by the network and in case to be lower than a specific threshold, a new random output is generated and the process is repeated. When the reward is greater or equal than desired, the output is used to modify the list of allowed agents in the service instance. In case of evaluation, the QoS provider will directly ask the network about an optimal output for a specific input. How to decide if an input is taken for the training or for the evaluation is based on the quantity of the already acquired knowledge in the network. For simplicity in IT-1, this decision is only based on a certain number of service executions.

While the QoS provider block could use the reward function without the need of using deep learning, the output would be only determined by that function missing other non-trivial factors like the relation between the failure of the execution of a service and the agents that were involved. For that reason, the proposed algorithm can learn in every situation by taking random decisions and help to improve the decision making in the evaluation period. To be noted, the presented algorithm is only a simplified version that will be used for testing, considering only the current development of the system. For future releases, the reward function, the input, the output or how the decision to choose training or evaluation is taken could change in order to improve the effectiveness of the algorithm or just to adjust the compatibility to other blocks.

3.8 Lifecycle Management

Once there is an mF2C cluster composed by one or more agents, the Lifecycle Management enters the scene. This component is responsible for managing the lifecycle of the services executed by the mF2C platform. This management includes the process of deploying these services in the agents, and all the basic operations: start, stop and termination of the service.

At this stage of the project (IT-1), the Lifecycle can handle two types of services:

- On one hand, it can manage “dockerized” services (single docker images). This includes images based on COMPSs (DER). These are single Docker images that include COMPSs and the applications that will be executed by this distributed runtime.
- On the other hand, it can manage services composed by two or more Docker images configured in *docker-compose* deployment files.

Depending on the capabilities of the selected agents and their resources, these services can be deployed and executed among several agents, or in one agent.

Service initialization

The first workflow in this section shows the submission or initialization of a service in an mF2C cluster. This workflow (Figure 19) is started when a user wants to submit a service to a set of devices. The Lifecycle of the agent that gets this request is the responsible for the following workflow.

First, the ***Lifecycle*** processes this request ('submit a service') and interacts with other mF2C components to get a list of all the available devices and their resources in the cluster. This task is done by two Platform Manager components: the ***Recommender*** and the ***Landscaper***. The result of the Recommender is a *service recipe*, a recommendation of devices where to run a service, based on previous executions of the service (e.g., 2 medium CPU cores), or the initial service categorization as initial condition. The Landscaper returns the available devices in the cluster that can satisfy the recipe. At the time of publication of the present deliverable, the integration of the lifecycle with these two components is not ready. Because of this, the functionalities of Landscaper and Recommender have been replaced by two functions that simulate the requests and responses to these components.

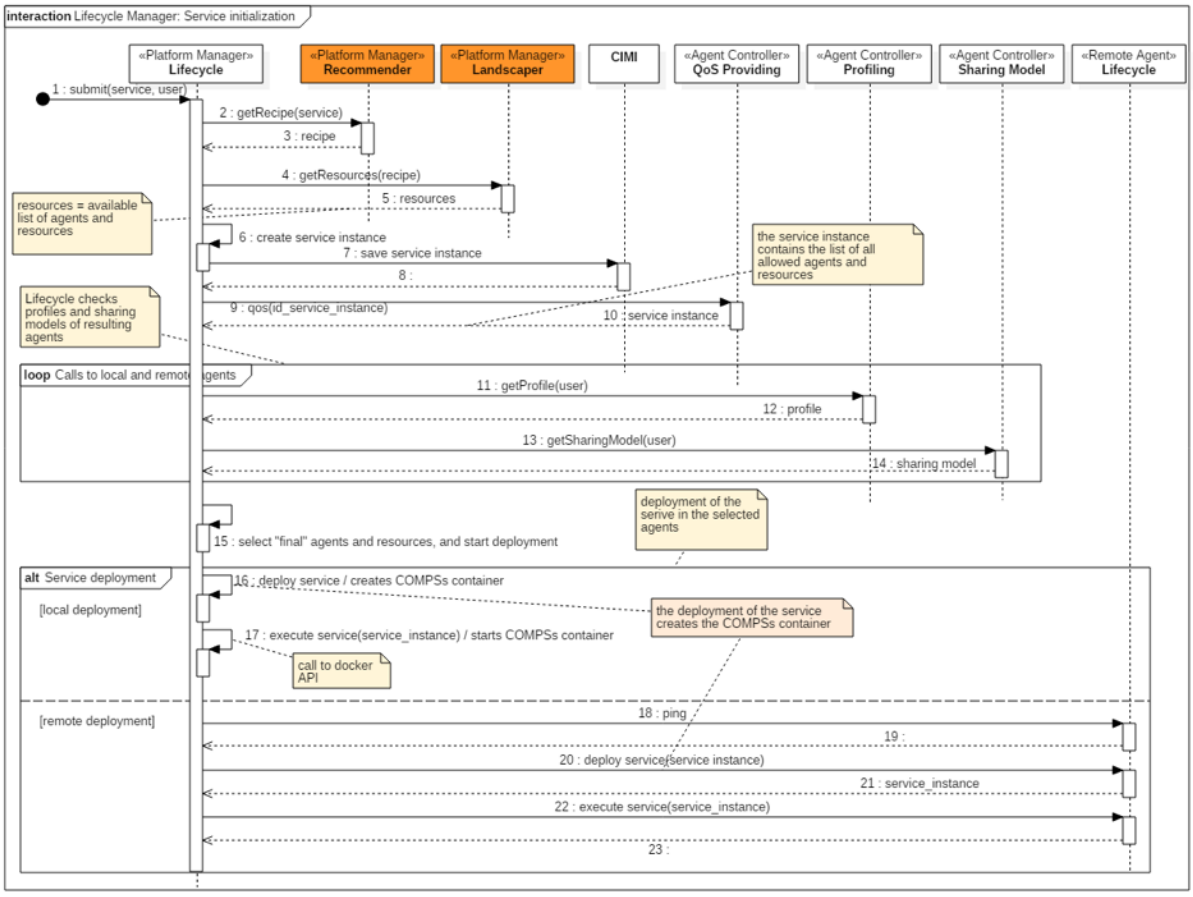


Figure 19 Lifecycle Management - Service initialization

Then, the Lifecycle Manager interacts with the ***QoS Providing***, the ***Profiling*** and the ***Sharing Model***. The information and results of these calls are used by the Lifecycle Manager to decide which ones of the agents and resources obtained from previous calls to Landscaper and Recommender, are the best ones to execute the service. The result of these operations is a list of the "best" available agents for executing this service. This list of agents is included in the service instance object that is created during the service initialization workflow.

The next step of the service initialization workflow consists of the deployment of the service in the selected agents. If the Lifecycle Manager has to deploy them in other (remote) agents, then it calls the lifecycle components of these other agents. As a result of this deployment, one or more Docker containers are created in each of the selected agents. In the case of applications that rely on ***COMPSs***, the result is a COMPSs (a Distributed Execution Runtime) container running in these agents.

Finally, the Lifecycle Manager starts all these containers (execute service request), and initializes the SLA agreement of this service instance. At this point, the SLA Management can start evaluating the agreement. For IT-1, a hand-crafted agreement for the service instance has been previously stored in the SLA Management by an authorized user. This means that the ID of the agreement must be passed as a parameter to the creation of the service initialization. The automatic creation of the agreement is intended for IT-2. In IT-1, we are assuming that the provided service is the execution of a service instance, the service provider is the mF2C platform and the service client is the application requesting the execution of a service instance. This means that each service execution uses a different agreement.

Service operation

After a service has been initialized, the lifecycle offers the following methods to manage the service:

- Start a service: this operation starts the service (e.g. the Docker container), and it is included in the initialization phase. This operation includes a call to CIMI in order to update the status of the service instance object. And it also includes a call to the SLA Manager to start the SLA Agreement process.

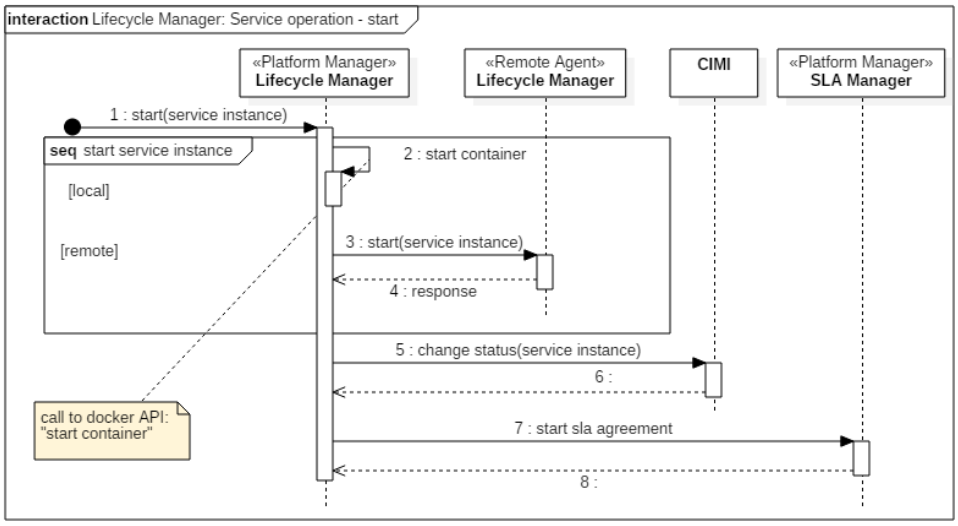


Figure 20 Start a service

- Stop a service: this operation stops a service, updates the status of the service instance object, and finally it stops the SLA Agreement process.

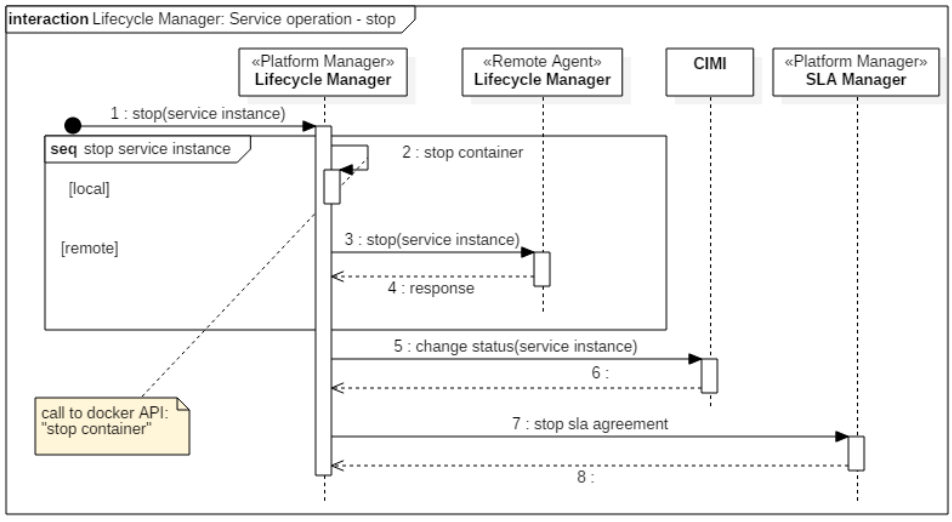


Figure 21 Stop a service

- Execute an application or job: In the case of applications that rely on COMPSs, the Lifecycle Manager component directly interacts with the Distributed Execution Runtime in order to execute applications or jobs in it. It requires that the COMPS container (the service) is started and running. The Lifecycle processes the request parameters and generates a call with the arguments that are needed by COMPSs in order to execute the job specified in the request.

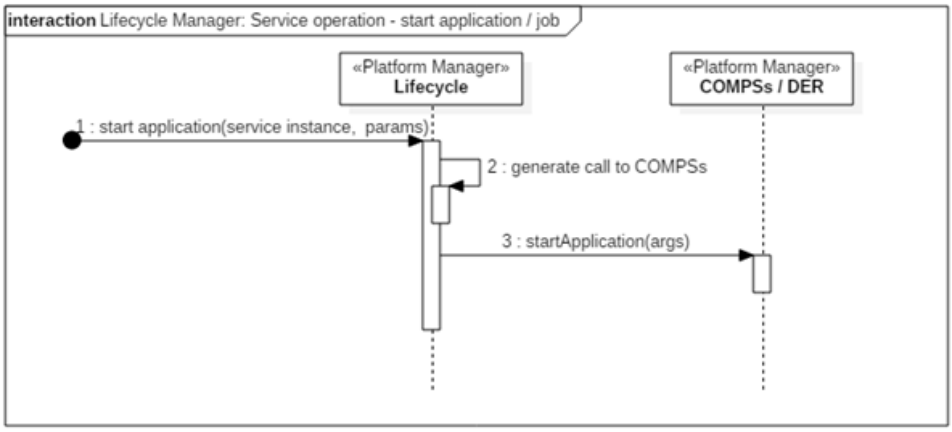


Figure 22 Service operation

Service termination

The following workflow (Figure 23) describes the termination of a service. When the Lifecycle Manager is told to terminate a service instance, it first stops all the containers associated with this service instance. Then it deletes these containers. And finally, the Lifecycle Manager component calls CIMI in order to delete this service instance and contacts the SLA Management to terminate the agreement.

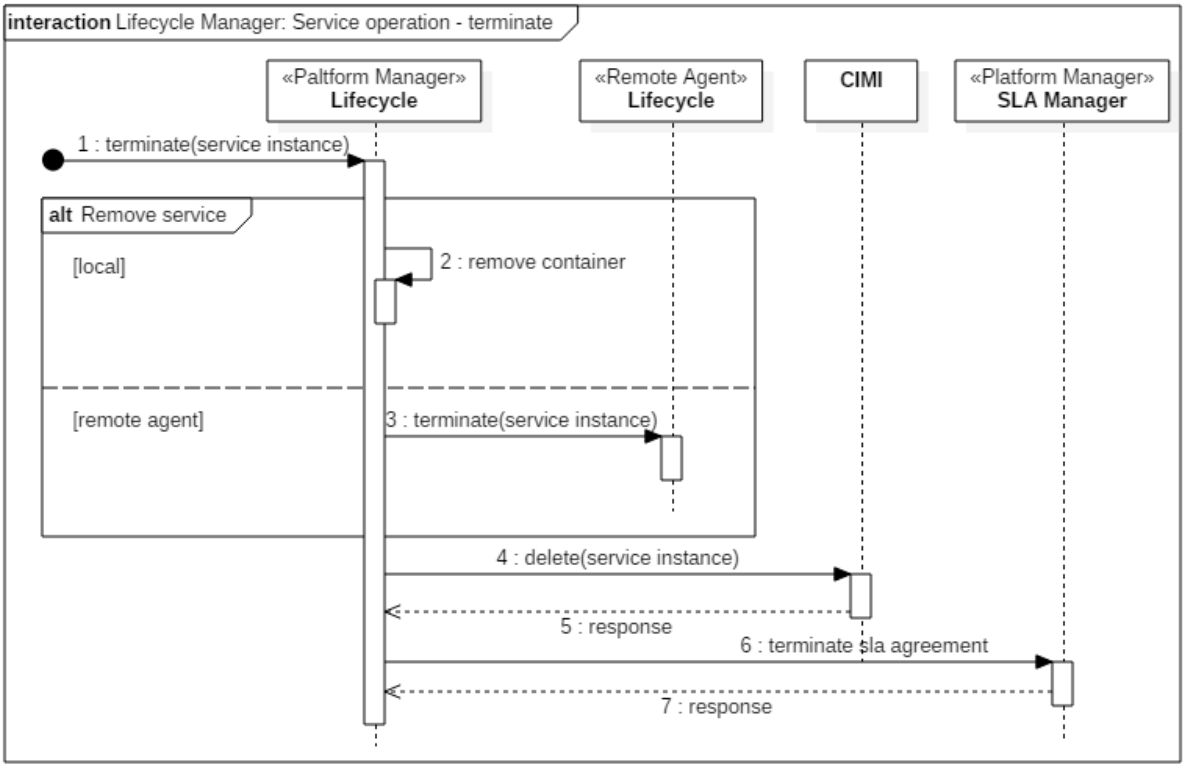


Figure 23 Lifecycle Management - Service termination

3.9 Landscaper

Figure 24 is an updated version of Figure 6 of D4.3 [4]. The update reflects minor changes in where the Landscaper will source the metadata required to generate the landscape model. Information about the devices is queried by the ResourceManager of the Agent Controller and stored in the Device class in DataClay. Lifecycle Manager will return a list of ServiceInstance objects for all services currently deployed. And finally, the COMPSs system is queried for information about the deployment configuration for each service instance. This metadata is then used to generate the Landscape model on start-up and stored in the database.

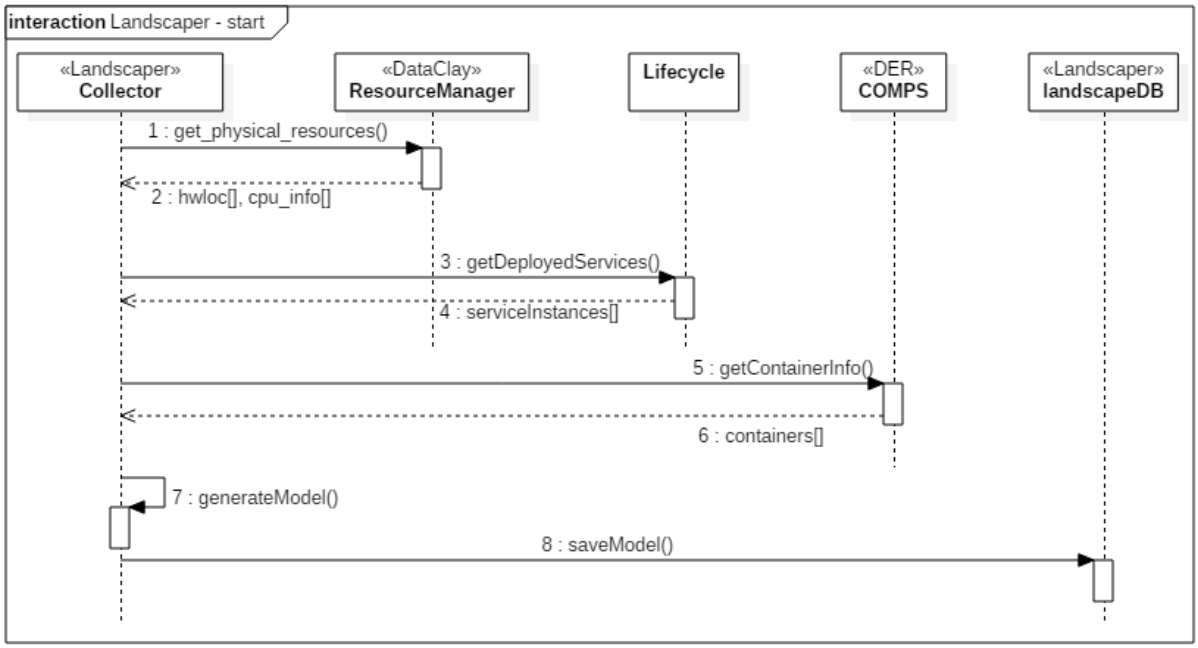


Figure 24 Landscaper start

Updating the Landscape Model:

This set of interactions has not changed from the original design. The REST API of the Landscaper has an update method that the Resource Manager's Discovery module will call when a device is added or removed. Similarly, Lifecycle Manager will call the update method when a new service is deployed.

It is possible that this functionality will be ported to a Message Queue component for IT-2 which would support using a Publish/Subscribe model to trigger and respond to these events.

3.10 Evaluate an SLA Agreement

Once a service is being executed, the service is evaluated to check whether it satisfies the expected performance. The expected service level is defined in the Service Level Agreement, a document that describes the parties that take part in the agreement and the service levels to be guaranteed. For IT-1, the service levels are related to the execution time of the operations provided by a service instance. For example, a service provides a set of applications and the agreement may set the maximum execution time for some of these applications. Every time the execution time is not fulfilled, a violation is generated. These violations are used later by the QoS Provisioning component to provide better recommendations about the agents to be used on next executions of the same service. An example of an agreement for IT-1 is shown in Figure 25, while the sequence diagram that details the evaluation process is shown in Figure 26. The evaluation process is executed periodically (e.g., every minute) and gets the monitoring information stored by the "Monitoring Agents" (in IT-1, the execution time records are stored by the DER component). Using

this information, all of the relevant agreement terms are checked and violations generated on non-fulfilment.

```
{
        "id": "id-graph-wos",
        "type": "agreement",
        "name": "Graph Operations",
        "provider": { "id": "mf2c", "name": "mF2C Platform" },
        "client": { "id": "wos", "name": "World Sensing Use Case" },
        "creation": "2018-01-16T17:09:45.0Z",
        "expiration": "2019-01-17T17:09:45.0Z",
        "guarantees": [
            {
                "name": "eu.mf2c.graph.Dijkstra",
                "constraint": "execution_time < 50"
            },
            {
                "name": "eu.mf2c.graph.MinimumSpanningTree",
                "constraint": "execution_time < 100"
            },
            {
                "name": "eu.mf2c.graph.HeuristicTSP",
                "constraint": "execution_time < 1000"
            }
        ]
}
```

Figure 25 Example of SLA agreement

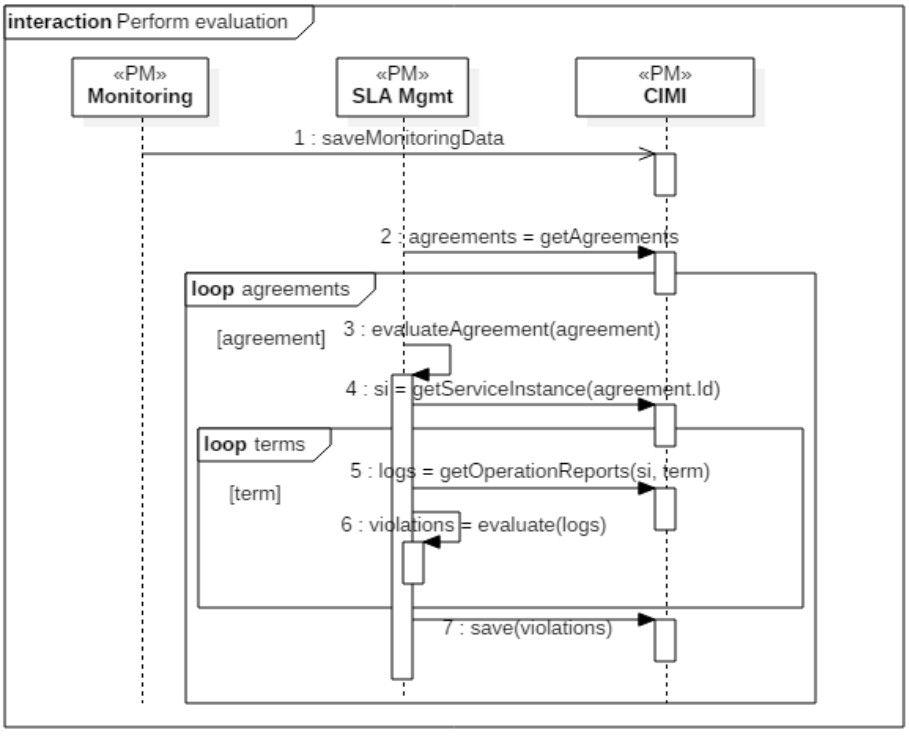


Figure 26 SLA Management – Evaluate agreement

3.11 Distributed Execution Runtime

The implementation of the DER component includes a Start App method that requires a list of resources where to execute the tasks. The resources can be local or remote; in the latter case the task is sent to a remote agent using the Execute Task method.

The implementation of the Update Resources functionalities has been pushed to IT-2.

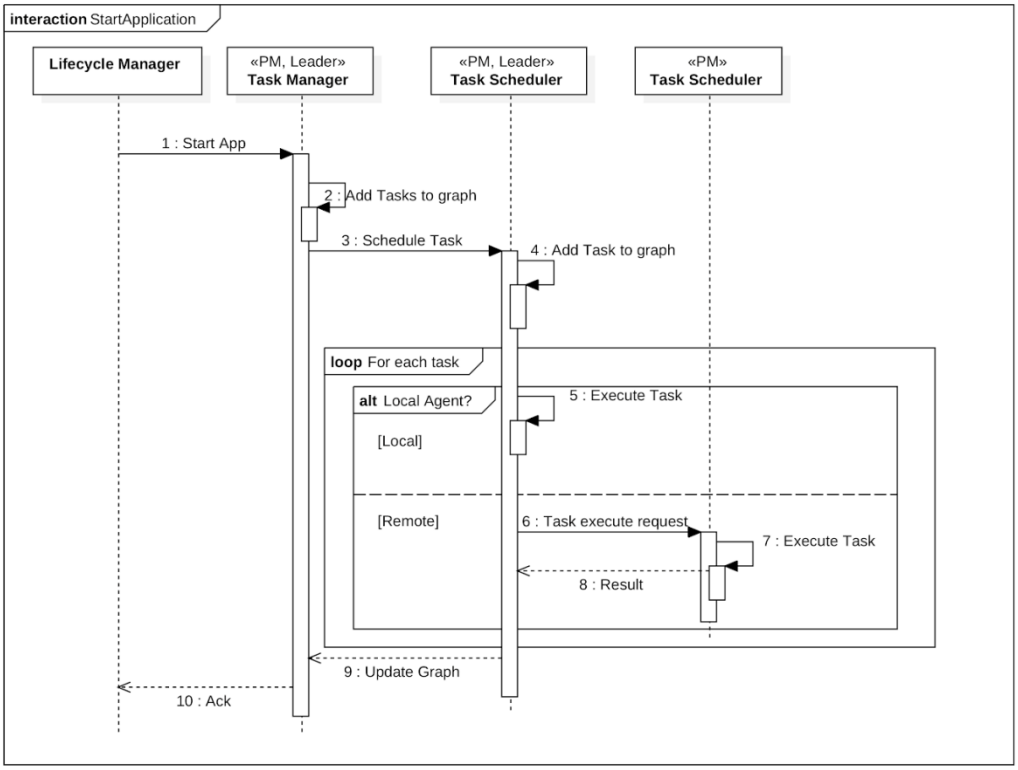


Figure 27 Distributed Execution Runtime – Start application

To start the execution of an application, a set of DER instances need to be deployed. One of these instances acts as master, and possibly also as worker, while the other instances are used as workers. Figure 28 depicts the deployment of a master (upper frame) and a worker instance. When the Start Application method is invoked, the list of resources is passed to the master DER.

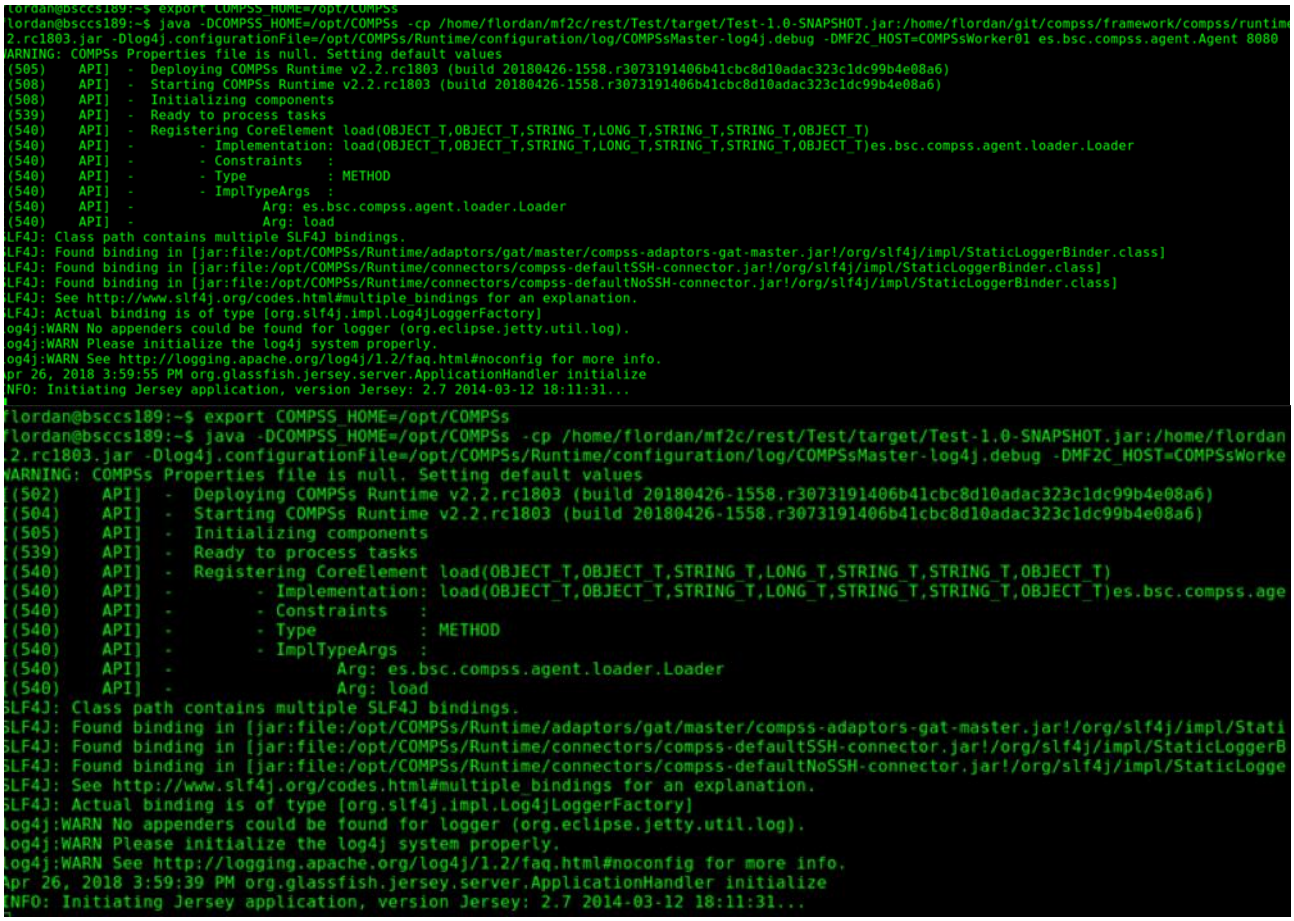


Figure 28 Deployment of DER

Figure 29 depicts the execution of an application (logs on the upper image) that executes three tasks. On the second part of the figure, the tasks are executed by the worker instance of the DER.
Appendix 1 contains the content of the Start Application request.

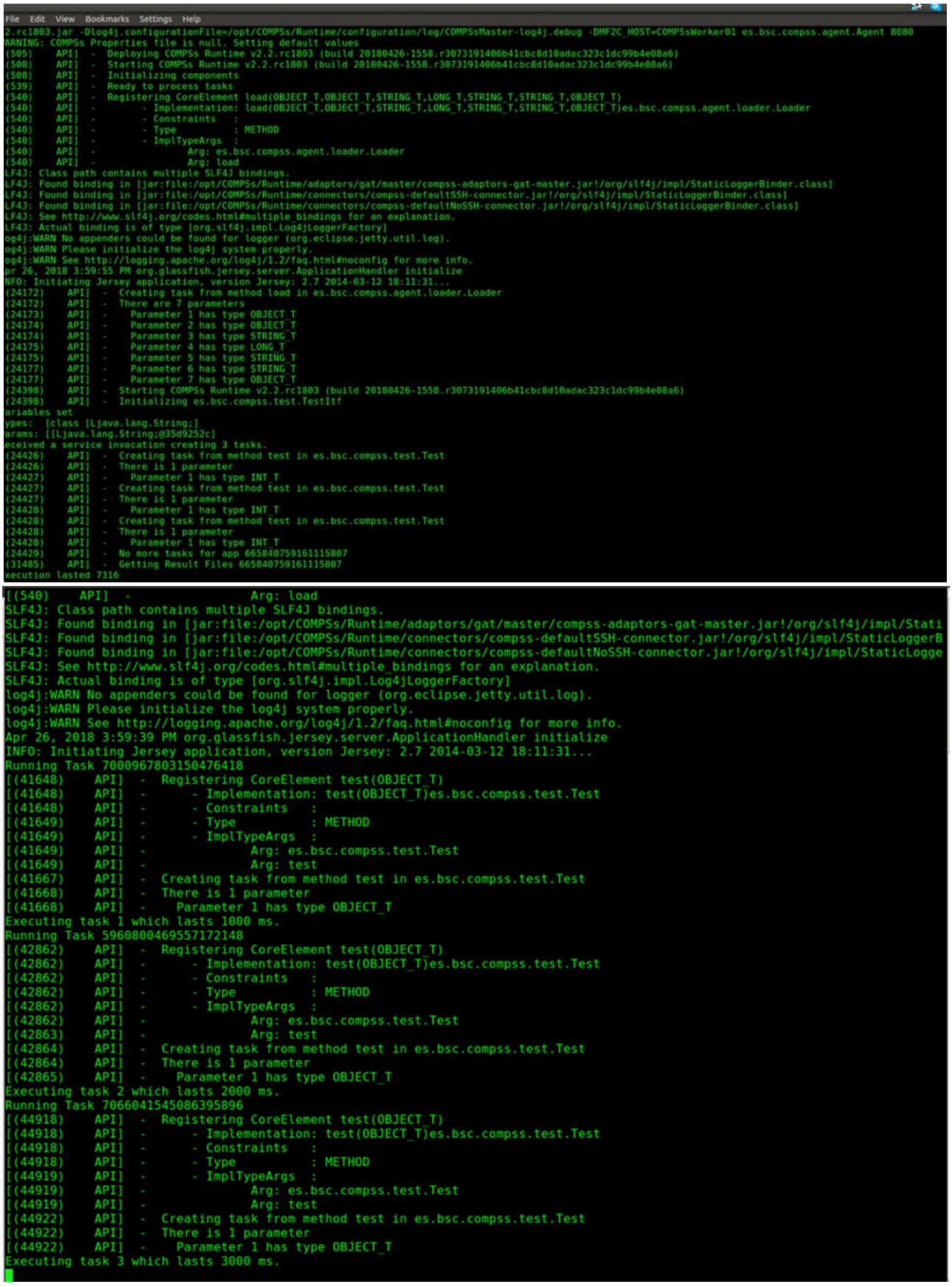


Figure 29 Execution of an application

3.12 Data Management

This section describes the functionalities of the Data Management component. For the IT-1 integration, the Data Management is not directly exposed to the rest of components, but is accessed through the REST-based interface provided by CIMI. Consequently, we have adapted the naming and external behavior of the different functionalities offered by the Data Management component to the CRUD (CREATE, READ, UPDATE, DELETE) data management operations. As explained in previous deliverables, the Data Management component relies on dataClay [9] in order to perform its functions, so that replication and synchronization of data between a device and its leader behaves according to the policies defined in D3.5 [10].

We provide workflows for each of the data management operations exposed by CIMI: CREATE, READ, UPDATE, DELETE, and QUERY. In all of them, for the sake of clarity, we omit the description of the error cases, such as trying to create an object that already exists, or trying to modify an object that does not exist. These cases are covered in the implementation, as will be shown in the corresponding tests.

Unless stated otherwise, all interactions happen locally within an agent, in order to reduce communication between agents as much as possible.

The first workflow corresponds to the storage of a new resource received from CIMI, according to a previous request from any mF2C component contacting CIMI.

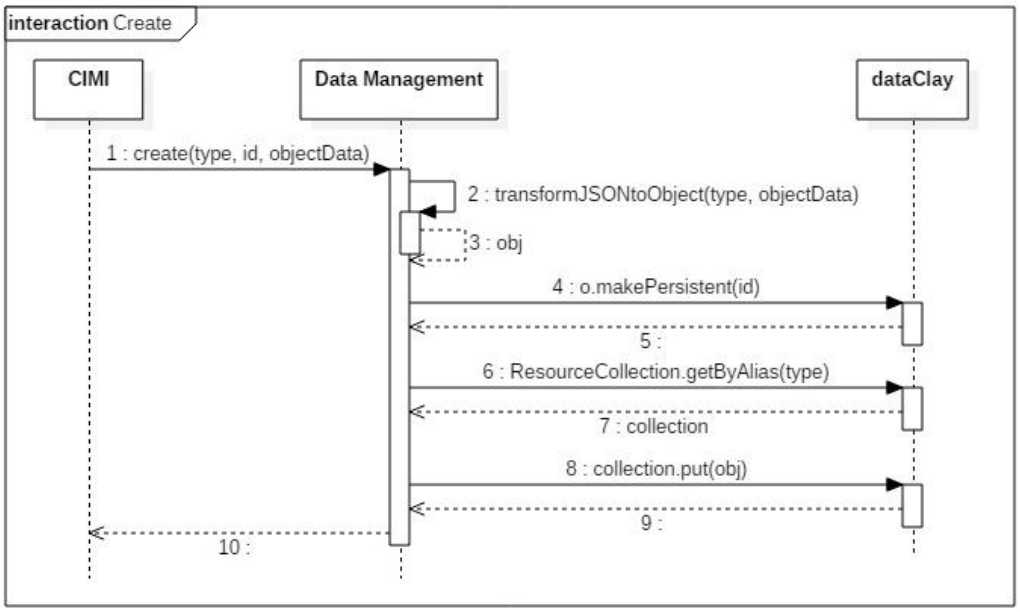


Figure 30 Data Management – Create

When a CREATE request is received from CIMI, the resource data in the form of a JSON string is transformed into an object to be stored and managed by dataClay, representing the same CIMI resource. The object is stored with the 'id' as an alias, which identifies it and provides direct access to the object. The object is also added to its corresponding collection of resources, i.e. a data structure containing all the resources of its same type. This data structure has been implemented in dataClay according to CIMI's access needs (i.e. accessing an object by its id), and will allow components to perform queries (i.e. filtering the objects that satisfy certain conditions), as will be seen in the corresponding workflow.

The next workflow responds to a request by identifier, which returns all the data contained in the object with the specified 'id'.

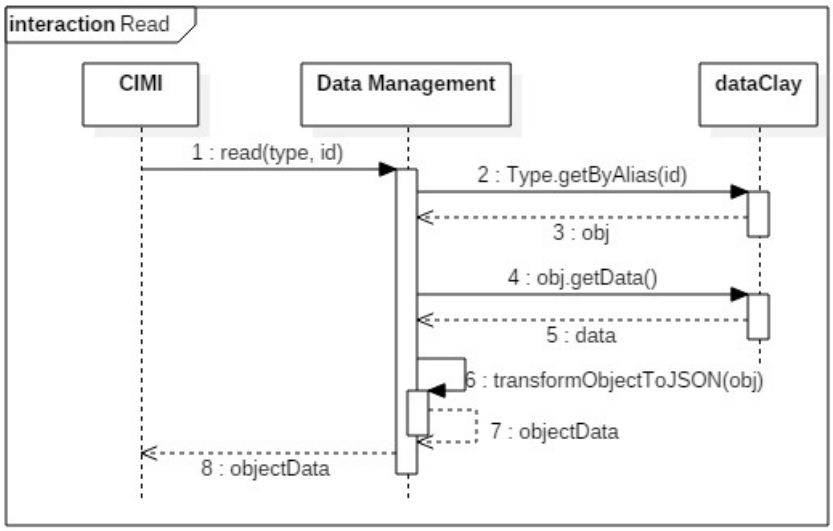


Figure 31 Data Management – Read

When a READ request is received from CIMI the reference to the object with 'id' as alias is obtained from dataClay. The object data is then obtained using this reference ('obj' in Figure 31), and serialize it as a JSON string following the CIMI resource specification corresponding to the resource type. Finally, this data is returned to CIMI.

The following workflow illustrates the update operation.

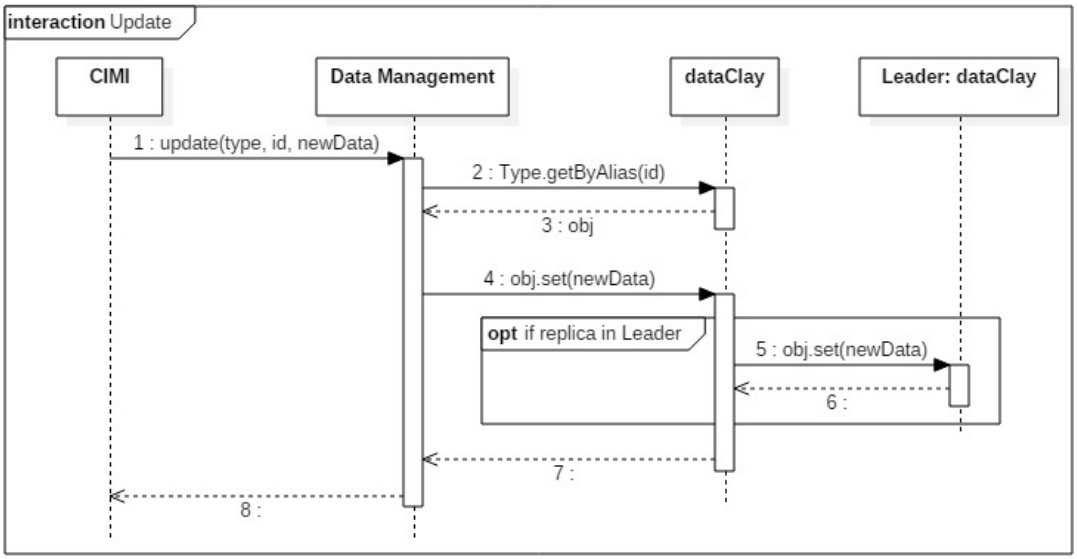


Figure 32 Data Management – Update

Upon an UPDATE request, the corresponding object reference is obtained from the local dataClay as in the previous workflow. The data to be updated is passed to a method that modifies the object locally, and also synchronizes the changes to the replica in the leader, if any. For IT-1, this synchronization follows a strong consistency model, that is, the replica in the leader is updated as soon as the local replica changes. However, more flexible consistency models will be implemented for IT-2 if needed, taking advantage of dataClay's customizable consistency policies, which may be different for different types of data, or different clients (a device may be temporarily disconnected from the fog).

The diagram for the delete operation is shown in the next figure.

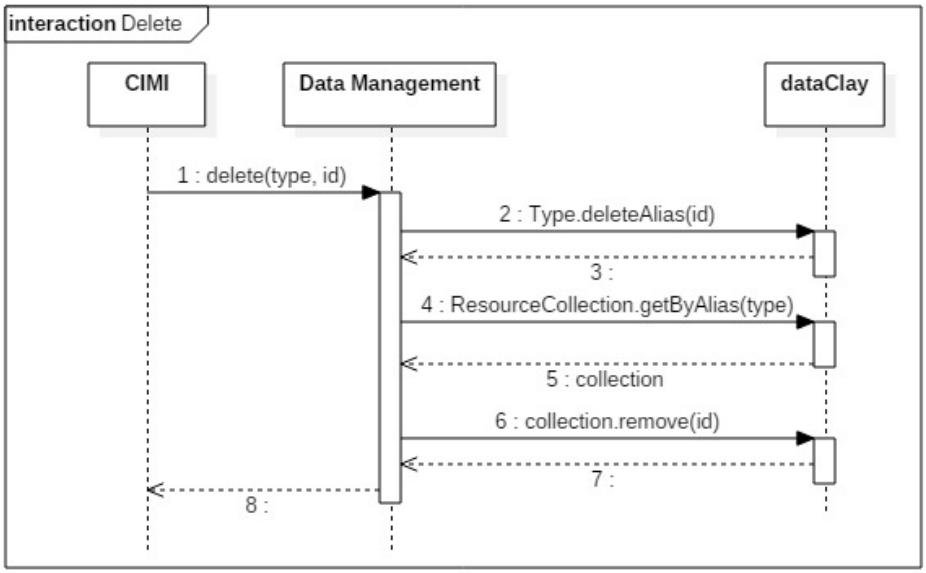


Figure 33 Data Management – Delete

To implement the DELETE operation, we delete in dataClay the alias of the object, and we also remove the corresponding entry from its resource collection. To do this, we first need to get a reference to the collection, and then remove the entry corresponding to the id. After the deletion, the object has no reference that points to it, which makes it inaccessible. Thus, the garbage collector in dataClay will effectively remove the object when the garbage collector is activated.

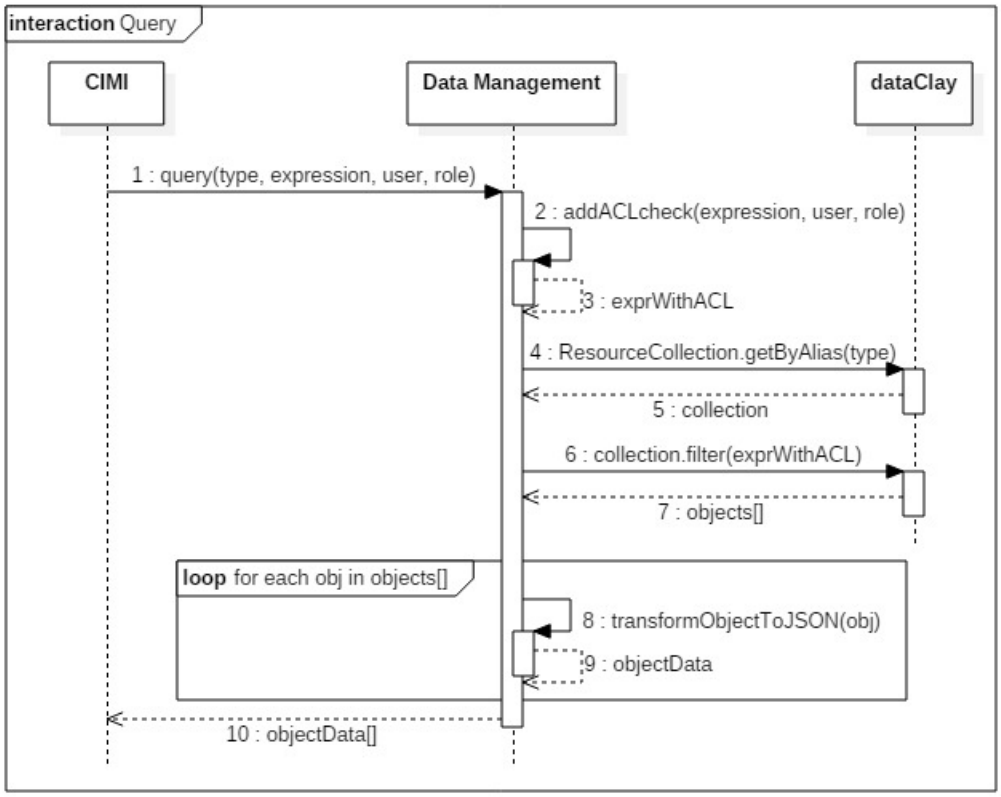


Figure 34 Data Management – Query

Finally, we have the diagram for the QUERY operation, which, from a collection of objects of the same 'type' (i.e. the collection of devices, of users, of services…), returns those objects that satisfy a certain condition and to which the user that issued the request has access. This has been implemented by

providing in dataClay a functionality that allows the Data Management component to filter a collection of objects according to an 'expression', with the access permissions stored in the data objects themselves. Thus, the first step is to generate an extended query expression that also includes the permissions check taking into account the user that executes the query, and his role. Then, the reference to the collection of objects belonging to 'type' is retrieved, and the extended query (the original 'expression' in conjunction with the acl check) is executed on it by invoking the filter method on the collection. The objects returned as a result, i.e. those objects that satisfy the initial query conditions and that are accessible to the user, are serialized into the JSON format expected by CIMI.

We have integrated the testing of all the Data Management workflows into a single test application that performs a sequence of CRUD operations. This test application has been successfully executed both in a regular laptop, as well as in a Raspberry Pi 3, with the Data Management component consuming around 10% of its 1 Gb RAM.

The application starts by creating a set of resources, reading their data from a set of JSON files provided as input. These files have the same format as the ones that would be received from CIMI.

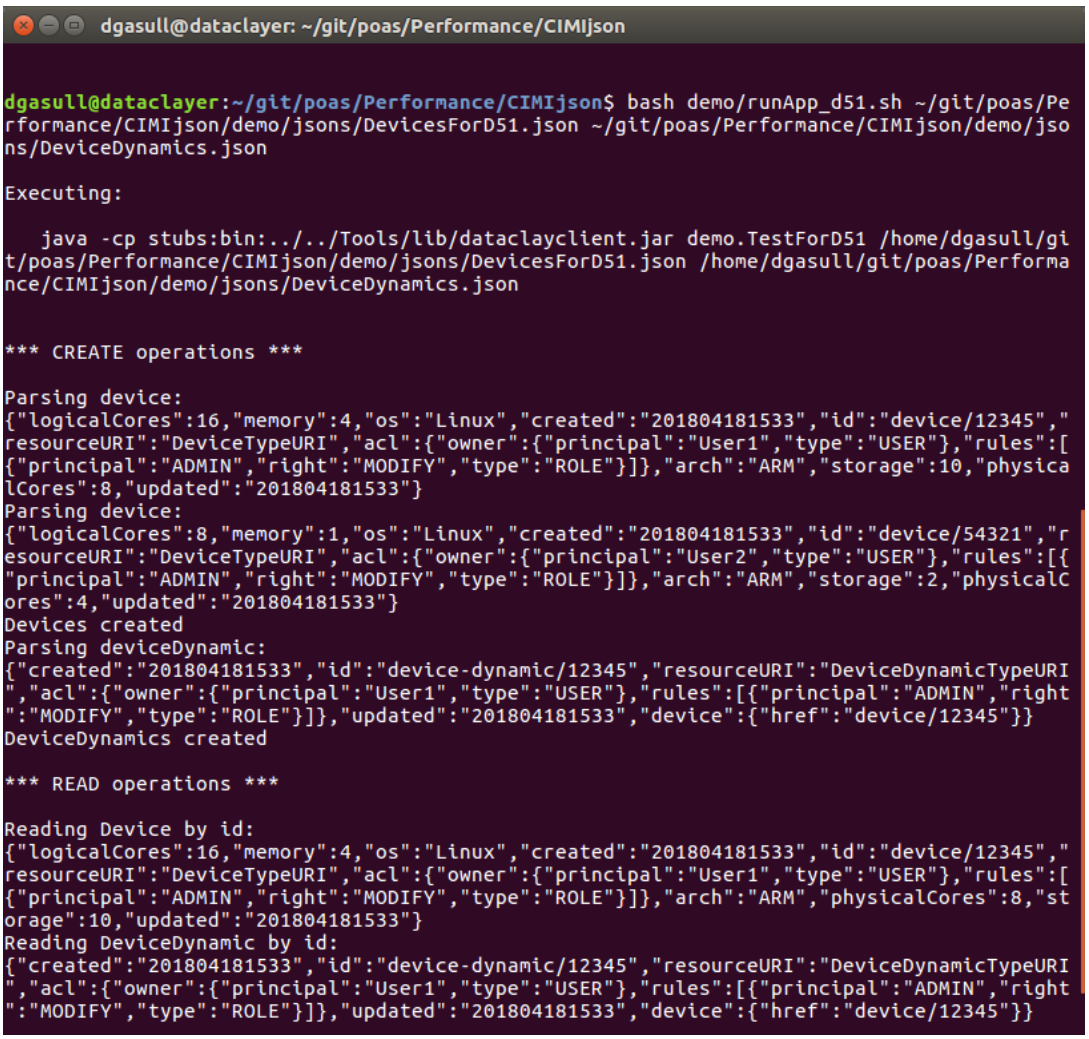


Figure 35 Data Management – Tests (part 1)

As can be seen in the output shown in Figure 35, under the title "CREATE operations", two resources of type "Device", and a resource of type "DeviceDynamic" are created and stored. The data contained in the new objects is shown as an unordered set of key-value pairs, where the key is the name of the property, and the value is the data it contains. For the sake of clarity, we are giving values only to a subset of

properties, since this does not affect the test results. In particular, we have created the Devices with identifiers “device/12345” and “device/54321”, and also the DeviceDynamic with identifier “device-dynamic/12345”. All these devices have been created by User1, who does not have an ADMIN role. A set of READ operations, that request to the database an object by its identifier, are executed afterwards. We request for Device “device/12345” and also for the “device-dynamic/12345”, and return all the data they contain in the form of a JSON document to be returned to the user as a result of his request. The test continues in next figure, were we show the UPDATE, QUERY and DELETE operations. We UPDATE a couple of properties of “device/12345”, and read the device by using the previous operation to show that the values in “storage” and “os” have been correctly modified.

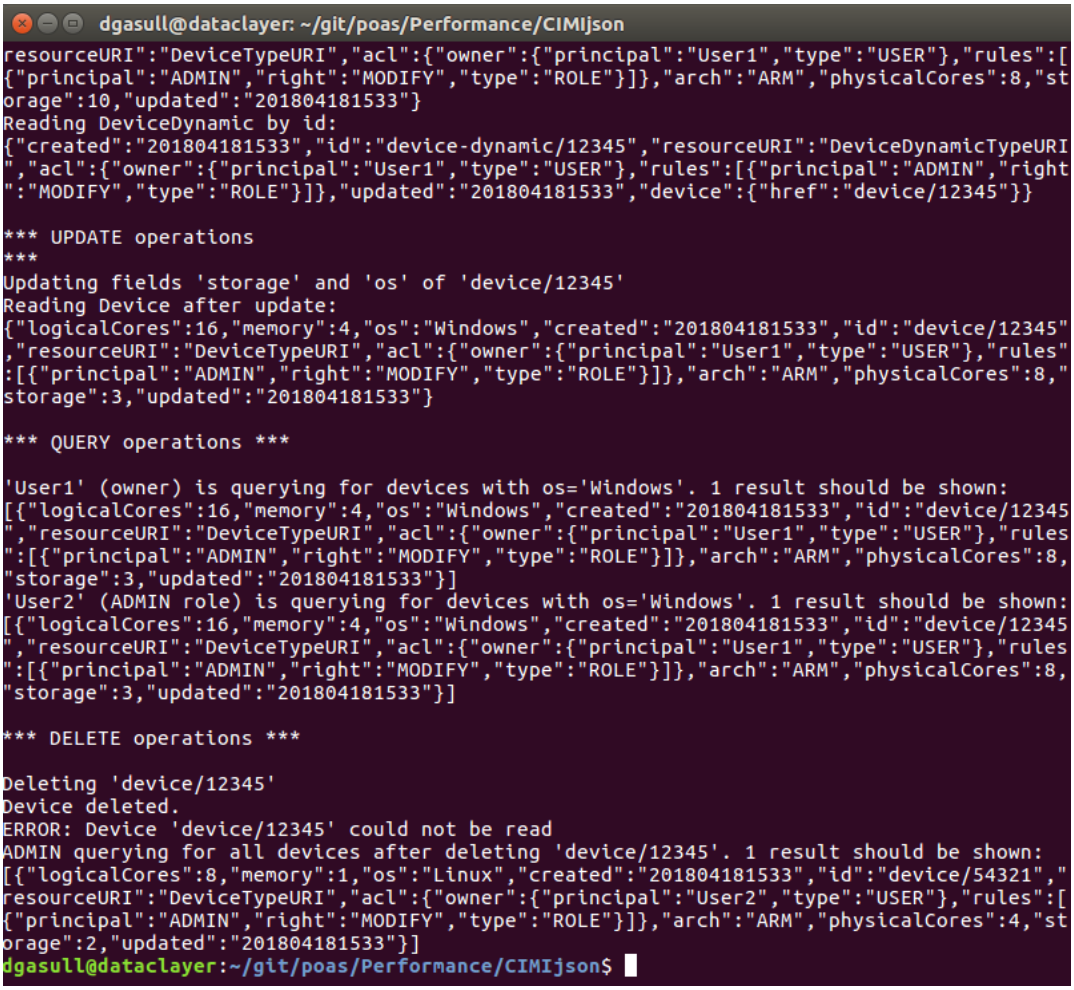


Figure 36 Data Management – Tests (part 2)

Now we perform some QUERY operations, testing with different permissions. As defined by CIMI, each resource has an owner, as well as a set of users/roles that can access it. For instance, when creating the devices, we have stated that the first one has been created by User1, who is a regular user, and the second one by User2, who has ADMIN role. Both devices can be accessed by any user with the ADMIN role, regardless of who created them, as happens with all resources in the mF2C platform. The first query is made by User1, who is the owner of the resources and asks for the devices with “os=Windows”. Only “device/12345” is correctly returned as a result of the query, since it’s the only one that satisfies the condition, and is also accessible by User1. Afterwards we can see that if a user with ADMIN role (User2) performs the same query, the result is the same, since we have established that all resources can be accessed by any user with this role. Finally we DELETE “device/12345” and see that if we request to read it by id, an error occurs, as expected. Also, if we list all the resources by issuing an empty query, then the deleted resource is not shown.

3.13 Analytics

The implementation of Service Analysis and the characterization of services have not changed much. For a previously deployed service, a subgraph is queried from the Landscaper and the associated telemetry of each of the elements of the graph queried. A characterization of that service is generated and the initial recipe used for deployment is updated in the Service Manager component.

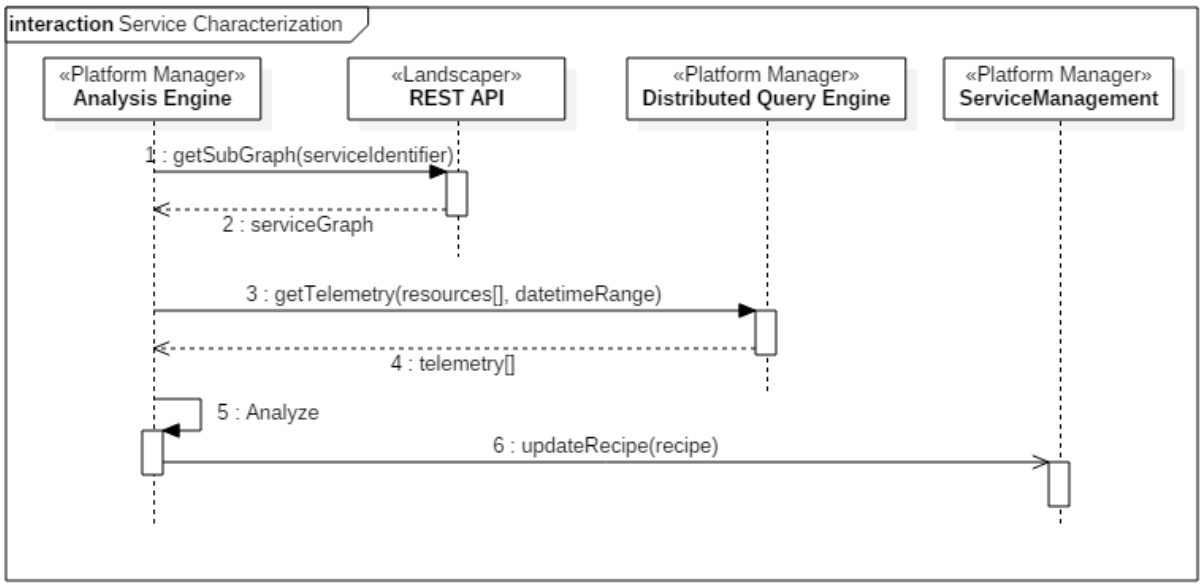


Figure 37 Service characterization

The real-time Service Performance analysis follows the same flow as Figure 37 above, with the exception that the metrics being queried are the latest and most current. The output of the task is to update the recipe so Lifecycle Manager is notified that a newer version of the recipe is available if it wants to issue service replacement.

4. PoC Description

This section introduces the first prototype delivered within the mF2C project, in terms of a PoC that is validated and evaluated as a result of the whole IT-1 period. As mentioned in the previous section for the workflows, the set of functionalities to be included in the IT-1 PoC has evolved as the project progressed, to adopt new functionalities, and thus improving the IT-1 PoC.

The IT-1 PoC may be described as two main conceptual components, the first referring to the set of assumptions adopted in IT-1 and the second referring to the set of functionalities included in IT-1.

Assumptions for IT-1

All assumptions and considerations for IT-1 are listed below. Certainly, IT-2 will cover the missing aspects as well as new aspects derived from practical experiences. We must observe that the main objective for IT-1 is to demonstrate the integration of the different elements of the mF2C architecture, with no need to evaluate performance characteristics or very complex systems or algorithms. Moreover, it is expected that the process of blocks integration along with the trials done to validate the first version of the architecture will drive improvements of the architecture as well as other fine tunings in IT-2.

- Services are executed from the mF2C dashboard
- The dashboard includes the portfolio of services categorized into different categories (IoT, data, smart cities, health…). Dashboard development is aligned with the definition of a service in CIMI, creating a JSON file compatible with the Service Manager
- For the sake of simplicity, in IT-1 the mF2C architecture considers three layers (cloud/fog/edge). All devices included in these layers deploy the mF2C Agent (note that cloud services are not devices and need not run agents.)
- There is no horizontal communication (between devices in the same layer) among devices at control level. The communication is multilayer and vertical.
- There is one leader and one backup selected in each fog area
- There is a limited set of categories for resources/services
- The processes of leader and backup selection need only to be very basic
- Clustering policy set manually at bootstrap.
- All services and mF2C functionalities fill a single container each
- A "recipe" = (#cores, core type, storage, #IoT need, IoT type)
- The recommender matches the service characteristics (obtained by the Service Categorization module) as well as the analytics from previous executions (Analytics module) to generate the Recipe
- There is neither allocation nor mapping at the Agent Controller
- mF2C is a software-only agent (no specialized hardware) downloaded from the mF2C (web service) and installed at the registration process
- No QoS enforcing will be done by the QoS providing block. This block in IT-1 feeds the Lifecycle Manager with information about resources' suitability to execute a specific service, based on the SLA violation history received from the SLA management.
- The SLA is set manually.

- The dynamic resources information is obtained from COMPSs notifying the Lifecycle Manager, when a task is done and the latter reporting to dataClay. The categorization module will also enrich this information.
- The sharing block includes (cores, memory, max apps, GPS, battery limit, BW, storage)
- Device categorization includes:
    - Hardware: the device is static (e.g. run after the discovery process)
    - Hardware: the device is dynamic (run according to a certain policy)
    - IoT: the device is manually introduced
- COMPSs does not change the resources to be used to execute a task from those requested in the recipe
- The SLA management block detects violations
- Application data may be stored in dataClay or in its own database
- The set of IDs (user and devices) are generated at cloud (mF2C cloud provider) at registration time
- A leader failure will not occur during app execution
- The interface with the mF2C agent is managed by CIMI.

Functionalities for IT-1:

The set of functionalities included in the IT-1 PoC matches the set of workflows described in the earlier sections. The functionalities involved in IT-1 are specified in section 2.2.

As described in the assumptions, the interface with the mF2C agent is managed by CIMI.

It is worth noticing that the set of blocks included in IT-1 are sufficient to make the system work and thus to validate the use cases included in the project.

4.1 General Functionality Demonstration

This section describes the strategy used to demonstrate and thus validate the proposed PoC. It must be highlighted that most of the different functionalities and technical contributions have been individually demonstrated and validated through the different scientific publications the consortium has already delivered, or simply by being tested independently. Therefore, in this deliverable we describe the strategy proposed to demonstrate IT-1 integration (as mentioned above IT-1 focuses on integration rather than on optimization).

In IT-1, the demonstration strategy is divided into two main categories:

- Individual or combined mF2C functionalities: Some of the mF2C functionalities are demonstrated independently of the execution of an mF2C service. For the sake of illustration, the tested functionalities are: i) the registration process (both of a user and also of a new mF2C service); ii) the discovery of a leader by an agent in the vicinity and mutual authentication between them, and; iii) the failure of a leader. It is worth mentioning that although identified as individual functionalities, they are not deployed as a unique feature, as shown in the set of workflows in section 3.
- Execution of mF2C services: The mF2C agent will be demonstrated in the three different use cases included in the project. Since not all functionalities will be deployed in IT-1 for the three proposed use cases, we have also proposed an additional use case, referred to as “Hello world”, aimed at representing a generic mF2C service, using all mF2C functionalities linked to the execution of a service.

4.1.1 Registration, Discovery and Leader failure

This section refers to the different set of functionalities individually demonstrated in IT-1, which corresponds to the workflows: Registration, Discovery and authentication and Leader Failure.

4.1.1.1. Registration

As described above, the first functionality to be demonstrated is the registration process for both users and services (as a part of the "Hello World" use case). This functionality is deployed in the testbed located at the UPC lab, using the mF2C dashboard and a device willing to get registered.

User registration:

A user can register a device with mF2C software pre-installed by connecting to the mF2C dashboard offered by an mF2C provider and filling in the required information. During the registration process the user gets specific credentials (userID and device ID), so the device can comply with the security policy later required for the discovery process. In the user registration process the following mF2C characteristics (matching the mF2C functional blocks) are shown:

- Security
- User Identification (ID Key)
- Device Identification (Device ID)
- mF2C agent installation on a device
- Initialization of the User Profile (Profiling)

Service registration:

The mF2C provider offers services to potential mF2C clients through a service catalogue available through the mF2C frontend. In this part, we show the procedure to register a new service in the catalogue. This includes the service registration itself but also a preliminary service categorization, to facilitate the tasks of the full service categorization and recommender blocks. The demonstration process is based on the use case that a service developer wants to upload a new service to the mF2C services catalogue. To this end the developer will upload the new service in the system and also provide the main requirements for running the service (such as CPU speed or IoT types and needs etc.), which will help the full service categorization later on. This process illustrates the following mF2C characteristics:

- Service categorization by the service developer
- Service uploaded to the service catalogue
- mF2C service catalogue

4.1.1.2. Discovery and Authentication

This functionality, as discussed earlier, refers to the process where a device becomes aware of the existence of a nearby mF2C leader. This discovery process assumes a Wi-Fi scenario in IT1, where the user/device scans the beacons broadcast by the leader within its proximity. The beacon message, a kind of welcome message, contains the necessary mF2C information for the user/device to kick off the joining process.

The mutual authentication process allows the leader to know if the agent in the user/device is a trustworthy, and, vice versa, allows the device to know if the leader is trustworthy. The authentication process uses X.509 certificates as credentials; these are issued by an mF2C CA (Certification Authority) residing in the cloud. A fog-based CAU (Control Area Unit [11]) acts as a gateway to the cloud CA, as devices connecting to the fog would not have access directly to the cloud, let alone the Internet as a whole. The

agent requests a certificate for the fog area that it wants to join by sending a CSR (Certificate Signing Request), generated by the local CAU Client, to a regional CAU with the necessary information identifying the target leader agent and itself. The CAU forwards the CSR to the CA and verifies the requestor's identity with the target leader agent's CAU. The signed certificate from the CA is returned via the regional CAU to the agent. To complete the authentication process, the new agent exchanges credentials with the agent leader through a TLS (Transport Layer Security) handshake.

To be precise, there are two CAs: one for issuing infrastructure certificates, which are long-lived (1-3 years) and can be revoked, and another for issuing the certificates to the agents. In IT-1, certificates for agents should be short-lived (on the order of a week, say), to avoid having to implement a revocation process. The initial trust-anchor distribution is achieved by including the CA certificate in the distribution of the mF2C agent software (i.e. the agent has it prior to connecting to the fog, so can validate the leader's identity and that of the CAU.)

To summarize, the Discovery and authentication processes demonstrate:

- A user/device is recruited by a leader in a completely secure process, initiated through the welcome beacon messages
- Secure interactions between the distributed local CAU client, regional and leader agent CAUs and cloud CA.
- Implementation of a trusted PKI (Public Key Infrastructure) within a fog2cloud environment.

4.1.1.3. Leader Failure

This functionality refers to the handover process when a leader goes down or gets inaccessible. As defined in the workflow (see Figure 10), the leader is a key component of the mF2C fog, hence the architecture is designed to keep leaders alive against common eventualities such as intermittent network connections, etc. In practice, we cannot possibly cater to every single situation that could bring a leader agent down. Hence, mF2C builds in a mitigation action based on selecting a backup leader when the main leader is chosen during the selection process, even if, in IT-1, the process is fairly simplistic. Indeed, we have already experimented with alternative approaches like setting flags for device availability, using a first fit policy linked for example to the lowest IPs in the linked address block, and using other simple characteristics). In the short term, we consider using a backup leader a good compromise to deal with potential leader failures. The process relies on dumping the current state of the running fog instance (i.e. the leader's database) to the backup leader when the leader fails; the alternative would be to rebuild the metadata from the agents in the leader's domain.

In short, the leader failure demonstrator shows how information is synchronized from the leader to the backup leader to guarantee an efficient handover process with minimal disruption to the running fog instance.

The backup leader periodically check the leader agent's status by keep-alive pings and when it detects a break in the keep-alive communication, the backup leader becomes the new leader and a new backup is simultaneously selected. The process shows how the system reacts and how information is synchronized.

4.2 Use Cases

4.2.1 Use Case 1

The first use case, emergency management, is an alarm manager for smart infrastructure. The main services of this use case will be (a) decision-making according to an inclination sensor that monitors emergencies in infrastructures and (b) to provide Emergency Situation Management in a Smart City context by processing information and triggering the intervention of the relevant emergency services. Several services are combined, using both micro agents (such as LoadSensing and the Jammer Detector) and smart

agents (Gateway). LoadSensing is a commercial solution proposed by Worldsensing for connecting and wirelessly monitoring infrastructures in remote locations. Construction and mining companies and operators of bridges, tunnels, dams, railways and many other inaccessible assets thus have access to this information, and real-time insights enables operators to anticipate needs, manage their workforce, diminish risks, and even prevent disasters. LoadSensing allows services to monitor the correct behavior of the infrastructure. If a sensor reports a value higher than an alarm threshold, the alarm manager will report an emergency situation to the cloud software that will trigger its alert methods. Furthermore, in order to improve the solution's security, the alarm manager is able to detect whether the LoadSensing and the Gateway are communicating (using a LoRa interface) with each other. If lack of communication is detected, the Jammer detector is automatically powered up and configured to detect jammers in the channel used for the LoRa communication.

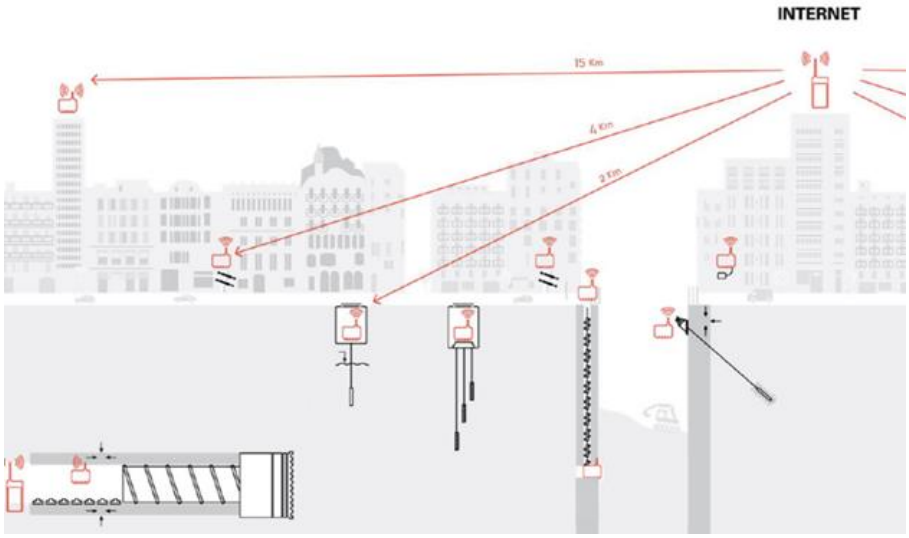


Figure 38 LoadSensing for Infrastructure monitoring

The use case consists in analysing sensor measurements. The tilt-meter sends inclination information to the Gateway, which relays this information to the cloud. This data is both analysed on the Gateway (on site) and on the cloud, where it is compared to the pre-established thresholds. If a critical situation is detected, an alarm service is notified and the alert protocols are started. In addition, if communication between tilt-meters and the Gateway is lost, the Jammer detection service is activated to detect potential attacks to the solution.

The data flow in the use case is the following:

- LoadSensing to Gateway: The LoadSensing gets sensors information and sends this information to the Gateway (via LoRa), which gets this information and stores it. It is also possible that the Gateway sends configuration messages (via LoRa) to the LoadSensing to set the desired characteristics. This is a periodical data flow.
- Jammer detector to Gateway: The multi-interface Gateway passes the parameters needed by the Jammer detector (over Ethernet), in order that the latter may locate a potential jammer easily (for example the frequency and the channel used by the LoadSensing tilt-meter that is having problems). The Jammer detector handles all of the SDR (Software Defined Radio) information and transmits (over Ethernet) the final decision (whether a jammer has been detected or not) to the Gateway.

- If the information provided by LoadSensing is identified as an alert and does not come from a communication issue, an emergency message is triggered to activate human the intervention. The relevant emergency services are started in order to get a quick response to the critical situation.

Regarding possible loss of communication alarm cases, there are two different data flows:

- When messages are not received from the LoadSensing, the Jammer detector is powered on and a jammer is not detected, an alarm is given that there is a problem with the LoadSensing datalogger but that it is not an attack.
- When messages are not received from the LoadSensing, the Jammer detector is powered on and a jammer is detected in the LoRa bandwidth. An alarm is raised because a jammer is blocking communication between LoadSensing datalogger and the gateway.

4.2.2 Use Case 2

For UC2, XLAB is developing a Smart Boat application for gathering, processing and sharing boat sensor or other data on the fog level. The Smart Boat concept enhances the boat monitoring experience provided by projects such as Sentinel Marine Solutions, which UC2 uses as sensor hubs besides directly connected sensors. The core concept for the enhancements on the fog level is the creation of fog fleets, which are groups of boats that can communicate between each other when in range of LoRa or Wi-Fi channels. The numerous enhancements enabled by it are grouped into 5 major user functionalities of the planned end product, where only 2 will be demonstrated for IT-1:

- Continuous Boat Monitoring
- Sensor Control

Both boat monitoring and sensor control is about providing the user with fresh boat data and control of sensors no matter the location. They also include alert systems for user-set limits, e.g. low fuel reserves. Both functionalities are already available in most hub sensors, but UC2 adds to them a fog level of execution in the fleet. Anomaly Detection and Data Plan Sharing on the other hand are only possible due to the concept of fog fleets. The former processes data provided by the sensors in the fleet to compare the values with local values to detect anomalies with either the boat or the sensor. The latter uses the fog channel between the boats, if access has been granted, to relay data to the cloud from another boat that is low on available data transfers or out of range of mobile connections. The last enhancement, Online Docking & Anchoring Reservation, is for simplifying the bureaucracy of docking and anchoring beforehand, while also providing an authentication system in the harbor. The true value of all functionalities is seen when you consider the end users of the application. There are two types of users inside Smart Boat application: owners and users. The distinction is notable in case of charter fleets, where we have one owner renting their charter boat/s to (multiple) users (at the same time).

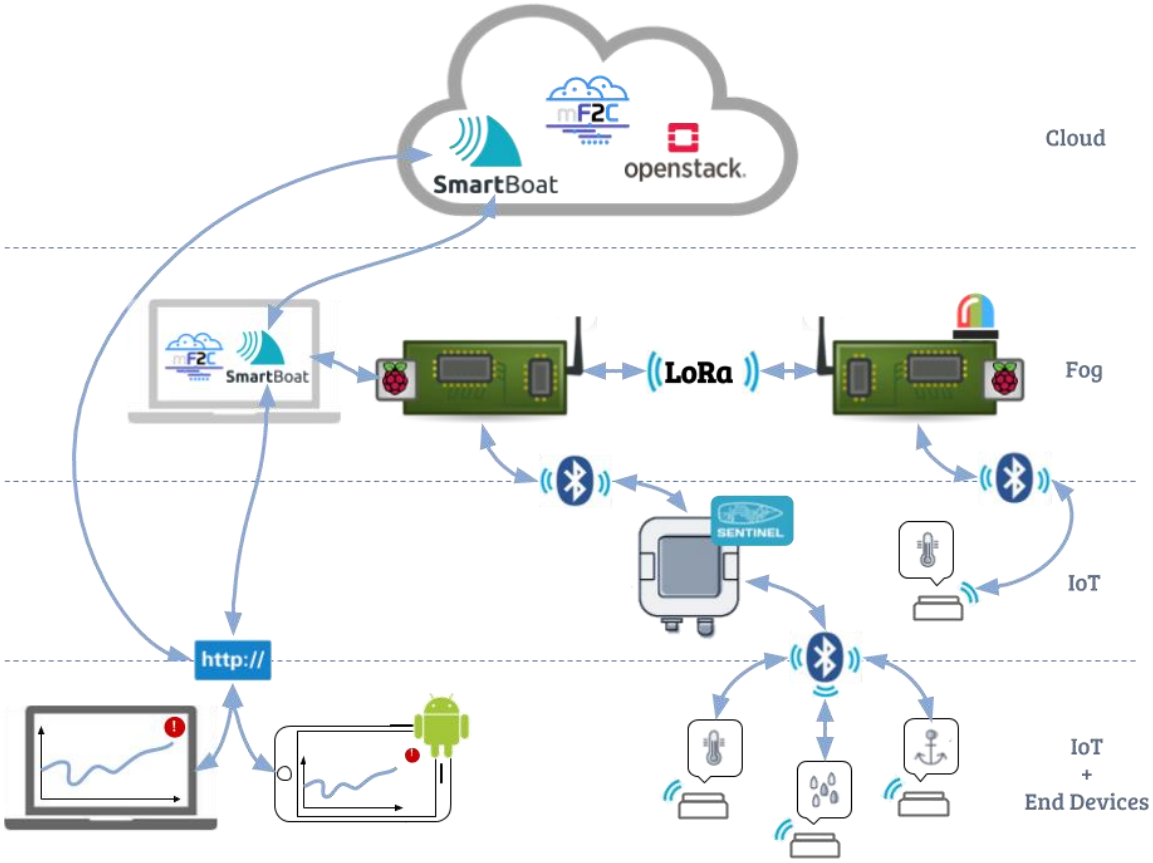


Figure 39 Use case 2 system architecture

Figure 39 presents Smart Boat deployment including multiple layers in the F2C scheme, depending on the number sensor hubs sequentially linked. Top layer (Layer 0) is a cloud, based on OpenStack, containing the mF2C agent with the UC2 application. The mF2C agent and the UC2 application are also present in the fog layer below (Layer 1), where the fog fleets are positioned. Layers from 2 downward contain the IoT devices that can be either sensors that directly communicate with the upper layer or sensor hubs that collect data from sensors or hubs in the lower layer and report to the upper layer. From the application aspect, layer 2 is the last one to take into account since lower layers are handled by the hubs. The demo will use the Sentinel Marine Solutions as a sensor hub. In final deployment, most of the fog processing will be done by a RaspberryPi3 device [12]. To avoid premature optimization of the mF2C agent in IT-1, the fog layer will be backed up with laptop devices, so that the more complex requirements of the mF2C agent will be calculated on a more powerful device that communicates with the RaspberryPi3 that manages the fog fleet and sensor data collection and control. The demo will have two RaspberryPi3 enhanced with LoRa modules and LED lights for fog fleet simulation and hardware status reporting. For actual user interaction there are Android and WEB applications, which can communicate either with the cloud application or local fog applications using HTTP(2) requests for communications.

The demo data flows are:

- The Bluetooth bridge application on a RaspberryPi3 acts as a listener for the sensor hub or sensor connected to the device and reports the current sensor values to the Smart Boat app on the laptop.
- The communication between components of the UC2 application is handled via gRPC [13] HTTP2 requests, which also simplifies the division of components between the devices of the demo.

- The UC2 application main/central component stores the values in a local database, which we use as cache, and later synchronize to the cloud database.
- User interaction is done via a WEB or Android app locally or via cloud. Depending on whether the end device is connected to the local laptop or to the cloud, the corresponding (local or cloud) database is used to present the measurement history.
- Alternatively, the user sends a control signal via the GUI to the laptop, which transmits it to the RaspberryPi3, which, depending on the type of signal, reacts accordingly. E.g. a change in led lights is requested, which will trigger the corresponding LED component via gRPC on the RaspberryPi3.
- For functionalities such as anomaly detection, we do not need to access the cloud, but want to use the LoRa communication channel to retrieve data from local caches of each boat in the fog fleet. Similarly, as with user interactions, a request is send to a second RaspberryPi3 from the application in the laptop that uses the LoRa component of first RaspberryPi3 via gRPC. The second RaspberryPi3 sends a request to the main component on the laptop for the data of the second boat; the data is delivered via the LoRa component back to the application of the first boat. The main component of the first boat compares the value (which should be an average in multiple boats in fleet in actual environment) and reports an anomaly if the difference is too big.

4.2.3 Use Case 3

The Use case 3 is under development in the Engineering Labs, and will be moved to the Cagliari Elmas Airport next year. In the final configuration, the fog elements will be positioned in the field in order to create a grid for Wi-Fi coverage.

The current environment that will support the IT-1 review demo in Brussels is an adaptation of the environment into which UC3 will ultimately be deployed, where an open space on the Engineering campus is used to simulate shops and other points of interest (PoIs), and airport events are simulated. Due to the unavailability of the mF2C agent in RaspberryPi, in IT-1, the system architecture has been adapted and is composed of the following elements:

- A cloud layer, based on an OpenStack instance, wire-connected with the fog layers, that provide scalable computing power for machine learning algorithms used for the recommendation system;
- A first fog layer, which acts as aggregator, based on a NuvlaBox mini [14], equipped with 8 GB RAM, that provides real-time computing and storage resources to the edge elements;
- A second fog layer, with a laptop with 4 GB RAM running the mF2C agent that interoperates with the NuvlaBox, and acts as worker node, providing processing, resource and security capabilities to the IoT layer;
- The edge layer with six RaspberryPi3 with 1 GB RAM, without the mF2C agent, each of which acts as access element and provides session management and fast response to the edge devices;
- Android smartphones, used by the end-users, connected to the access nodes with Wi-Fi, and using an android app to be engaged with the system; in this phase, they are used as data generator.

The mF2C agent runs in all Cloud and Fog elements; this is not currently supported on RaspberryPi and android smartphones, and will be released for IT-2. The android app to be installed in the smartphone implements security and privacy features to preserve managed data both at rest and in transit, with a security level comparable to the ones adopted by the mF2C agent.

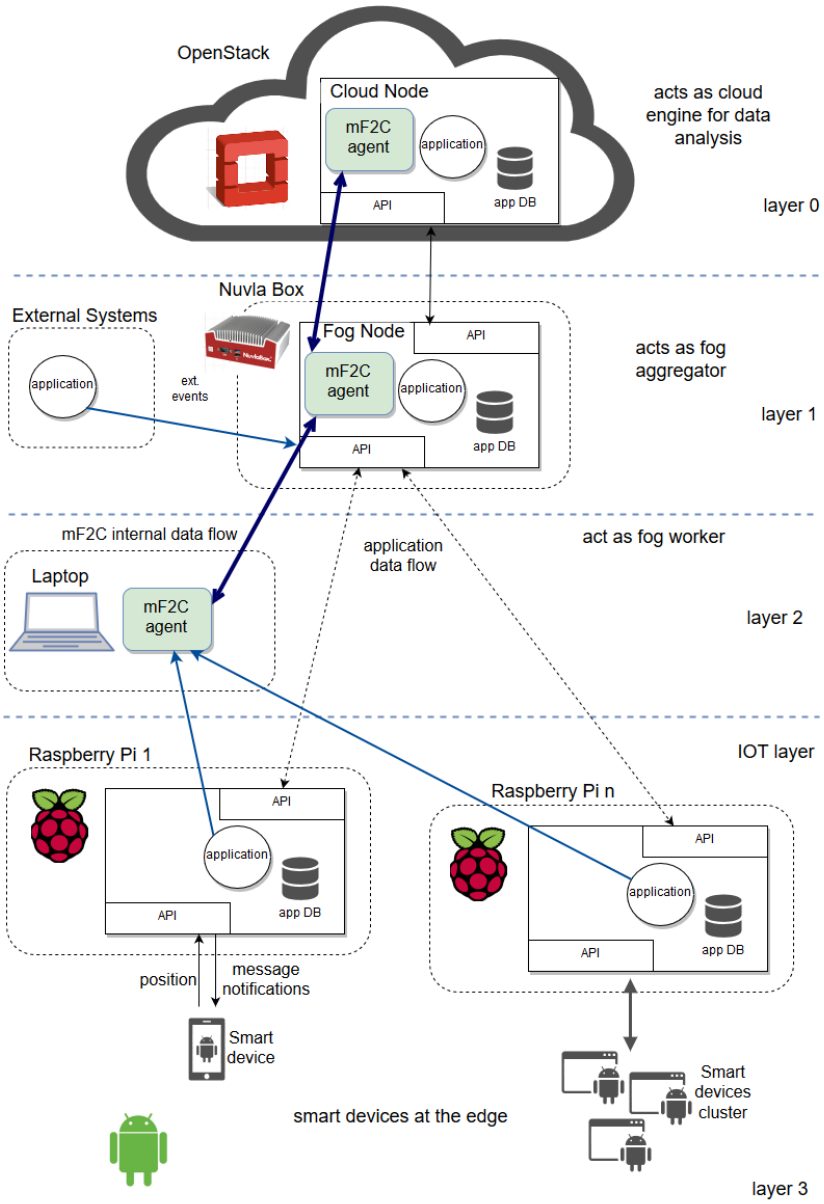


Figure 40 Use Case 3 system architecture

At the application level, the following business processes have been identified and are under development:

- App installation and device registration
- Position calculation, check for PoIs' proximity and user notification
- Position data sync in fog and cloud
- Airport events notification (flight call, but also invitation to move closer to the gate)
- Reporting (real-time and history) with the dashboard
- Management of PoI and promotions (in case of shops)
- Filtering and behavior calculation in positions streams (IT-2)
- Recommendations generation based on user similarities (and recalculations with data caching) (IT-2)

4.3 Service Catalogue – The Hello World

For the users that have been logged in the mF2C system, with their credentials validated, there is an additional option that allows them to register their services, as mentioned in Section 4.1.1.1. After login, a user will be able to register a new service with defined parameters based on its specific service requirements. The form for the definition of the specific service requirements in the mF2C provider webpage is shown in Figure 41.

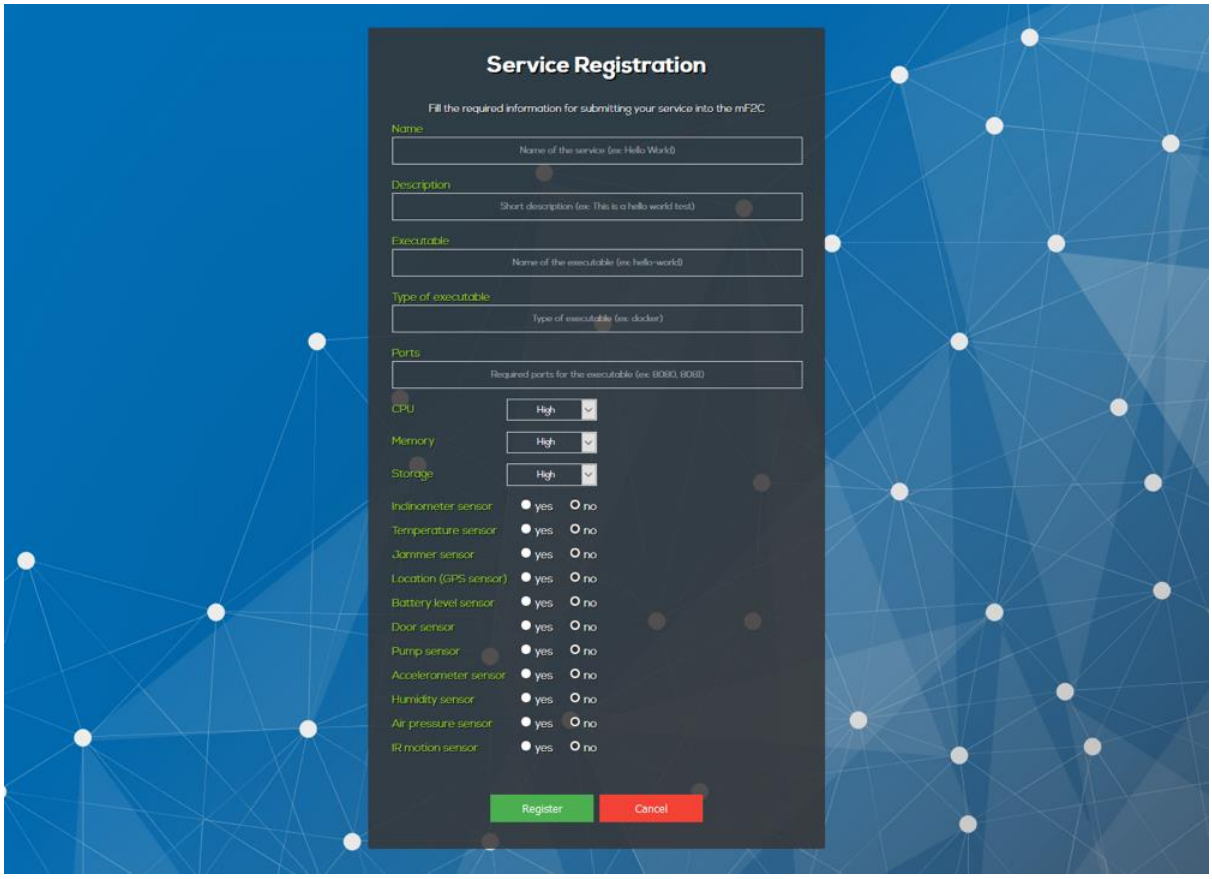


Figure 41 Registration of a new service

Beside description and the name of its service, the user can define the type of executable and a port that will be used. The other set of parameters, that include CPU, memory, storage as well as the information about the existence of certain sensors that are necessary for running a specific service, allow for services to be categorized later. For example, a user can specify whether their service has 'high', 'medium' or 'low' CPU, memory and storage requirements. At the moment, based on the three mF2C use cases, possible sensors include inclinometer, temperature sensor, jammer detector, location (GPS sensor), battery level sensor, door and pump sensor, accelerometer, humidity sensor, air pressure sensor and IR motion sensor. This list can be expanded later.

After defining the parameters, the user will click the submit button, which generates a JSON file that will be submitted to the mF2C system through CIMI. The example of JSON for a 'Hello World' service is shown in Figure 42.

```
{
  "name": "hello-world",
  "description": "Hello World Service",
  "exec": "hello-world",
  "exec_type": "docker",
  "exec_ports": [8080],
  "category": {
    "cpu": "low",
    "memory": "low",
    "storage": "low",
    "inclinometer": false,
    "temperature": false,
    "jammer": false,
    "location": false,
    "battery_level": false,
    "door_sensor": false,
    "pump_sensor": false,
    "accelerometer": false,
    "humidity": false,
    "air_pressure": false,
    "ir_motion": false
  }
}
```


Figure 42 Generated JSON for a 'Hello World' service example

In parallel with the submission of the service to the mF2C system, these services will be uploaded to the mF2C service catalogue. The list of the services from the catalogue for each registered user will be different and will depend on service ACLs. This functionality enables the users to access a number of already defined services and launch them from the catalogue, and not just register a new service. At the moment Figure 43 demonstrates a simple example of a service catalogue without taking into account a specific user. This example includes a catalogue that consists of the 'Hello World' service and predefined use case services 'Emergency Management System', 'Smart Boats Application' and 'Airport Location System'.

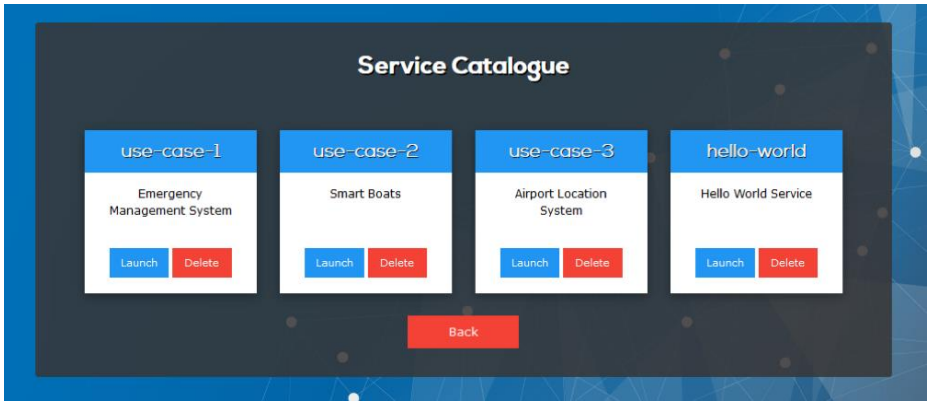


Figure 43 Service Catalogue

5. mF2C in IT-1

5.1 Testbeds

5.1.1 UPC-WoS Testbed

The testbed provided by UPC and Worldsensing will be used to validate the use case 1 (emergency management). This service is developed in a Smart City scenario with mF2C capacity (Figure 44), where logically we consider that there is an agent in the cloud, two leader agents in charge of two different areas (clusters) with other agents and/or IoT devices within these areas.

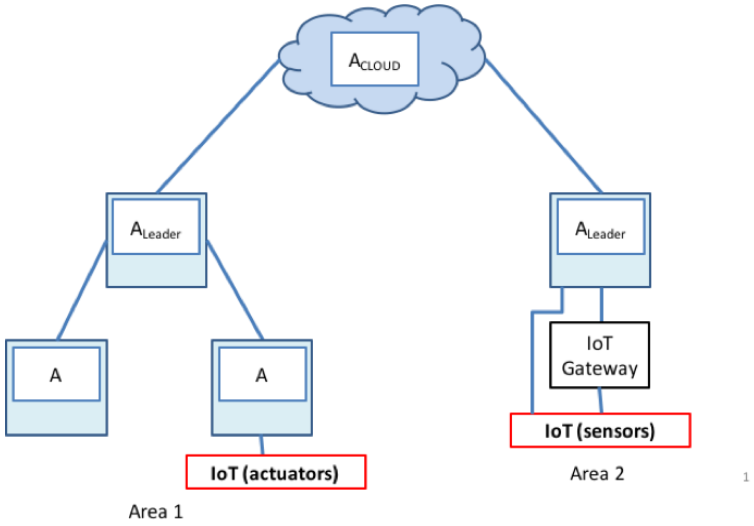


Figure 44 Topology of use case 1

The set of agents that make up the topology has a set of IoT devices associated with it that allow it to respond to an emergency service. The emergency service is emulated in the testbed shown in Figure 45, where 3 buildings are shown (red, pink and blue boxes):

- The sensorized building is to be monitored (red one in Figure 5.2).
- Hospital with the ambulance inside (pink one in Figure 5.2).
- Fire station with the fire engine inside (blue one in Figure 5.2).

Additionally, other elements are illustrated, such as:

- Roads and streets.
- Traffic lights.

In particular, the elements that will participate in this case are:

- Temperature and humidity sensor installed in the sensorized building.
- Jammer detector installed in (or close to) the sensorized building.
- Actuator installed in a traffic light → to change the status green/red.
- Actuator installed in a fire engine → to start/stop the fire engine.
- Actuator installed in an ambulance → to start/stop the ambulance.

In this scenario, we will consider two types of actions:

- Action produced by a Jammer attack that interrupts the connection between the inclinometer and the data reception center.
- Action produced by a seismic movement that causes a collapse of the building.

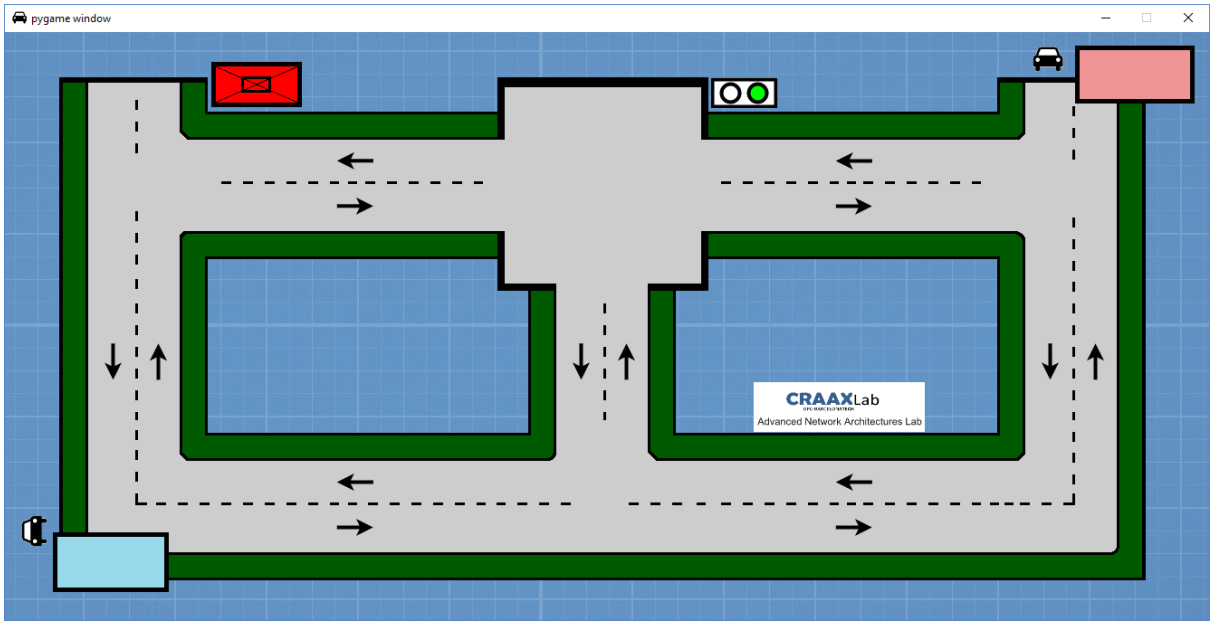


Figure 45 CRAAX Testbed

The two areas of action, shown in Figure 44, represent two separated zones of a Smart City, as shown in Figure 46.

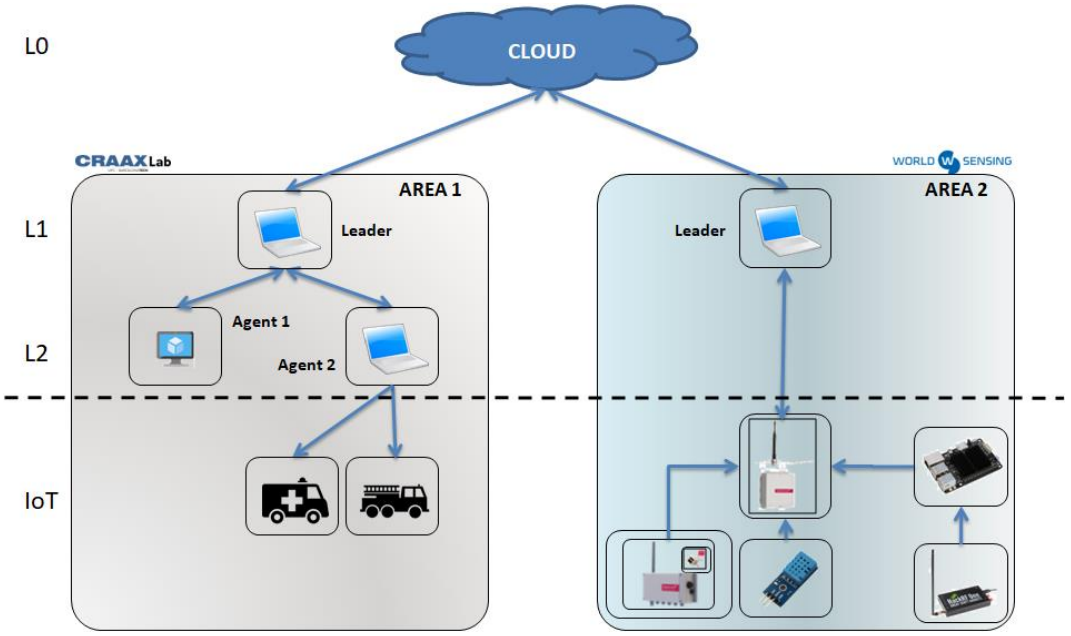


Figure 46 Infrastructure of use case 1

In the first area, we will have the part of the emergency service that responds to a trigger set off by an alarm event and the devices in this area are:

- Leader (laptop)
- Agent 1 (laptop)→ connected to an ambulance and to a fire engine.

- Agent 2 (laptop)→ to be used as a computation element.

The event detection itself is located in the second area 2, where the activation and the triggering of the emergency will take place due to the excessive inclination of a building produced by a seismic movement. It will consist of a leader and different IoTs connected to the leader through a gateway,

- Leader (laptop) -> connected to a gateway
- The Gateway is also connected:
- through wire to a Jammer detector.
- through LoRa to an inclinometer, which detects the seismic movement.
- through wire to a temperature sensor. In IT-2 we will use this sensor to detect also a fire in the building.

The two areas contain Agents in the Fog layer and are interconnected through the Cloud.

At the time of writing, the group of devices working together could be classified in two blocks:

LoadSensing's group

- Loadsensing's Inclinometer (Edge)
- LoadSensing (Edge/L2 LW Fog)
- Fog Kerlink Gateway (L1 Fog)

Jammer Detector's group:

- Jammer Detector's HackRf (Edge)
- Jammer Detector's ODroid C2 (L2 Fog)
- Cloud Server

5.1.2 XLAB Testbed

The Smart Boat's testbed consists of a set of hardware modules that mimic a small boat fleet and the fog and cloud layer above this. The devices intentionally are not mounted in a specific environment, as the tests require them to be a mobile set. A mobile set consists of:

- Sentinel Boat Monitor
- Raspberry PI
- LoRa module (optional)
- 3G/4G gateway (optional)
- Laptop (IT1 Only)

The Cloud part will be deployed on private cloud based on OpenStack.

5.1.3 ENG Testbed

The development of the Use Case3 infrastructure has been supported by a preliminary system environment in Engineering Labs, and will be moved later to the Cagliari Elmas Airport. This is based on Openstack for the cloud layer, one NuvlaBox mini (as fog leader/aggregator) and one Laptop HP (as fog worker). In the edge, 6 RaspberryPi are used for access management and position tracking of smartphones used by end-users.

The adaptations are related to the following points:

- Some open spaces are used as the airport lounge, where we simulate shops and other PoIs (Points of Interest);
- Airport events are simulated. The official timetable of the airport has been taken to create the full list of events related to departures with relevant events like open check-in, assign gate, call flight, last call, close flight/gate.

5.2 Orchestration and Installation

For IT-1, the mF2C system will be deployed through Docker Compose, having each component running as a service in its own container. By default, Docker provides enough portability, isolation, security and flexibility to enable the collaborative development of a modular architecture, where every component can execute on its own, without any core dependencies. At the same time, by using Docker, it will be possible to prove the IT-1 functionalities on multiple device types, without having to specifically build architecture specific components.

One of the downsides of such an approach will be resource usage, where the containers will not be sharing application resources like JVM, thus resulting in a final non-optimized IT-1 mF2C System. Optimization will not be targeted in IT-1.

The installation of the mF2C System will be provided through a single Docker Compose YAML file (version 3), which registered users shall download and install by running the command `docker-compose up`.

This YAML file will have one service definition per component, plus additional auxiliary services like Traefik. All the services will by default be deployed in the same Docker network, which allows the different components to find each other by name, while providing isolation from any other non-mF2C containers and system processes that might be running in the host device.

The requirements for deploying the IT-1 mF2C system are:

- Docker CE 17.12.0+
- Docker Compose 1.18.0+
- 2GB of RAM or more

5.3 Security Tests

5.3.1 Security test validation

In addition to validating the use cases, it makes sense to also validate the security testing methodologies, and, of course, the tests may usefully uncover unknown security holes.

The scope of these security tests has been restricted to the individual components, and obviously only those that were available for testing. The Use Case applications will be tested separately and the results documented in the future deliverable D5.3. For further background information on the planned security tests, the reader is referred to D2.4 [15]; for details of the actual tests, the reader is referred to the appendices.

As regards the software, for IT-1 the main goal is to demonstrate the feasibility of the foreseen components and functionalities of the mF2C system. Security testing of the software components therefore has no sense of “fail” because components and security are in a process of development. On the other hand there is a sense of “pass” in that it is considered already suitable for use, potentially even in a hostile environment.

The expected outcome of the security test is therefore:

- Discovery of known security vulnerabilities, because:
  - The protection is out of scope for IT-1, or
  - The security feature has not been implemented yet.
- Potential discovery of unknown or unexpected vulnerabilities.
- A validation of the testing methodology.

Furthermore, the results of the tests may be useful in their own right, for example for a third party wishing to reuse individual components of the current mF2C software.

5.3.2 Security test implementation

The two main tools used in these tests were Network Mapper (nmap [16]) and w3af, for testing, respectively, network (ICMP) and web protocols. Figure 47 shows our first example of nmap in use; it has correctly identified unexpected software listening on the smtp port and correctly identified it as Postfix.

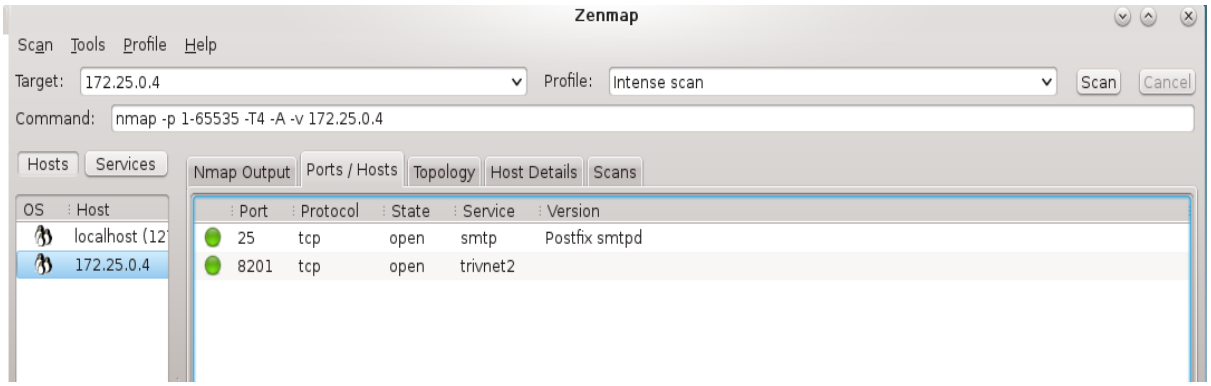


Figure 47 Example of nmap

In the second example of using nmap (), below, we can see the sort of information that nmap is capable of finding from ICMP scans. Note the SMTP (= email) server is listed here along with the commands that are “alive” on it plus the name of the container network FQDN.

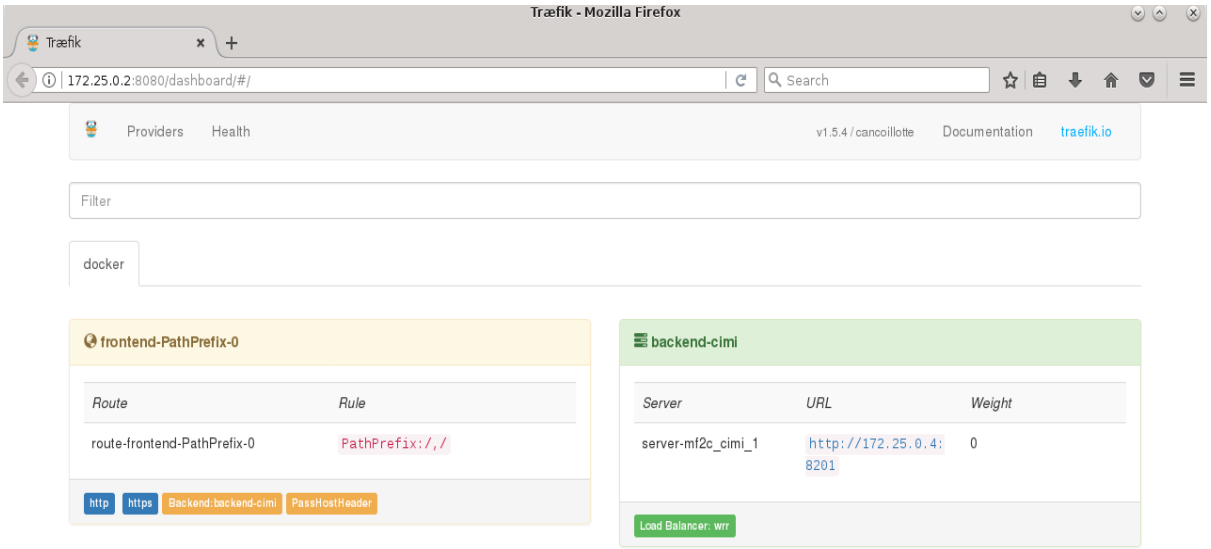


Figure 48 Traefik monitoring page exposed

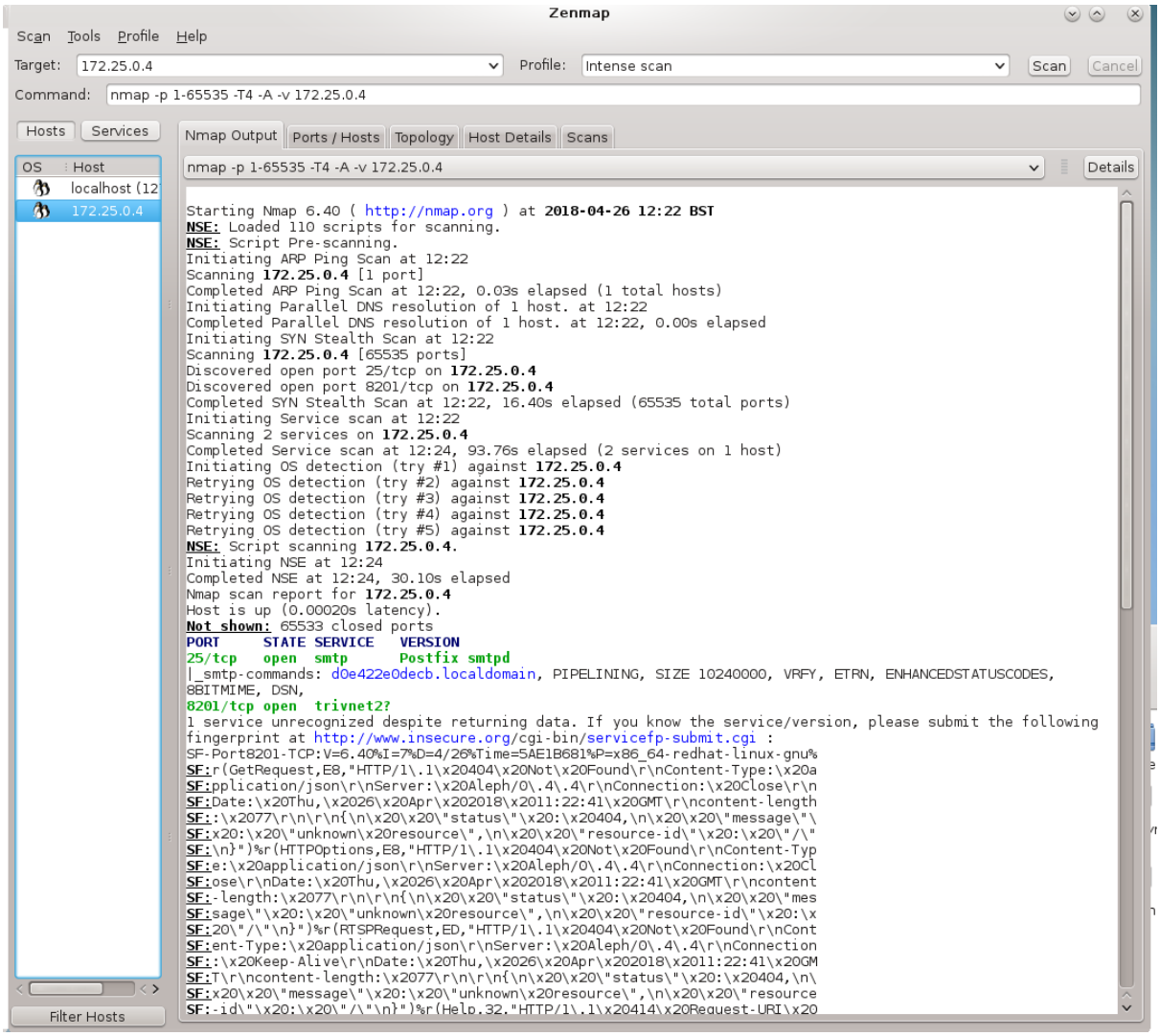


Figure 49 Example 2 of use of nmap

This is a screenshot of the exposed Traefik proxy server admin page. It is not visible from a remote location so anything exposed here is not a huge concern, but a compromised host or container could get useful information from it, such as the hidden network address of the CIMI server, plus the name and version number of the proxy server.

5.3.3 Overall result

Initial testing and analysis have uncovered a number of vulnerabilities, most of which were known beforehand – as they have either not been implemented yet, or are out of scope for IT-1. There are some actionable tasks from the security testing (see appendices for details):

- [KNOWN] There are no backups of data or systems software
- [KNOWN] There is no protection against denial of service attacks
- [KNOWN] There is no logging of security events and other important events
- [KNOWN] There is no audit trail of security events e.g. new user account
- [KNOWN] There is no alerting in real-time of security incidents in progress; in particular, there is no botnet protection.
- [KNOWN] Physical tampering of edge devices is possible.

- [NEW] The MQTT broker has no protection against snooping and fake data injection
- [NEW] Every user of the CIMI server has authorization set to “ADMIN”
- [NEW] The Docker network was misconfigured; exposing components to attack that had otherwise no protection. The real issue, however, is that there is no detection of misconfiguration.
- [KNOWN] No implementation of data privacy.
- [KNOWN] No implementation of data compartmentalization or at-rest encryption.

As can be seen, most of the results are quite obvious: protection against botnets and denial of service were not foreseen till IT-2; and, as mentioned above, the testing was done against a snapshot of the development and fixes for many of these issues are already in progress. However, rediscovering the vulnerabilities validates the testing and security analysis methodology.

5.3.4 Discussion

The IT-1 release is a proof of concept, and security was not a strong goal for this release as we are expecting only to validate functionalities. However, we note that some components already have security implemented, others are in the process of having them implemented. For a third party wishing to reuse our components, they obviously cannot reuse them in a hostile environment (on the open Internet, say, or in a foreign fog infrastructure) if the component has security vulnerabilities, or, in our analysis, we have found that a subcomponent has a vulnerability classified as **severe**. Similarly, as the current release is software-only, there is no physical protection of edge devices; it is left to the deployer to ensure that their devices are protected against intrusion.

For example, the lack of support for privacy, particularly at this time when the GDPR (D2.4 section 2.4.1 and Annex 8) is about to come into force, means handling personal data is not advised.

5.4 Testers Task Force

Testing mF2C components has been considered a primary task. This could help in increasing the quality of software components and fulfilment to requirements. For this purpose, a specific testing task force has been assembled, consisting of two software developers from Engineering.

The chosen people have competence in the mF2C software requirements and were involved later in the use of mF2C components to develop one of the Use Cases. This kind of organization has several advantages:

- Chosen people participated in the mF2C software design, so have a knowledge of specifications and expected behavior,
- These people are real users of the mF2C software components, which are going to use these components to develop the Use Cases, so this activity would serve as additional training for them, helping in acquiring more practical competence on mF2C software components.

A test report template has been defined and shared, to be used to document all tests. For each test case, some information on pre-conditions (environment, assumptions, etc.) and user to be impersonated, with related authorizations, have been registered, followed by the list of execution steps, expected and actual results, according to the following:

Tester: <user to be impersonated>

Pre-conditions: <all information regarding environment, assumptions etc.>

Test Case

Step	Step Description	Expected Result	Actual Result (if different from expected)	Successful /Failed (opt Comments)
1			-	Passed/fail
2				

Table 5. Test case table

So, once each mF2C component was released, the corresponding content was can loaded from the mF2C GitHub repository, and the test environment prepared according to the provided document. All information about the environment, the executor and authorization profile was tracked, the corresponding output included in the report, with the expected behavior. In case of different behavior, an analysis was performed and the outcome registered as well.

The summary of tests is in the table below with name of mF2C component, tester, date, outcome.

component	version	date	tester	purpose	outcome
COMPSs	v.1.4	13/dec/2017	Paolo Cocco	Installation test with Centos7 (with Docker installed) on the COMPSs have to verify the correct installation for the application	Not passed
COMPSs	v.1.4	13/dec/2017	Paolo Cocco	Installation test with Centos6 (with Docker installed) on the COMPSs have to verify the correct installation for the application	Not passed
COMPSs	v.1.4	13/dec/2017	Paolo Cocco	Installation test with Centos6 (with Docker installed) on the COMPSs have to verify the correct installation bypassing the packages sign	passed
COMPSs	v.1.4	13/dec/2017	Paolo Cocco	Create and Execute of an application image using Docker container with COMPSs	passed
COMPSs	v.1.4	13/dec/2017	Paolo Cocco	Create and Execute of an application image using Docker container with COMPSs and INSTALL ”realpath” command (centos6)	Not passed

COMPSs	v.1.4	13/dec/2017	Paolo Cocco	Create and Execute of an application using Docker container with COMPSs (Centos6). This test is execute in correct way just for the installation of “realpath” command for un external repository but failed for other error	passed
COMPSs	v.2.2	15/jan/2018	Paolo Cocco	Test the VM appliance, with the Hello World sample	passed
COMPSs	v.2.2	15/jan/2018	Paolo Cocco	Installation test with Centos6 (with Docker installed) on the COMPSs have to verify the correct installation	Not passed
COMPSs	v.2.2	15/jan/2018	Paolo Cocco	Installation test with Centos6 (with Docker installed) on the COMPSs have to verify the correct installation bypassing the packages sign	passed
COMPSs	v.2.2	15/jan/2018	Paolo Cocco	Create and Execute of an application image using Docker container with COMPSs	Not passed
COMPSs	v.2.2	15/jan/2018	Paolo Cocco	Create and Execute of an application image using Docker container with COMPSs and INSTALL ”realpath” command (centos6)	Not passed
COMPSs	v.2.2	15/jan/2018	Paolo Cocco	Create and Execute of an application using Docker container with COMPSs (Centos6). This test is execute in correct way just for the installation of “realpath” command for un external repository but failed for other error	passed

DataClay	v.1	30/jan/2018	Paolo Cocco	Installation test on the dataClay have to verify the correct installation for the application. Centos7 with Docker, java 8, python installed, have java class People, Person and HelloPeople described in the manual	passed
DataClay	v.1	30/jan/2018	Paolo Cocco	Installation test on the dataClay have to verify the correct installation for the application. One Server with Centos7 with Docker and One Client with Ubuntu, java 8, python installed, have java class People, Person and HelloPeople described in the manual.	Not passed - Dataclay tries to contact the Docker image instead of the host
Service Manager	v.1.1.2	1/dec/2017	Paolo Cocco	Installation test for Service Manager component have to verify the correct installation for the application. Ubuntu 16.04 vanilla with Docker installed, and java8	passed
Service Manager	v.1.1.2	1/dec/2017	Paolo Cocco	Installation test for the Service Manager have to verify the correct installation and dockerizing the application. Ubuntu 16.04 vanilla with Docker installed, and java8	passed

Table 6. Test results summary

Major issues have been communicated to the software developers for diagnostic purposes, and to speed-up the software fixing; some issues has been recognized as network misconfiguration between different dockerized components, so they were not able to connect each other.

An additional effort, devoted to the understanding of the use of the mF2C components in real applications like the Use cases, have been provided with a simplified version of the Use Case 3 main processing that, given a position, search the list of Points of Interest nearby. This chunk of code uses both the COMPSs and dataClay components. The resulting code has been validated by the COMPSs and dataClay software experts, then published in GitHub as a reference [17].

References

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

[2] mF2C, "mF2C Project Deliverables - D2.6," [Online]. Available: http://www.mf2c-project.eu/d2-6-m6/.

[3] mF2C, "mF2C Project Deliverables - D3.3," [Online]. Available: http://www.mf2c-project.eu/d3-3-m9/.

[4] mF2C, "mF2C Project Deliverables - D4.3," [Online]. Available: http://www.mf2c-project.eu/d4-3-m9/.

[5] Containous, "Traefik," [Online]. Available: https://traefik.io/.

[6] I. GitHub, "GitHub," [Online]. Available: https://github.com/.

[7] I. Docker, "Docker," [Online]. Available: https://www.docker.com/.

[8] mF2C, "mF2C Docs," [Online]. Available: http://mf2c-project.readthedocs.io/.

[9] J. Martí, A. Queralt, D. Gasull, A. Barceló, J. Costa and T. Cortes, "Dataclay: A distributed data store for effective inter-player data sharing," *Journal of Systems and Software,* vol. 131, no. 0164-1212, pp. 129 - 145, 2017.

[10] mF2C, "mF2C Project Deliverables - D3.5," [Online]. Available: http://www.mf2c-project.eu/d3-5-m12/.

[11] S.Kahvazadeh, V.Barbosa, X.Masip-Bruin, E.Marín-Tordera, J.Garcia and R.Diaz, "Securing combined Fog-to-Cloud System through SDN approach," in *4th Workshop on CrossCloud Infrastructures & Platforms (ACM Digital Library)*, Belgrade, 2017.

[12] R. P. Foundation, "Raspberry Pi," [Online]. Available: https://www.raspberrypi.org/.

[13] Google, "Remote Procedure Calls," [Online]. Available: https://grpc.io/ .

[14] SixSq, "NuvlaBox," [Online]. Available: http://sixsq.com/products-and-services/nuvlabox/tech-spec.

[15] mF2C, "mF2C Project Deliverables - D2.4," [Online]. Available: http://www.mf2c-project.eu/d2-4-m4/.

[16] "Network Mapper," [Online]. Available: https://nmap.org/.

[17] "dataClay demo," [Online]. Available: https://github.com/mF2C/uc3-compss-dataclay-demo.

Appendix 1: Start Application request

Content of a Start Application request in the Distributed Execution Runtime:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<startApplication>
   <ceiClass>es.bsc.compss.test.TestItf</ceiClass>
   <className>es.bsc.compss.test.Test</className>
   <methodName>main</methodName>
   <parameters>
      <params paramId="0">
         <direction>IN</direction>
         <type>OBJECT_T</type>
         <array paramId="0">
            <componentClassname>java.lang.String</componentClassname>
            <values>
               <element paramId="0">
                  <className>java.lang.String</className>
                  <value
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">3
                  </value>
               </element>
            </values>
         </array>
      </params>
   </parameters>
   <resources>
      <resource name="COMPSsWorker01:8080">
         <description>
            <memorySize>4.0</memorySize>
            <memoryType>[unassigned]</memoryType>
            <operatingSystemDistribution>[unassigned]</operatingSystemDistribution>
            <operatingSystemType>[unassigned]</operatingSystemType>
            <operatingSystemVersion>[unassigned]</operatingSystemVersion>
            <pricePerUnit>-1.0</pricePerUnit>
            <priceTimeUnit>-1</priceTimeUnit>
            <processors>
               <architecture>[unassigned]</architecture>
               <computingUnits>1</computingUnits>
               <internalMemory>-1.0</internalMemory>
               <name>[unassigned]</name>
               <propName>[unassigned]</propName>
               <propValue>[unassigned]</propValue>
               <speed>-1.0</speed>
               <type>CPU</type>
            </processors>
            <storageSize>-1.0</storageSize>
            <storageType>[unassigned]</storageType>
            <value>0.0</value>
```

```
            <wallClockLimit>-1</wallClockLimit>
          </description>
        </resource>
        <resource name="COMPSsWorker02:1200">
          <description>
            <memorySize>4.0</memorySize>
            <memoryType>[unassigned]</memoryType>
            <operatingSystemDistribution>[unassigned]</operatingSystemDistribution>
            <operatingSystemType>[unassigned]</operatingSystemType>
            <operatingSystemVersion>[unassigned]</operatingSystemVersion>
            <pricePerUnit>-1.0</pricePerUnit>
            <priceTimeUnit>-1</priceTimeUnit>
            <processors>
              <architecture>[unassigned]</architecture>
              <computingUnits>1</computingUnits>
              <internalMemory>-1.0</internalMemory>
              <name>[unassigned]</name>
              <propName>[unassigned]</propName>
              <propValue>[unassigned]</propValue>
              <speed>-1.0</speed>
              <type>CPU</type>
            </processors>
            <storageSize>-1.0</storageSize>
            <storageType>[unassigned]</storageType>
            <value>0.0</value>
            <wallClockLimit>-1</wallClockLimit>
          </description>
        </resource>
      </resources>
</startApplication>
```

Appendix 2: Single-server detailed security test results

Method of testing

We have tested the following components:

- cimi
- cimi proxy
- dataclay
- compss
- user-management

We have used the following tools to test the security of the components:

- nmap - for icmp-based port scanning
- w3af - for http-based vulnerability scanning

For the MQTT broker test we simply connected to the server without supplying any credentials and found we were able to snoop on traffic or send data on any mqtt topic.

The tests that were used by the w3af tool are listed at the end of this appendix.

Analysis of results

There was one severe vulnerability found – the MQTT broker has no protection against snooping and fake data injection.

No component or server has received a PASS rating.

Results

The column headings have the following meanings:

- Test description - the check that was performed
- Test result - the vulnerability - if found - is named. The results of the test are listed. If it was not tested then Not Tested is entered here.
- Impact and probability - best practice for evaluating risk is to score the impact of a problem, score the probability of the problem occurring and then multiply the two scores together. Here we have a more informal approach. We assign Low, Medium and Severe values to this column (impact and probability) then factor them together (in an informal way) to produce the next column, Severity.
- Severity - the calculated risk. We informally assign values of Low severity, Medium severity and Severe.

  Category - Analyzed into Confidentiality, Integrity and Availability.

Test description	Test result	Impact and probability	Severity	Category
Cimi server – check for cache control of https content.	Missing cache control for HTTPS content	Low impact. Low probability.	Low severity	Integrity

User-management component – check for cache control of https content.	Missing cache control for HTTPS content	Low impact. Low probability.	Low severity	Integrity
User-management component – check for click-jacking. This vulnerability tricks users into clicking on something that is not what they think it is.	Vulnerable to click-jacking.	Medium impact Low probability	Low severity	Integrity
CA server – check for cache control of https content.	Missing cache control for HTTPS content	Low impact. Low probability.	Low severity	Integrity
CA server – check for click-jacking. This vulnerability tricks users into clicking on something that is not what they think it is.	Vulnerable to click-jacking.	Medium impact Low probability	Low severity	Integrity
Restrict software that can be run on the host server without authorization	There is no restriction on what can run outside of the controls provided by Compss (i.e. at the operating system level). However local access would probably be required to run anything.	The software that runs might be malicious. Impact medium. Probability low.	Low severity (not demonstrated)	Integrity
Restrict software that can be run on the server via autorun on usb and cd drives	Autorun is not often enabled, but where it is an attack can be done easily if it is possible to get physical access to the server to insert a CD or usb stick.	The software that runs might be malicious. Impact medium. Probability low.	Low severity (not demonstrated)	Integrity

Attempt to connect to an MQTT broker, attach to a topic and listen in to data or send data without authorization.	The MQTT broker has no checks on what can connect to a topic and send or receive messages.	Fake messages can be sent. Confidential data can be snooped. Impact very variable. Probability medium	Severe.	Confidentiality. Integrity.
Check host server for vulnerabilities	The host server was found to have several open ports not used by mf2c but these were not serious vulnerabilities.	The host server must be checked in the same way as the containers. If the host server is compromised it can access the containers used by mf2c and view secrets or alter data. Impact high. Probability low.	Low severity	Integrity

The following test cases were performed using w3af:

w3af plugin name	Description
blind_sqli	Identify blind SQL injection vulnerabilities.
buffer_overflow	Find buffer overflow vulnerabilities.
cors_origin	Inspect if application checks that the value of the "Origin" HTTP header is consistent with the value of the remote IP address/Host of the sender of the incoming HTTP request.
csrf	Identify Cross-Site Request Forgery vulnerabilities.
dav	Verify if the WebDAV module is properly configured.
eval	Find insecure eval() usage.
file_upload	Uploads a file and then searches for the file inside all known directories
format_string	Find format string vulnerabilities.
frontpage	Tries to upload a file using frontpage extensions (author.dll).
generic	Find all kind of bugs without using a fixed error database.
global_redirect	Find scripts that redirect the browser to any site.

htaccess_methods	Find misconfigurations in Apache's "<LIMIT>" configuration.
ldapi	Find LDAP injection bugs.
lfi	Find local file inclusion vulnerabilities.
mx_injection	Find MX injection vulnerabilities.
os_commanding	Find OS Commanding vulnerabilities.
phishing_vector	Find phishing vectors.
preg_replace	Find unsafe usage of PHPs preg_replace.
redos	Find ReDoS vulnerabilities.
response_splitting	Find response splitting vulnerabilities.
rfi	Find remote file inclusion vulnerabilities
rosetta_flash	Find Rosetta Flash vulnerabilities in JSONP endpoints
shell_shock	Find shell shock vulnerabilities.
sqli	Find SQL injection bugs.
ssi	Find server side inclusion vulnerabilities
ssl_certificate	Check the SSL certificate validity (if https is being used).
un_ssl	Find out if secure content can also be fetched using http.
websocket_hijacking	Detect Cross-Site WebSocket hijacking vulnerabilities.
xpath	Find XPATH injection vulnerabilities
xss	Identify cross site scripting vulnerabilities.
xst	Find Cross Site Tracing vulnerabilities.
afd	Find out if the remote web server has an active filter (IPS or WAF).
allowed_methods	Enumerate the allowed methods of an URL.
detect_reverse_proxy	Find out if the remote web server has a reverse proxy.
detect_transparent_proxy	Find out if your ISP has a transparent proxy installed.
dns_wildcard	Find out if www.site.com and site.com return the same page.
domain_dot	Send a specially crafted request with a dot after the domain(http://host.tld./) and analyze response.

favicon_identification	Identify server software using favicon.
find_jboss	Find default Jboss installations.
find_vhosts	Modify the HTTP Host header and try to find virtual hosts.
fingerprint_WAF	Identify if a Web Application Firewall is present and if possible identify the vendor and version.
fingerprint_os	Fingerprint the remote operating system using the HTTP protocol.
frontpage_version	Search FrontPage Server Info file and if it finds it will determine its version.
halberd	Identify if the remote server has HTTP load balancers. This plugin is a wrapper of Juan M. Bello Rivas' halberd.
hmap	Fingerprint the server type, i.e apache, iis, tomcat, etc.
http_vs_https_dist	Determines the network distance between the http and https ports for a target
php_eggs	Fingerprint the PHP version using documented easter eggs that exist in PHP.
server_header	Identify the server type based on the server header.
server_status	Find new URLs from the Apache server-status cgi.
werkzeug_debugger	Detect if Werkzeug's debugger is enabled.
basic_auth	Bruteforce HTTP basic authentication.
form_auth	Bruteforce HTML form authentication.
analyze_cookies	Grep every response for session cookies sent by the web application.
blank_body	Find responses with empty body.
cache_control	Grep every page for Pragma and Cache-Control headers.
click_jacking	Grep every page for X-Frame-Options header.
code_disclosure	Grep every page for code disclosure vulnerabilities.
content_sniffing	Check if all responses have X-Content-Type-Options header set
credit_cards	This plugin detects the occurrence of credit card numbers in web pages.

cross_domain_js	Find script tags with src attributes that point to a different domain.
csp	Identifies incorrect or too permissive Content Security Policy headers.
directory_indexing	Grep every response for directory indexing problems.
dom_xss	Grep every page for traces of DOM XSS.
dot_net_event_validation	Grep every page and identify the ones that have view state and don't have event validation.
error_500	Grep every page for error 500 pages that haven't been identified as bugs by other plugins
error_pages	Grep every page for error pages.
feeds	Grep every page and finds rss, atom, opml feeds.
file_upload	Find HTML forms with file upload capabilities.
form_autocomplete	Grep every page for detection of forms with 'autocomplete' capabilities containing password-type inputs.
get_emails	Find email accounts.
hash_analysis	Identify hashes in HTTP responses.
html_comments	Extract and analyze HTML comments.
http_auth_detect	Find responses that indicate that the resource requires auth.
http_in_body	Search for HTTP request/response string in response body.
lang	Read N pages and determines the language the site is written in.
meta_tags	Grep every page for interesting meta tags.
motw	Identify whether the page is compliant to mark of the web.
objects	Grep every page for objects and applets.
oracle	Find Oracle applications.
password_profiling	Create a list of possible passwords by reading HTTP response bodies.
path_disclosure	Grep every page for traces of path disclosure vulnerabilities.
private_ip	Find private IP addresses on the response body and headers.

ssn	This plugin detects the occurrence of US Social Security numbers in web pages.
strange_headers	Grep headers for uncommon headers sent in HTTP responses.
strange_http_codes	Analyze HTTP response codes sent by the remote web application.
strange_parameters	Grep the HTML response and find URIs that have strange parameters.
strange_reason	Analyze HTTP response reason (Not Found, Ok, Internal Server Error).
strict_transport_security	Check if HTTPS responses have the Strict-Transport-Security header set.
svn_users	Grep every response for users of the versioning system.
symfony	Grep every page for traces of the Symfony framework.
url_session	Finds URLs which have a parameter that holds the session ID.
user_defined_regex	Report a vulnerability if the response matches a user defined regex.
wsdl_greper	Grep every page for web service definition files.
xss_protection_header	Grep headers for "X-XSS-Protection: 0" which disables security features inthe browser.
web_spider	Crawl the web application.

Appendix 3: Cross-system detailed security test results

Method of testing

We used w3af and nmap scans to search for vulnerabilities in CIMI and other components.

By inspection we were able to see that many security functions have not been implemented.

For a list of security functions refer to Deliverable D2.4 Security/Privacy Requirements and Features, paragraph 3.1.

Analysis of results

There were three severe vulnerabilities found:

- The docker network was misconfigured exposing components to attack that had no protection
- There are no backups of systems software
- There is no protection against denial of service attacks

There was one PASS – data in flight is protected by https connections in all components examined.

Results

For a description of the column headings refer to Appendix 2.

Test description	Test result	Impact and probability	Severity	Category
Denial of Service – various attacks	There is no rate-limiting anywhere in the system	DoS successful – systems will be unusable. Very likely to eventually appear in an attack	Severe (not demonstrated)	Availability
Backups of systems software.	The CA servers have limited backups of data, configuration and software. Other systems software is usually generated on demand from images stored on docker-hub. Configuration and necessary data eg certificates, will be lost.	Disruption to services while CA servers are restored and rebuilt. Probability low. Minor disruption while other systems software is regenerated from images. Probability low.	Medium. Low.	Availability

Configuration of the docker network must be correctly set up. Scan the network topology using nmap and view the docker network configuration files directly.	The docker network for the standard cimi deployment via docker-compose.yml was found to be incorrect and exposed other components to the Internet that should not have been. In particular they expose Rest interfaces that have no authentication.	A scan will reveal the presence of exposed ports. Probability high. The ports can be connected to from a remote location without credentials and control commands injected without authorization. Probability high.	Severe	Confidentiality. Integrity Availability.
Validation of the configuration of the docker network.	There are no checks anywhere in the system for serious misconfigurations.	Misconfiguration can result in the exposure of severe vulnerabilities. Probability medium.	Medium severity.	Integrity
After a component has been compromised an outbound attack can be made e.g. to extend the compromised by attacking other components, or to attack other systems not involved with mf2c.	Not tested			
Scan the docker network for unexpected software running.	An smtp server was found listening in the cimi container. It was possible to connect to it via telnet and run simple commands. It was not possible to do any further attacks from this base e.g. smtp relay.	Unintended software listening on ports within a container can increase the attack surface. The impact can be very variable. In this case the impact was low. Probability low.	Low severity.	Integrity

Intercept data in flight by connecting to copper cable	Data in flight is mostly protected by https connections.	Ability to sniff passwords etc.	PASS	Confidentiality. Integrity.
Unauthorized access to data at rest by host server or compromised neighboring containers.	Data at rest is not encrypted and is not signed against tampering. To protect data at rest it is necessary for the server to also be secured in the normal way. The level of protection of the server is dealt with elsewhere. We were not able to demonstrate this vulnerability however.	Data at rest can include credentials leading to an elevation of privilege attack. It can contain private information leading to a regulatory breach. Modification of data at rest can include the configuration settings for components leading to various attacks. Impact severe. Probability low	Medium severity (not demonstrated)	Confidentiality. Integrity.
Use of ENV variables in docker	We are not testing for this vulnerability because temporarily it ***is*** actually in use for development purposes. Note that it would be necessary to have local access to the host server to be able to view the ENV content.	Passwords passed via ENV variables can be sniffed and used. Impact severe. Probability low.	Severe (not demonstrated)	Confidentiality
Only the authorized local unix user accounts are able to control docker. Inspect the /etc/groups file to confirm that only the expected unix accounts are in the docker group.	The expected unix accounts were in the docker group.	If docker can be controlled by unauthorized users then data within a container can be directly viewed or modified. Impact severe. Probability low.	Medium severity.	Confidentiality Integrity

Transfer of dataClay data to another dataClay instance in an insecure zone run by a malicious host server, leading to unauthorized access to data at rest.	Within dataClay the data at rest is not encrypted and is not signed against tampering. We were not able to demonstrate this vulnerability however.	Data at rest can include credentials leading to an elevation of privilege attack. It can contain private information leading to a regulatory breach. Modification of data at rest can include the configuration settings for components leading to various attacks. Impact severe. Probability low	Medium severity (not demonstrated)	Confidentiality. Integrity.

Appendix 4: Security architecture competence results

Method of analysis

By inspection we were able to see that many security functions have not been implemented.

For a list of security functions refer to Deliverable D2.4 Security/Privacy Requirements and Features, paragraph 3.1.

Analysis of results

There were four severe vulnerabilities found:

- There is no logging of security events and other important events
- There is no audit trail of security events e.g. new user account
- There is no alerting in real-time of security incidents in progress
- Physical tampering is possible of any hardware outside of a secure computer room

There was one PASS rating for a process to restrict who can make changes to systems software.

Discussion

Almost all of the necessary security functionality is missing. This makes it impossible to secure the system. For example, although we have SSL/TLS in use to secure data in flight, if the systems at either end are vulnerable the resulting security is no better than the weakest component.

Results

For a description of the column headings refer to Appendix 2.

Test description	Test result	Impact and probability	Severity	Category
Logging of security events and important other events	There is no logging at all in some components. Logs are not gathered in a central secure place e.g. a syslogger machine. Logs cannot be searched. Logs are not stored.	Attacks are unlikely to be detected if logs are not generated or are not stored centrally for analysis. This leads to attacks being successful. Probability high.	Severe	Integrity
Auditing of security events	There is no audit trail generated anywhere in the system of security events e.g. new user created. Audit trail is not gathered into a central location. Audit trail is not searchable. Legal admissibility of evidence from audit trail is zero.	Audit trails are essential for following the course of a breach of security or privacy. Audit trails are essential for criminal prosecution of hackers. Probability high.	Severe	Integrity

Alerting in real time of significant security events.	There is no alerting anywhere in the system of security events.	Alerts are essential for rapid response to a security situation. If sysadmins are unaware of a problem then a security breach is likely to succeed. Probability high	Severe	Integrity
Human process to deal with security breaches.	There are no policies or procedures to deal with security breaches or other problems.	Humans will not know how to react correctly to a security breach with a significant probability that it will be covered up because it is embarrassing. Also evidence required for a prosecution of a hacker is likely to be made inadmissible. Impact medium. Probability high.	Medium severity.	Integrity Availability
Process to restrict changes of systems software to authorized people only	There is a human process during the development phase of the project (through accounts with access to github and docker-hub) but there is no human process for the production phase and no security controls in place to enforce it. Once the software is deployed to the target server it is necessary for the server to be secured in the normal way. The development phase is adequately protected. The production phase does not have any protections at the moment, though this is acceptable at this stage of the project. The level of the protection of the server is dealt with elsewhere.	If there are no controls on changes to systems software a hacker could substitute a malicious program. Probability low.	PASS	Integrity

The inbound network access is controlled by a firewall	There are no dedicated firewall devices anywhere in the system. Docker has a simplistic firewall that does not forward any port that is not explicitly declared, which means containers are protected. But the host server will not be protected.	Network users can run scans for unprotected ports and access ports that should not have been exposed. Impact medium. Probability high.	Medium severity	Confidentiality Integrity
Patching of known software vulnerabilities	There is no patching of vulnerable programs at the moment.	Lack of patching increases the risk of a security breach. Impact very variable. Probability medium	Medium severity	Integrity Availability
Vulnerability to viruses	There is no antivirus service anywhere in the system.	Infection with a virus could have a variable impact. Probability low.	Low severity	Integrity
Impersonate the identity of a device. The PKI Certificate contains a FQDN but this is assigned in an informal way that means it is unreliable as Identity. The Beacon / CAU function assigns an ID that is loosely tied to a human email address.	The Certificate and Beacon device ID can be cloned and used for an impersonation attack if they are viewed so are dependent on the device being resistant to physical tampering.	Impersonating a device is not usually useful in an attack, but for a safety-critical system this may be important. Probability medium.	Medium severity	Confidentiality Integrity
Physical tampering with a device in an exposed location to get root or steal credentials	Any hardware that is not in a secure, locked computer-room is vulnerable to physical tampering.	Credentials can be stolen, configurations altered and software tampered with. Impact very severe. Probability high	Severe	Confidentiality Integrity Availability

Appendix 5: Business function security test results

Method of testing

By inspection we were able to see that many security functions have not been implemented.

Analysis of results

There were severe vulnerabilities in all tests we examined. There were no PASS results.

Discussion

These results mean that it is not possible to operate a business for a long period of time with this software due to the severity of the problems that are likely to eventually occur. There would be regulatory non-compliance (GDPR) and damage to business reputation.

Results

For a description of the column headings refer to Appendix 2.

Test description	Test result	Impact and probability	Severity	Category
Backups of data.	There are no backups of data	Data loss certain when storage fails. Low probability in most systems	Severe	Availability
In the event of hardware failure or major systems problems, business continuity is possible	There are no backups of data or systems so data would be lost in the event of hardware failure etc. Business continuity not possible.	Severe impact on the business from lost data and non-availability of hardware. Probability medium.	Severe	Availability
The privacy of humans is protected at all times (for GDPR compliance)	Security vulnerabilities listed in Appendix D mean that it is not possible to protect their privacy.	Non-compliance with GDPR and possible fines. Probability almost certain.	Severe	Confidentiality
Business reputation is protected by integrity of the systems.	Security vulnerabilities listed above mean that it is not possible to protect business reputation.	Lost business and revenue. Probability medium.	Severe	Integrity

Appendix 6: Threat model reference

We use the STRIDE threat model[3,4,5].

For a description of STRIDE's relevance to mF2C refer to Deliverable D2.4 Security/Privacy Requirements and Features, paragraph 2.1.1.

To use Threat Modelling, the threats are categorized in a way that is easily understood (i.e. STRIDE). They are connected in a simple way and this model is then used to drive the security remediation work. The strength of this model is that it is seen from an attacker's viewpoint.

Following is a description of the Stride topics:

Stride topic	Examples
Spoofing	Impersonating another's identity Stealing another's credentials Making a connection that is fake to a service
Tampering	Altering data, viewing data Stealing credentials Opening a physical piece of hardware to access the storage device
Repudiation	Denying having performed an action, but there is no way the system can prove it was done, usually due to lack of an audit trail
Information disclosure	Revealing information to someone who is not supposed to access to it
Denial of service	Block access to a service for everyone by overwhelming it with fake requests
Elevation of privilege	Getting access to something that has greater control over the system

Because of the character of the mF2C system, which involves the location of IoT devices and Fog servers in physically exposed locations, we give more than normal attention to the topic of physical tampering.

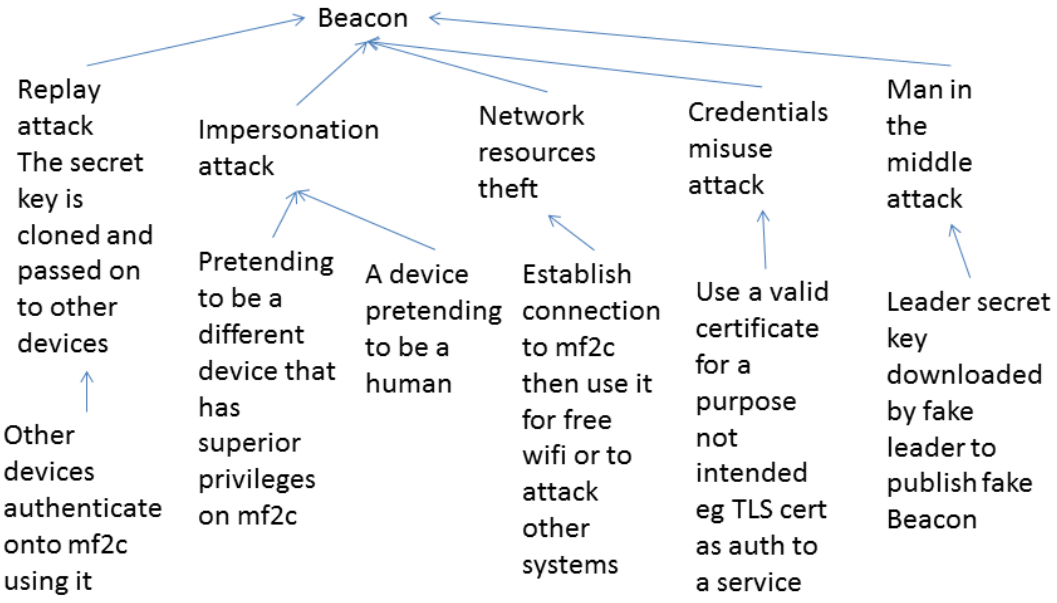
We shall not produce a complete model, only a representative model, because of the amount of work involved. Below is shown a very small part of the threat model so far produced.

[3] https://en.wikipedia.org/wiki/STRIDE_(security)

[4] https://www.owasp.org/index.php/Threat_Risk_Modeling

[5] https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx

Simplified threat model for Beacon and CAU



Appendix 7: Security architecture reference

A computer security model is a scheme for specifying and enforcing security policies. The security policies are then enforced through security controls.

In this system, we are using an Information Flow security model in which data is held in discrete logical compartments. It is compartmentalized based on classification and need to know. These are provided by the ACLs.

The logical compartmentalization is provided by

- multiple intermediate Certification Authorities (CA) that have no trust of each other
- network compartmentalization
- encrypted containers that use the keypairs provided by the CAs.

Logical compartment	Partitioning by...
Layered software	Cloud versus Fog versus Use Case application versus Device
CA	Separate CAs for Fog servers versus IoT devices.
Keypairs	• Separate keys for each device or server • Frequent renewal of keys • ACLS
Network	Subnet and gateway/firewall

The purpose of compartmentalizing data is to limit damage due to compromise. It also fits in with the character of Internet of Things processing which is primarily a dataflow.

Security policies are enforced through security mechanisms. In this system, we use a small set of security mechanisms (except for physical security which requires multiple layers of security and human supervision).

Type of security mechanism	Example
Physical security	Secure computer-room, encrypted data
Public Key Infrastructure	Encryption, Identity, revocation of access
ACLs	Grant or deny access in a fine-grained way
Gateways	Acts as firewall, network gateway and Policy Enforcement Point.

Appendix 8: Privacy tests detailed results

Method of testing

We were not able to test the privacy of the system due to lack of time.

Analysis of results

Elsewhere in this document, we note that because the security of the system is very weak it is not possible to provide privacy. However, no tests have been performed to verify this with facts, so it has been rated as Not Tested.

It seems that the components do not handle any Personally Identifiable Information (PII) so would be out of scope of the GDPR. However physical location is considered PII and there is a possibility this may be held in the system. Further checks are necessary.

Devices are assumed to be Internet of Things devices and smartphones. They are very likely to hold PII. We shall not consider them here but will do so with the Use Case applications in Deliverable D5.3.

Results

For a description of the column headings refer to Appendix 2.

Test description	Test result	Impact and probability	Severity	Category
Do components handle any data that can be Personally Identifiable Information (PII)? Check that it is not disclosed to unauthorized humans or systems.	Not tested			
Do devices handle any PII? Check that it is not disclosed to unauthorized humans or systems.	Not tested			