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:	*07/06/2018*
Version:	*2.0*
Status	*Final (updated)*
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), Jasenka Dizdarevic (TUBS), Francisco Carpio (TUBS), Eva Tordera (UPC), Xavi Masip (UPC), Sergi Sànchez (UPC), Jordi Garcia (UPC), Ester Simó (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 H2020 Programme		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

This 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)
1.5	07/06/2018	Document update	All
1.6	08/06/2018	Internal review	Jens Jensen (STFC)
1.7	12/06/2018	Internal review	Admela Jukan (TUBS)
1.8	13/06/2018	Final document harmonization and comments addressed	Cristovao Cordeiro (SIXSQ), All
1.9	18/06/2018	Improve executive summary	Jens Jensen (STFC)
2.0	19/06/2018	Fix captions for tables and images. Style table of contents	Cristovao Cordeiro (SIXSQ)

Table of Contents

List of figures

List of tables

Executive Summary

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

The mF2C platform implements the architecture and design from earlier deliverables, but some choices had to be made. The project needed to make a choice of features, to ensure that they can be implemented well enough in time for IT-1, in order for them to be tested and validated. Yet the project also needed a wide enough selection of features to ensure that the concept of fog-to-cloud can be proven, and the entire mF2C platform remains sufficiently usable. Thus, some features were postponed to IT-2, and other features that had been foreseen in earlier work were considered obsolete, which generally means they have been replaced with existing products that offer the same functionality, or better. The main examples of the latter are dataClay which has obsoleted the "Data Manager" component, and COMPSs which has obsoleted the "Services Runtime." Not only has this approach saved developer time and effort, but also, the overall functionality was improved because the reused components come with a full set of features.

Source code and documentation are generally publicly available, components are loosely coupled using web services, and components are individually "packaged" and deployed in containers which can, in turn, be "composed" to form more complex "super-components" which can be tested as if they were a single component.

The ultimate validation of the platform is through the use cases, and each of these describes how it makes use of the mF2C IT-1 release. Our chosen approach for development had the use cases being developed and integrated at the same time as the platform. The use cases thus sometimes simplify their implementation, e.g. by substituting a simpler component from the full one, or bypassing steps they would ordinarily take. However, the advantage of this approach was that feedback and experiences from the implementation of use cases could be taken on board in the development of the platform.

In order to ensure that the entire platform is validated, we ensure that the component tests mentioned above cover the platform comprehensively, and we also introduce a supplementary use case which aims to demonstrate the initial discovery, registration, deployment of credentials, etc.; features which may not be readily apparent in the use cases demonstrators.

Another important feature of the validation is the introduction of test beds, wherein services can be deployed and tested. This approach makes it possible to demonstrate a full-scale scenario in a simulated environment – a simulated city, or harbor – as well as introduce faults – e.g. a fog leader disappearing, a signal being jammed, etc. More intrusive security tests, which would not be permitted in a public cloud infrastructure, can also be run.

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 sample workflows update.
- Section 4 introduces the first mF2C prototype (IT-1) 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
ACL	Access Control List
API	Application Programming Interface
BT	BlueTooth
BW	BandWidth
CA	Certification Authority
CAU	Control Area Unit
CIMI	Cloud Infrastructure Management Interface
CLI	Command Line Interface
CSR	Certificate Signing Request
FQDN	Fully Qualified Domain Name
GPS	Global Positioning System
ICMP	Internet Control Message Protocol
IoT	Internet of Things
IR	Infra Red
IT	Iteration
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LED	Light Emitting Diode
MQTT	Message Queuing Telemetry Transport
OS	Operating System
PoC	Proof of Concept

PoI	Point of Interest
QoS	Quality of Service
REST	REpresentational State Transfer
SLA	Service Level Agreement
UC	Use Case
YAML	Yet Another Markup Language

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

Table 2 and Table 3 below include new functionalities for the Platform Manager and the Agent Controller respectively, 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, and finally in cases where a component has been considered as obsolete, a 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. Postponed to IT-2
	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. Prototyped but not included in IT-1; to be included in IT-2
Service Orchestration	SLA Management	The automatic creation of an SLA agreement (based on templates) has been postponed to IT-2

Table 2. Platform Manager

Module	***Component***	***Description***
Resource Manager	Data Manager	Obsolete. 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.
	Monitoring	Obsolete. It has been merged with the Telemetry Monitoring module and the Landscaper component, which together provide a global monitoring overview of the infrastructure. Available in IT-1

Service Manager	Services Runtime	Obsolete. This has been replaced by the COMPSs engine in IT-1
	Allocation	Allocates available resources to requests to meet security and privacy rules, cost models, guaranteeing overall optimal resources usage. (postponed to IT-2)
	Mapping	Matches the needs of the task with the resources of the specific device, and additionally sends the task (if it is the first time this task is requested in that device) to the service categorization module to be categorized so that later it can be used for a better matching between the tasks and the device's resources.
User Management	Assessment	Component verifying that the mF2C applications "respect" the sharing model and the profile properties defined by the device's user. (postponed to IT-2)

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.

On the Platform Manager side, 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 mainly be based on 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 in building the Recommender’s output, referred to as "recipe".

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.

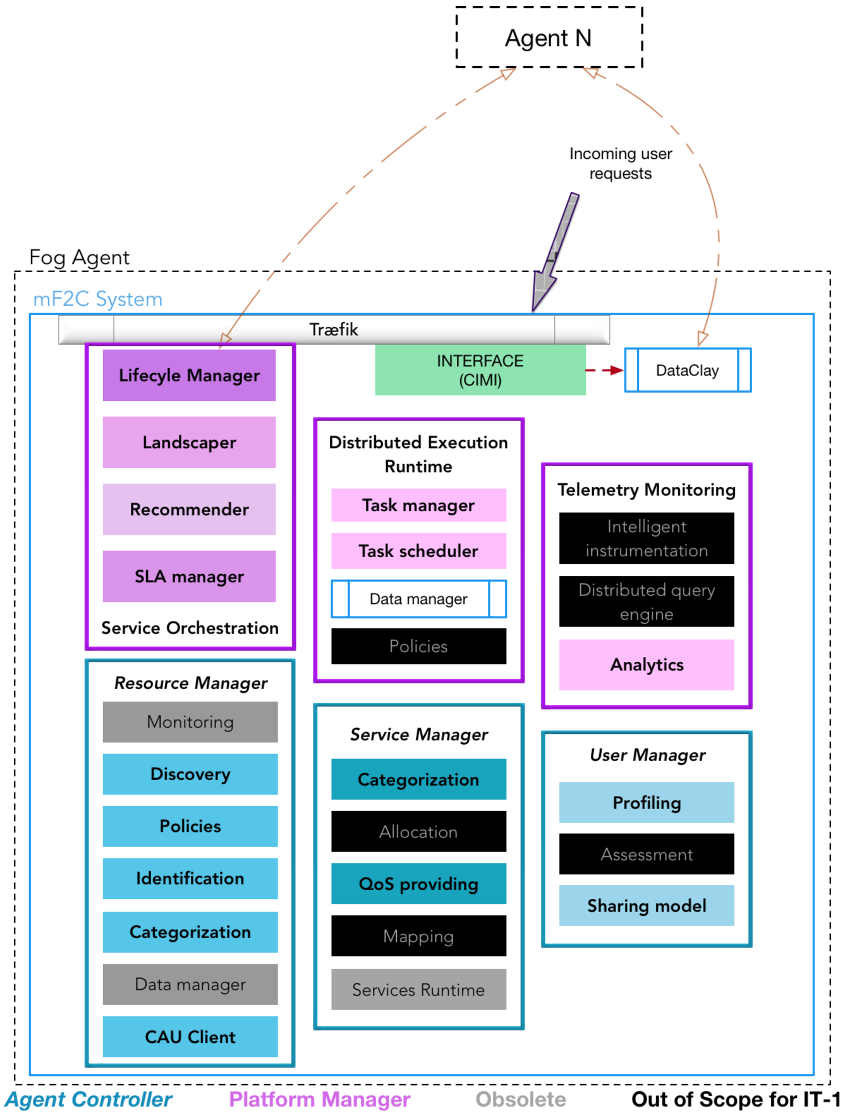


Figure 1 Components layout for IT-1

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;

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

- **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;
- **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 and Telemetry Monitoring	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		
mF2C Middleware				
Security (CA)		Certificate Authority	51433 52433 53433 51022 52022 53022	
CAU		Control Area Unit	46400 46410	

Table 4. Ports and naming conventions

3. Workflows update

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.

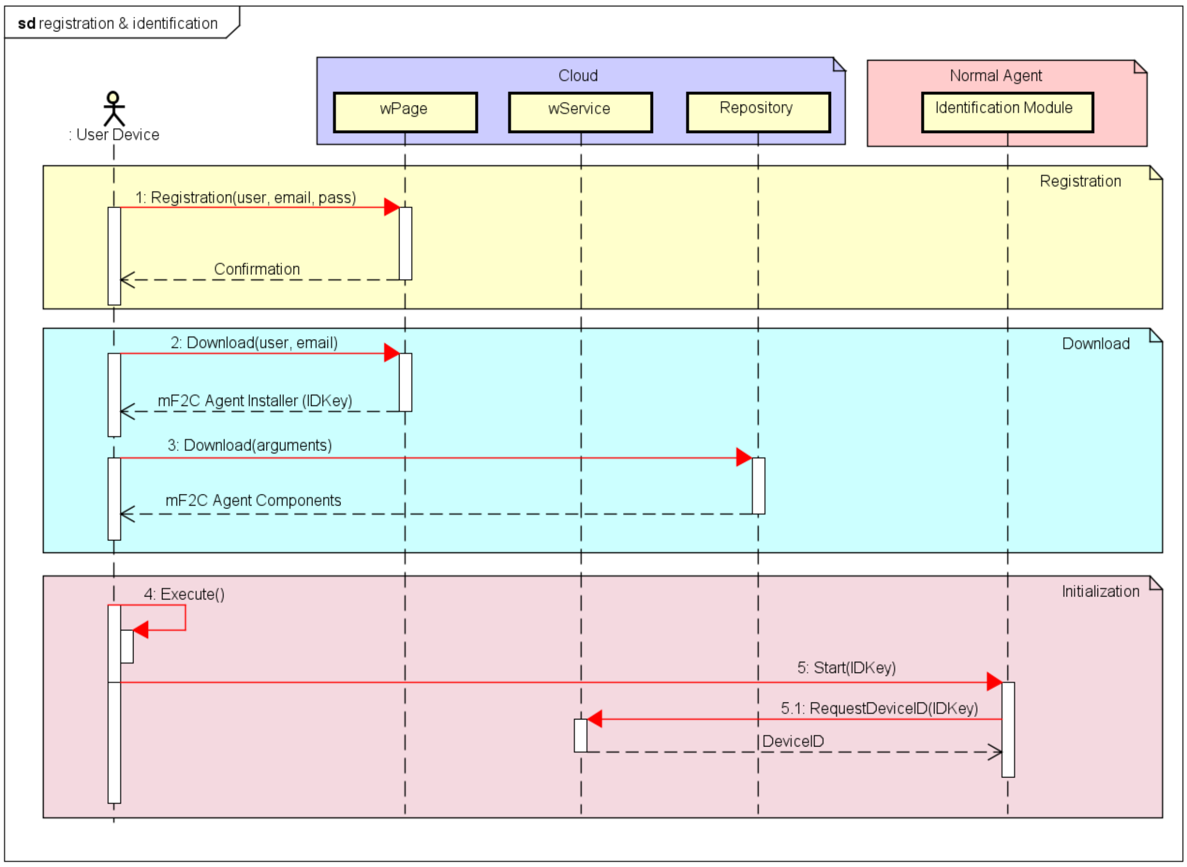
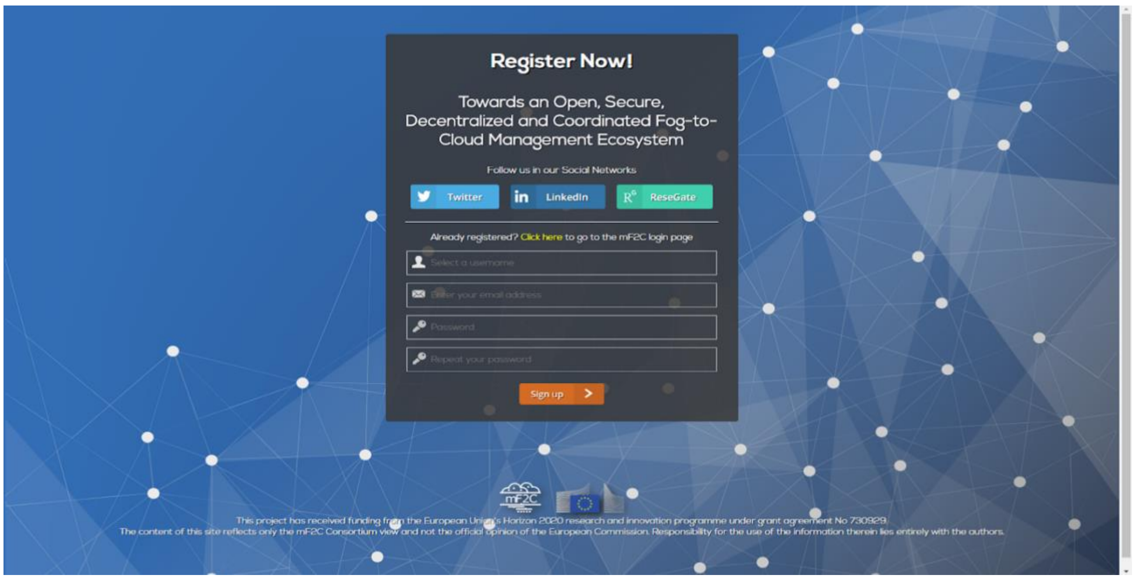


Figure 2 Registration and Identification Workflow

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.



Register Now!

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

Follow us in our Social Networks

Twitter LinkedIn ReseGate

Already registered? Click here to go to the mF2C login page

Select a username

Enter your email address

Password

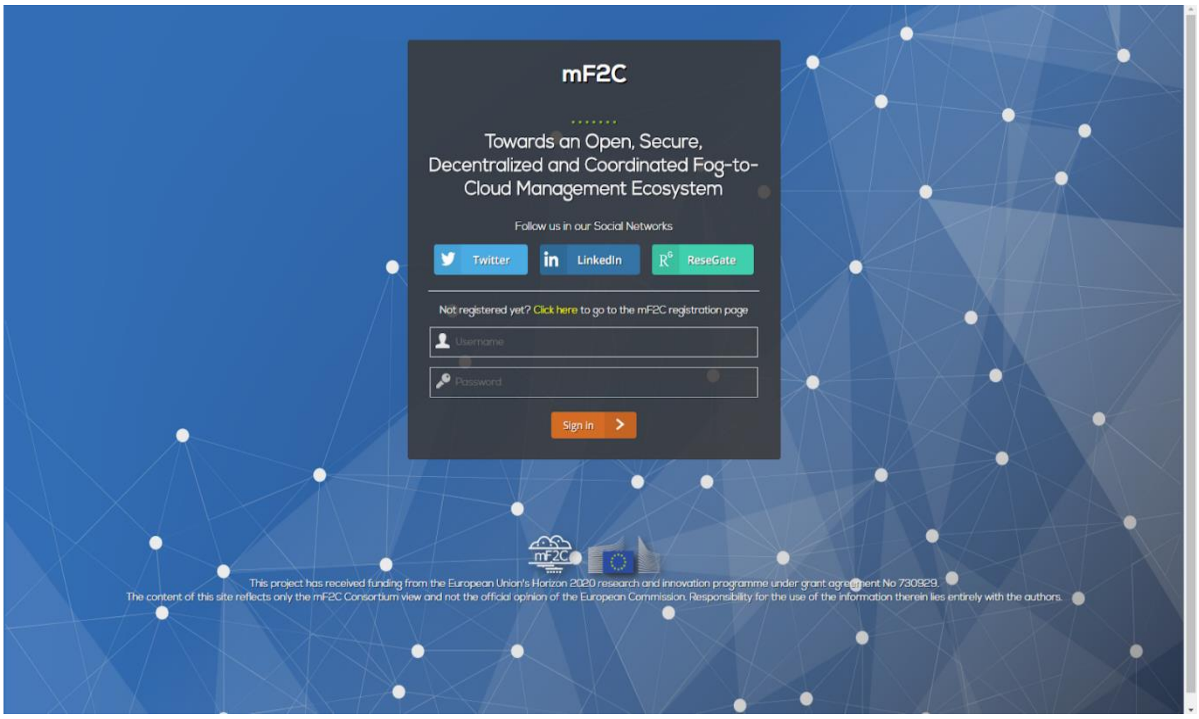
Repeat your password

Sign up

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730929.
The content of this site reflects only the mF2C Consortium view and not the official opinion of the European Commission. Responsibility for the use of the information therein lies entirely with the authors.

Figure 3 Registration page screenshot

Once the user is registered, the next time he/she accesses 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 is shown in Figure 4.



mF2C

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

Follow us in our Social Networks

Twitter LinkedIn ReseGate

Not registered yet? Click here to go to the mF2C registration page

Username

Password

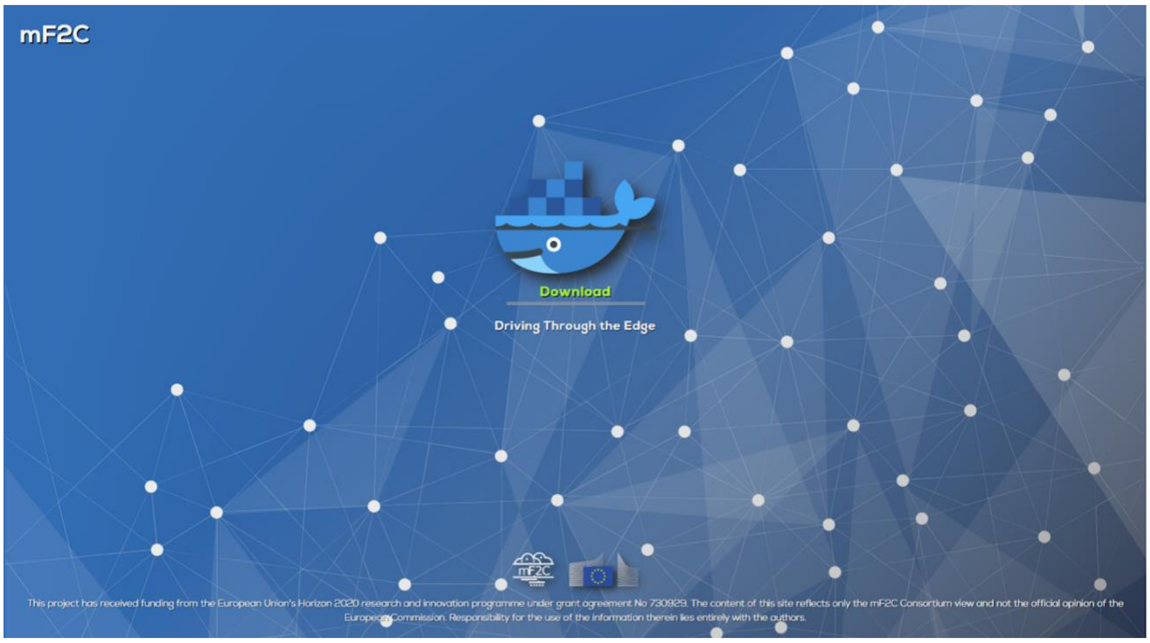
Sign in

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730929.
The content of this site reflects only the mF2C Consortium view and not the official opinion of the European Commission. Responsibility for the use of the information therein lies entirely with the authors.

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:

$$IDKey = hash512(useremailaddress)$$



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730929. The content of this site reflects only the mF2C Consortium view and not the official opinion of the European Commission. Responsibility for the use of the information therein lies entirely with the authors.

Figure 5 Download page screenshot

3.1.1 Integration tests

Test Case Number Version	**TC-Registration and Identification**
Test Case Title	Registration and Identification workflow
Modules tested	Registration process Identification module
Related requirement(s)	**Registration:** 1. None **Identification (first time):** 1. Network connectivity **Identification:** 1. The device already has a valid ID.
Target	To acquire a unique ID for the m2FC agent.
Initial condition(s)	None
Expected results	1. The user data collected during the registration process (email, username and password) stored in the database. 2. The IDKey and device ID are obtained and available to other system components.
Status	Completed
Owner	Universitat Politècnica de Catalunya (UPC)
Assigned	Alejandro Gómez Cárdenas (UPC)
Steps	**Registration:**

	1. Fill the web form and submit the information. 2. Validate the account using the link provided via email. **Identification (first time):** 1. Start the mF2C agent 2. Request an ID from the cloud web service 3. Store the acquired ID internally **Identification:** 1. Start the mF2C agent 2. Provide IDKey and device ID to requester components
Feedback receiver	Alejandro Gómez Cárdenas (UPC)
Start date	M10
End date	M16
Passed?	Passed
Bug ID	-
Problems	
Required changes	-
Comments	-

Table 5. Integration tests for the registration and identification workflows

3.2 Discovery, Authentication (Security) and Categorization

The workflow in 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 (Control Area Unit) and the CA (Certification Authority) is performed. The CAU acts as a gateway that securely connects the device with the credentials-issuing service in the cloud, because we cannot give access directly to the Internet/Cloud to an unauthenticated agent. 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 right after the successful authentication of a new device, the categorization module is executed, producing the information about the device's available resources, to be stored in the agent's local database, and later synchronized with the leader's database by dataClay. Some comments have been added to illustrate the deployment of the whole process: discovery, authentication and categorization.

In Figure 7 we show one of the steps of this 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. It is worth highlighting that in the current deployment (IT-1), no attempt is considered to handle a potential problem in this process, though it should be added in next iteration.

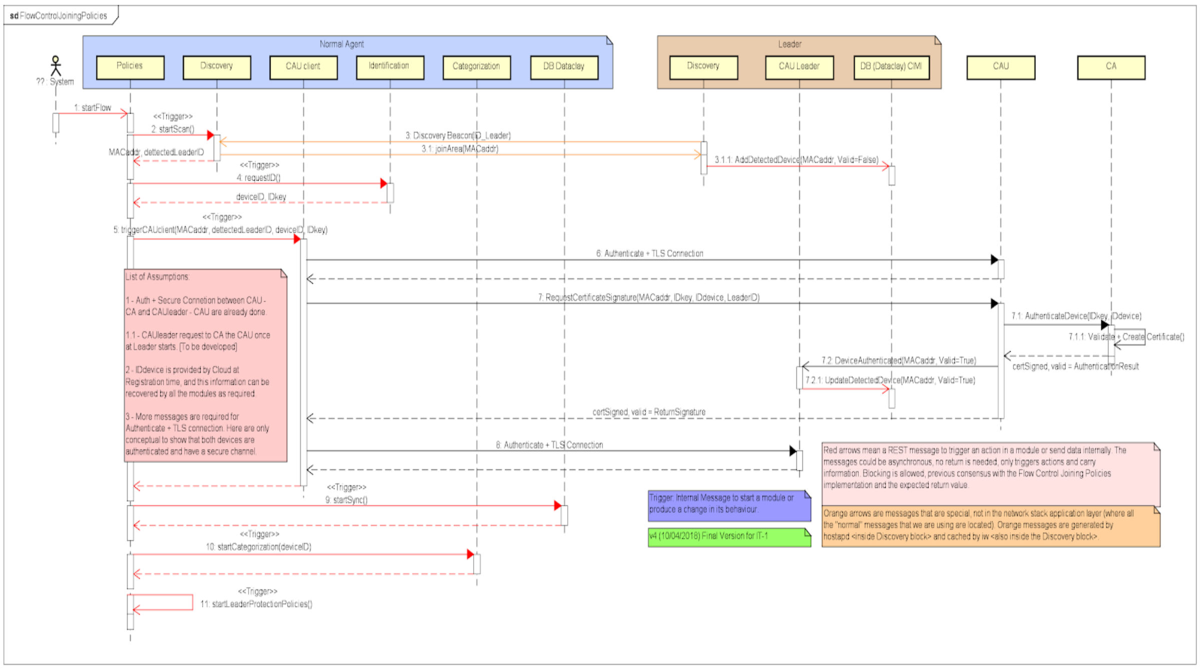
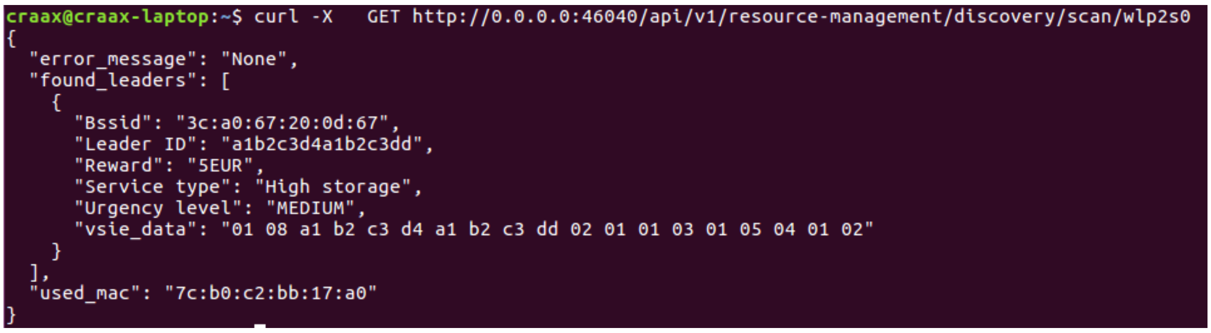

sd FlowControlJoiningPolicies
Normal Agent
Policies
Discovery
CAU client
Identification
Categorization
DB Dataclay
Leader
Discovery
CAU Leader
DB (Dataclay) CIMI
CAU
CA
?? : System
1: startFlow
<<Trigger>>
2: startScan()
3: Discovery Beacon(ID_Leader)
3.1: joinArea(MACaddr)
3.1.1: AddDetectedDevice(MACaddr, Valid=False)
MACaddr, dettectedLeaderID
<<Trigger>>
4: requestID()
deviceID, IDkey
<<Trigger>>
5: triggerCAUclient(MACaddr, dettectedLeaderID, deviceID, IDkey)
6: Authenticate + TLS Connection
7: RequestCertificateSignature(MACaddr, IDkey, IDdevice, LeaderID)
7.1: AuthenticateDevice(IDkey, IDdevice)
7.1.1: Validate + Create Certificate()
certSigned, valid = AuthenticationResult
7.2: DeviceAuthenticated(MACaddr, Valid=True)
7.2.1: UpdateDetectedDevice(MACaddr, Valid=True)
certSigned, valid = ReturnSignature
8: Authenticate + TLS Connection
<<Trigger>>
9: startSync()
<<Trigger>>
10: startCategorization(deviceID)
<<Trigger>>
11: startLeaderProtectionPolicies()
List of Assumptions:
1 - Auth + Secure Connetion between CAU - CA and CAUleader - CAU are already done.
1.1 - CAUleader request to CA the CAU once at Leader starts. [To be developed]
2 - IDdevice is provided by Cloud at Registration time, and this information can be recovered by all the modules as required.
3 - More messages are required for Authenticate + TLS connection. Here are only conceptual to show that both devices are authenticated and have a secure channel.
Trigger: Internal Message to start a module or produce a change in its behaviour.
v4 (10/04/2018) Final Version for IT-1
Red arrows mean a REST message to trigger an action in a module or send data internally. The messages could be asynchronous, no return is needed, only triggers actions and carry information. Blocking is allowed, previous consensus with the Flow Control Joining Policies implementation and the expected return value.
Orange arrows are messages that are special, not in the network stack application layer (where all the "normal" messages that we are using are located). Orange messages are generated by hostapd <inside Discovery block> and cached by iw <also inside the Discovery block>.

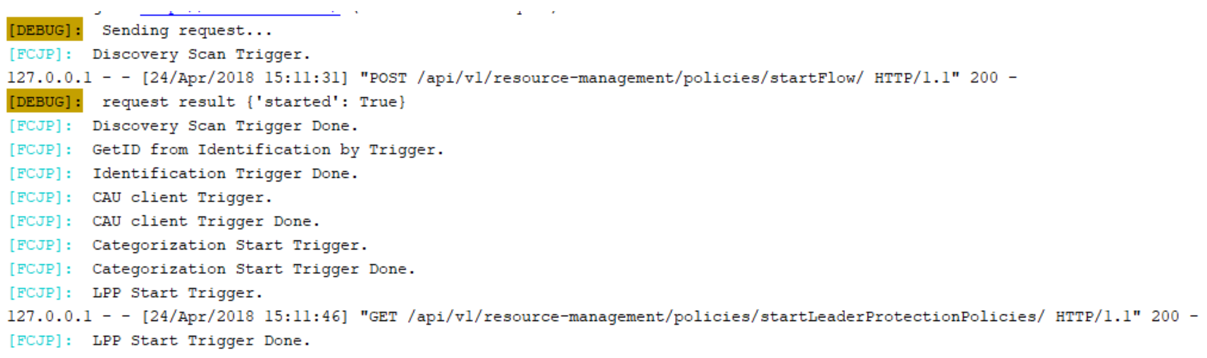
Figure 6 Discovery, authentication and categorization workflow (zoomed in figure in Annex 9)



```
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

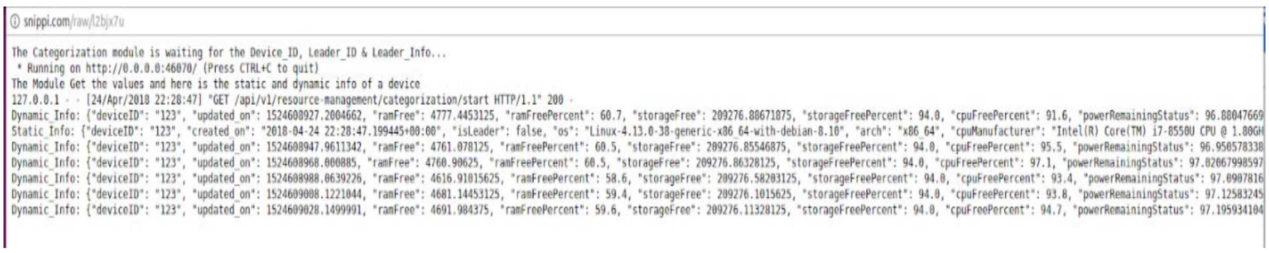
Figure 8 shows the discovery, authentication and categorization flow (from Figure 6) being executed.



```
[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 in Figure 6.



```
snippi.com/raw/l2bjx7u

The Categorization module is waiting for the Device_ID, Leader_ID & Leader_Info...
 * Running on http://0.0.0.0:46070/ (Press CTRL+C to quit)
The Module Get the values and here is the static and dynamic info of a device
127.0.0.1 - - [24/Apr/2018 22:28:47] "GET /api/v1/resource-management/categorization/start HTTP/1.1" 200 -
Dynamic_Info: {"deviceID": "123", "updated_on": 1524608927.2004662, "ramFree": 4777.4453125, "ramFreePercent": 60.7, "storageFree": 209276.88671875, "storageFreePercent": 94.0, "cpuFreePercent": 91.6, "powerRemainingStatus": 96.88047669
Static_Info: {"deviceID": "123", "created_on": "2018-04-24 22:28:47.199445+00:00", "isLeader": false, "os": "Linux-4.13.0-38-generic-x86_64-with-debian-8.10", "arch": "x86_64", "cpuManufacturer": "Intel(R) Core(TM) i7-8550U CPU @ 1.80GH
Dynamic_Info: {"deviceID": "123", "updated_on": 1524608947.9611342, "ramFree": 4761.078125, "ramFreePercent": 60.5, "storageFree": 209276.85546875, "storageFreePercent": 94.0, "cpuFreePercent": 95.5, "powerRemainingStatus": 96.950578338
Dynamic_Info: {"deviceID": "123", "updated_on": 1524608968.000885, "ramFree": 4760.90625, "ramFreePercent": 60.5, "storageFree": 209276.86328125, "storageFreePercent": 94.0, "cpuFreePercent": 97.1, "powerRemainingStatus": 97.82067998597
Dynamic_Info: {"deviceID": "123", "updated_on": 1524608988.0639226, "ramFree": 4616.91015625, "ramFreePercent": 58.6, "storageFree": 209276.58203125, "storageFreePercent": 94.0, "cpuFreePercent": 93.4, "powerRemainingStatus": 97.0907816
Dynamic_Info: {"deviceID": "123", "updated_on": 1524609008.1221044, "ramFree": 4681.14453125, "ramFreePercent": 59.4, "storageFree": 209276.1015625, "storageFreePercent": 94.0, "cpuFreePercent": 93.8, "powerRemainingStatus": 97.12583245
Dynamic_Info: {"deviceID": "123", "updated_on": 1524609028.1499991, "ramFree": 4691.984375, "ramFreePercent": 59.6, "storageFree": 209276.11328125, "storageFreePercent": 94.0, "cpuFreePercent": 94.7, "powerRemainingStatus": 97.195934104
```

Figure 9 Screenshot of the categorization module execution

3.2.1 Integration tests

Test Case Number Version	**TC-Discovery, authentication and categorization**
Test Case Title	Validation of the Discovery, authentication and categorization workflow.
Modules tested	Discovery, Identification, Categorization, Policies, CAU, CA, CAUclient, CAUleader, CIMI.
Related requirement(s)	The system must have docker and docker-compose clients installed and running. For the discovery functionality, the required OS is a Linux with the iw and git CLI utilities installed. The registration of the agent is already done. An active area is already started and a Leader is broadcasting beacons.
Target	In this scenario, a not-yet-joined agent approaches an area, detects the leader and joins in a secure way. The discovery module is responsible for scanning the beacons from the leader. The CAUclient module performs the authentication and establishes the secure connection between the agent and the leader. The categorization is responsible for scanning the resources in the agent and creating the information that is sent to the Leader. If needed, the policies module starts the leader failure mitigation procedure to ensure availability of an eligible backup on a suitable device. After all these steps, the agent is able to receive and execute service requests and executions.

Initial condition(s)	1. Discovery, Identification, Categorization, Policies, CAUclient and CIMI modules must be deployed from scratch, through a single compose file. 2. CIMI's interface must be published into the tester's host machine, so it can be reached at *localhost.* 3. The WiFi interface must be linked to the container that executes the Discovery Module.
Expected results	The user with the agent installed approaches an area and detects the Leader beacons, joins the area in a secure way, categorizes the device and starts the leader failure mitigation protocol as an eligible backup if it is capable of being a backup.
Status	**Without Security**: Completed. **With Security**: Completed.
Owner	Universitat Politècnica de Catalunya (UPC)
Assigned	Eva Marín (UPC)
Steps	1. Execute the agent. 2. Check if all the modules are started.
Feedback receiver	Eva Marín (UPC)
Start date	M10
End date	M16
Passed?	Passed with and without security integrated. With security, the procedure is yet to be improved and repeated.
Bug ID	-
Problems	
Required changes	-
Comments	-

Table 6. Integration tests for the discovery, authentication and categorization workflow

3.3 Leader failure

This workflow describes 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 exists 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:

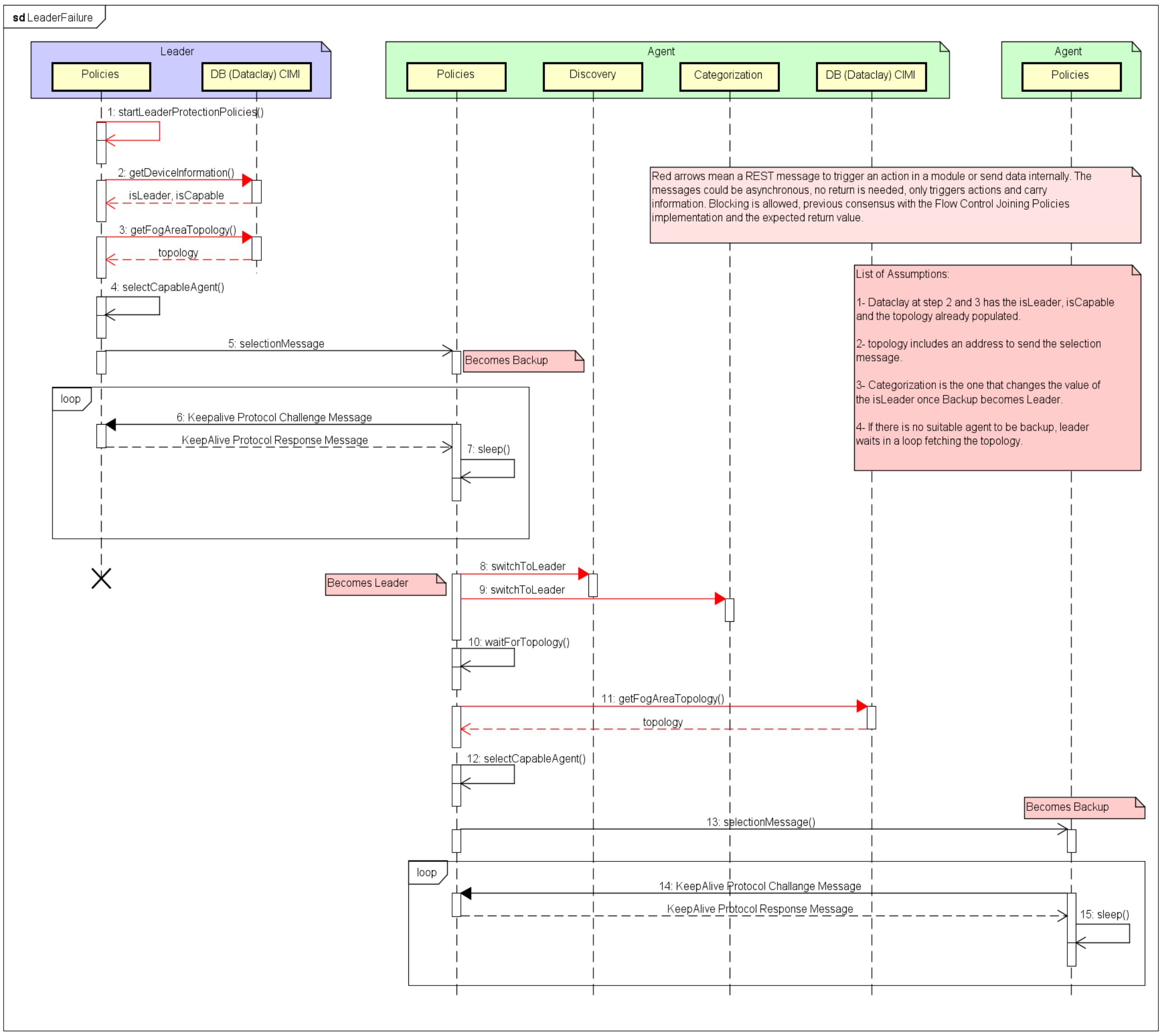
1. Accept that a leader can fail - with severe loss of fog functionality, but this may be acceptable in some cases, e.g. a temporary loss of fog connectivity.
2. Designate a backup leader. Through a process of data transfer (from the failed leader’s database, if possible) and/or rebuilding its database (re-replicated from the agents’ local database), the backup takes over.

3. Implement a leader selection process whereby any agent 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 (FogAreaTopology 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 top border in Figure 10). 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* message from the backup. Once the selection 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 performs 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]. Finally, Figure 13 shows 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.



Red arrows mean a REST message to trigger an action in a module or send data internally. The messages could be asynchronous, no return is needed, only triggers actions and carry information. Blocking is allowed, previous consensus with the Flow Control Joining Policies implementation and the expected return value.

List of Assumptions:

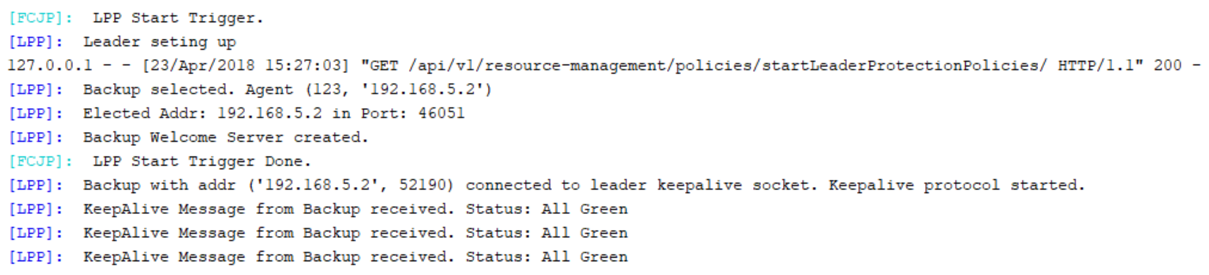
1- Dataclay at step 2 and 3 has the isLeader, isCapable and the topology already populated.

2- topology includes an address to send the selection message.

3- Categorization is the one that changes the value of the isLeader once Backup becomes Leader.

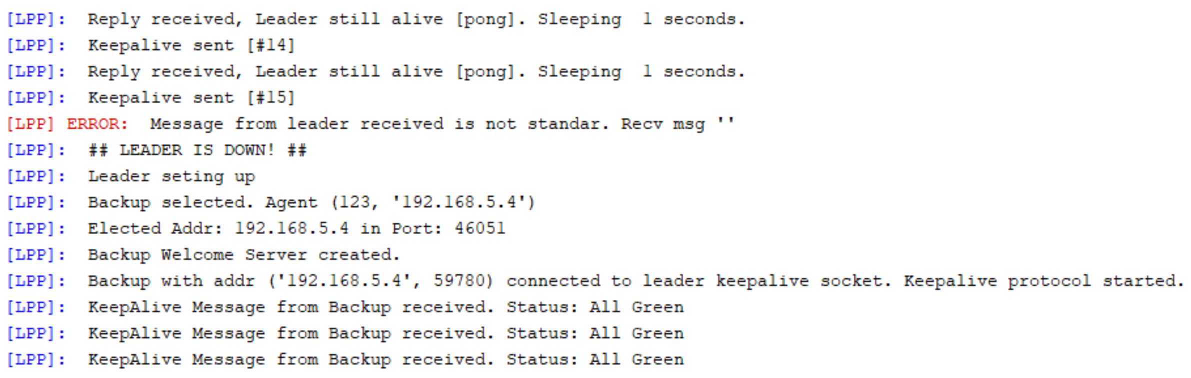
4- If there is no suitable agent to be backup, leader waits in a loop fetching the topology.

Figure 10 Leader failure workflow



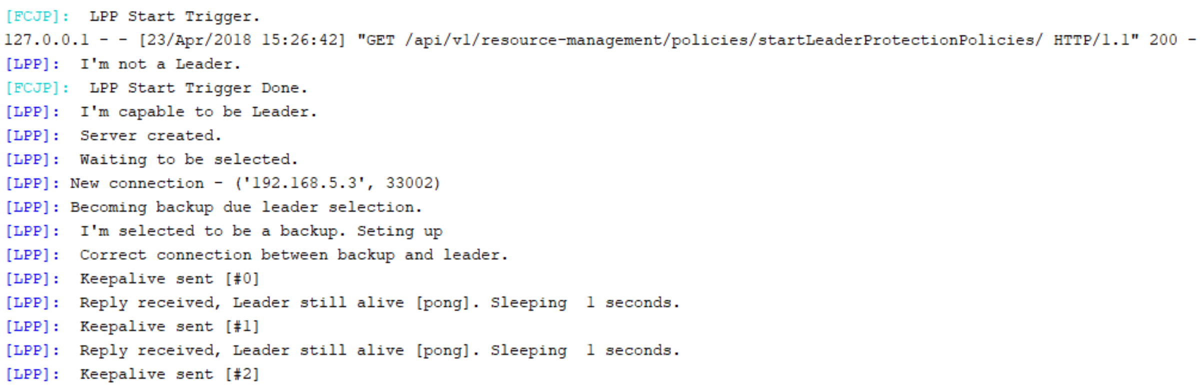
```
[FCJP]:  LPP Start Trigger.
[LPP]:  Leader seting up
127.0.0.1 - - [23/Apr/2018 15:27:03] "GET /api/v1/resource-management/policies/startLeaderProtectionPolicies/ HTTP/1.1" 200 -
[LPP]:  Backup selected. Agent (123, '192.168.5.2')
[LPP]:  Elected Addr: 192.168.5.2 in Port: 46051
[LPP]:  Backup Welcome 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

3.3.1 Integration tests

Test Case Number Version	**TC-Leader Failure**
Test Case Title	Validation of the Leader Failure workflow
Modules tested	Discovery, Identification, Categorization, Policies, CIMI
Related requirement(s)	The system must have docker and docker-compose installed and running. For the discovery functionality, the required OS is a Linux with iw and git. The registration of the agent is already done. An active area is already started and a Leader is broadcasting beacons. A backup is already selected and performing the keepalive protocol.
Target	In this scenario, a leader is running and a backup, which performs the leader failure detection protocol, has been selected. When the leader interrupts the normal activity, the backup switches role and becomes a new leader. Then, a new

	backup is selected from the eligible set, otherwise no backup is selected. The policies block implements all the leader/backup functionalities. The discovery and categorization modules must change their internal functionality and outcome when they are required to change from backup role to leader role. After the procedure, the area doesn't lose control and a new leader coordinates the agents.
Initial condition(s)	4. The Discovery, Identification, Categorization, Policies and CIMI modules must be deployed from scratch, through a single compose file. 5. CIMI's interface must be published into the tester's host machine, so it can be reached at *localhost.* 6. The WiFi interface must be linked to the container that executes the Discovery Module. 7. The Agent is part of one area, is already joined with security. 8. The area topology is set and reachable from the Policies block.
Expected results	The user with the agent starts the protocol as a normal agent that can be eligible to be a backup agent of the leader, if it is capable and the leader selects it. If the agent is already a leader or it becomes a leader due a failure of the leader, it performs a selection of a backup based on policies. If the Leader fails, the backup is able to detect it and switches role to become the new Leader.
Status	Completed.
Owner	Universitat Politècnica de Catalunya (UPC)
Assigned	Eva Marín(UPC)
Steps	1. Provoke a failure in the leader (network disruption, abort the execution, etc.) 2. Check the new leader and backup if selected.
Feedback receiver	Eva Marín(UPC)
Start date	M10
End date	M16
Passed?	Passed.

Bug ID	-
Problems	
Required changes	-
Comments	-

Table 7. Integration tests for the leader failure workflow

3.4 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. This component will take part in the initialization and registration processes for the next iteration. In IT-1 it will work as a "standalone" module that will be used by the Lifecycle during the service initialization workflow.

The following workflow (Figure 14) presents some minor changes regarding the workflow presented in D3.3 [3]. In this workflow, the Profiling module interacts with CIMI instead of calling the database directly. 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).

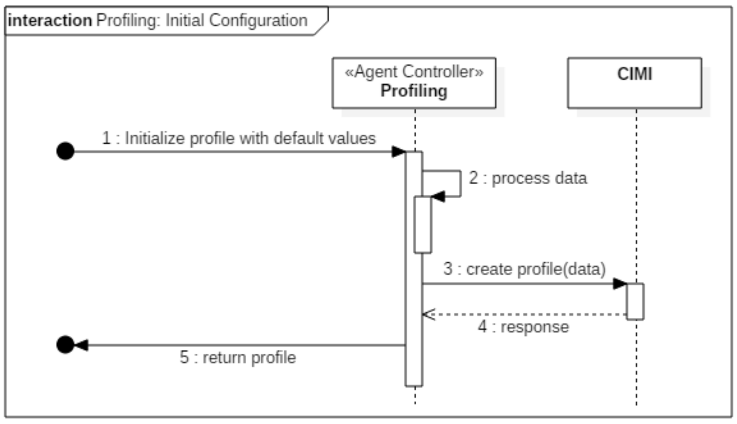

interaction Profiling: Initial Configuration
«Agent Controller» Profiling
CIMI
1 : Initialize profile with default values
2 : process data
3 : create profile(data)
4 : response
5 : return profile

Figure 14 Profile properties configuration (initialization and update workflow)

3.4.1 Integration tests

Test Case Number Version	**TC-Profiling - Initial configuration evaluation**
Test Case Title	MF2C (IT-1): Evaluate the initial configuration of the user profile
Module tested	User Management module, CIMI
Related requirement(s)	One agent (with CIMI and User Management modules) deployed and running on the testbed
Target	This scenario validates the creation of a user profile in an agent.

	The User Management module is responsible for processing the profile creation request, and CIMI is called to store the profiling properties.
Initial condition(s)	- The user has been introduced in the system
Expected results	- The user profile is created
Status	Completed
Owner	ATOS
Assigned	Roi Sucasas (ATOS)
Steps	1. Send a POST request to the User Management REST API with the required parameters: profile properties 2. The User Management module creates the profile by calling CIMI, and sends the results back. 3. Check the response. It should be a *json* containing all the profile information.
Feedback receiver	Roi Sucasas (ATOS)
Start date	-
End date	M16
Passed?	Passed
Bug ID	-
Problems	-
Required changes	-
Comments	-

Table 8. Integration tests for the profile initialization workflow

3.5 Sharing model

The sharing model module is responsible for the management of the properties that define the device's shareable resources (i.e. the device's resources that can be used for mF2C). As with the User Profiling module, this component should also take part in the initialization processes for IT-2. Meanwhile, in IT-1 it will work as a "standalone" module that will be used by the Lifecycle during the service initialization workflow.

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.

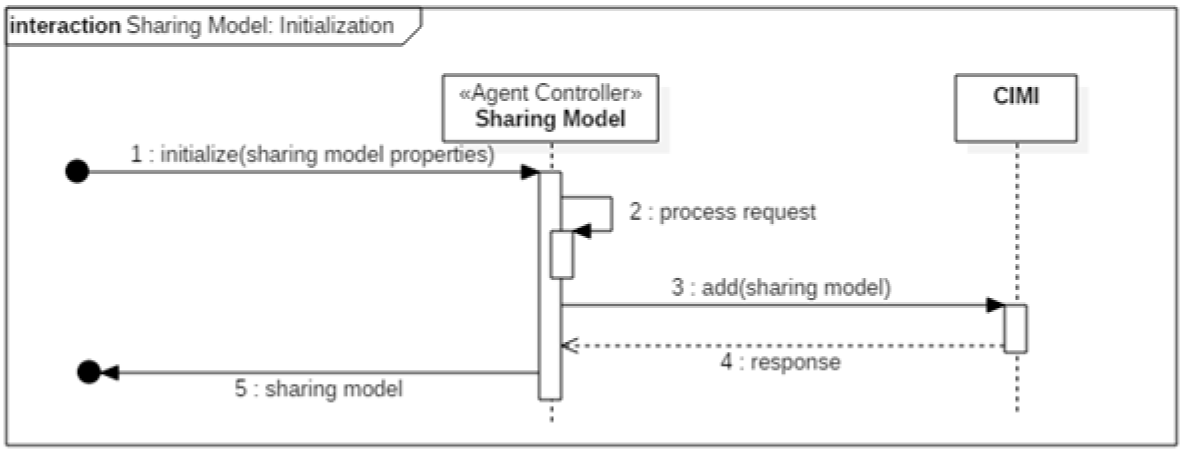


Figure 15 Configuration of shareable resources when installing mF2C software

3.5.1 Integration tests

Test Case Number Version	**TC-Sharing Model – Initialization evaluation**
Test Case Title	MF2C (IT-1): Evaluate the creation of the sharing model
Module tested	User Management module, CIMI
Related requirement(s)	One agent (with CIMI and User Management modules) deployed and running on the testbed
Target	This scenario validates the creation of a sharing model for a specific user and agent. The User Management module is responsible for processing the request, and CIMI is called to store the sharing model properties.
Initial condition(s)	- The user has been introduced in the system
Expected results	- The sharing model is created and returned
Status	Completed
Owner	ATOS
Assigned	Roi Sucasas (ATOS)
Steps	1. Send a POST request to the User Management REST API with the required parameters: sharing model properties 2. The User Management module creates the sharing model by calling CIMI, and sends the results back. 3. Check the response. It should be a *json* containing all the sharing model information.
Feedback receiver	Roi Sucasas (ATOS)
Start date	23/05/2018

End date	25/05/2018
Passed?	Passed
Bug ID	-
Problems	-
Required changes	-
Comments	-

Table 9. Integration tests for the sharing model initialization workflow

3.6 Service registration and Service Level Agreement-based QoS Analysis

This workflow demonstrates the registration of a new service to be added later to the mF2C service catalogue and the QoS-providing function in the Service Manager (see Figure 16). When a new service is registered into the mF2C system, and defined in a JSON format following a specific structure (shown later in Figure 41), the service definition is sent to the service manager through an interface in the service Categorizer. Before accepting or rejecting 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 (if any) from the SLA Manager through CIMI 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 that occurred in previous executions of that specific service. With this information, the QoS Provider uses a reinforcement learning technique to individually determine whether each one of the specified agents can be used or not. The QoS provider design block is shown in Figure 17.

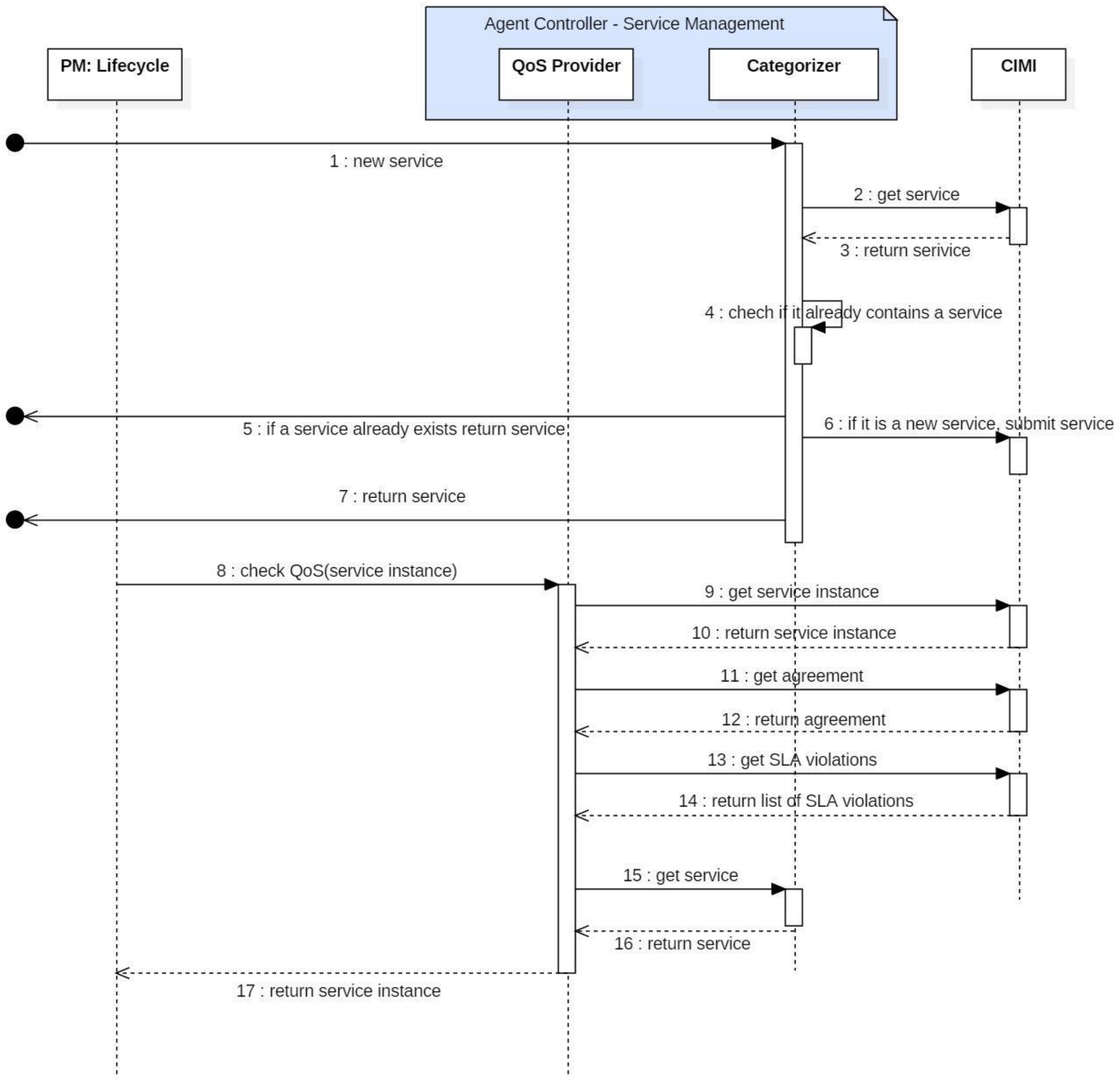

Agent Controller - Service Management
PM: Lifecycle
QoS Provider
Categorizer
CIMI
1 : new service
2 : get service
3 : return serivice
4 : chech if it already contains a service
5 : if a service already exists return service
6 : if it is a new service, submit service
7 : return service
8 : check QoS(service instance)
9 : get service instance
10 : return service instance
11 : get agreement
12 : return agreement
13 : get SLA violations
14 : return list of SLA violations
15 : get service
16 : return service
17 : return service instance

Figure 16 Service registration in the Service Manager and QoS provider

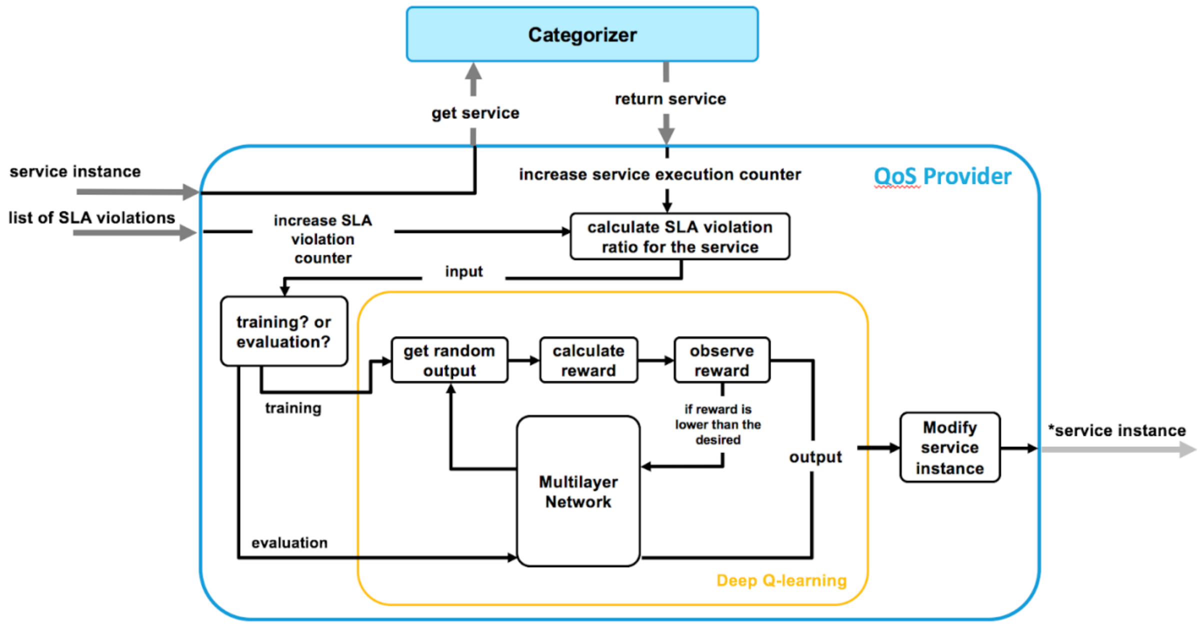


Figure 17 QoS Provider - Deep Q-learning algorithm design

In order to determine if the agents suggested 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 using the following function:

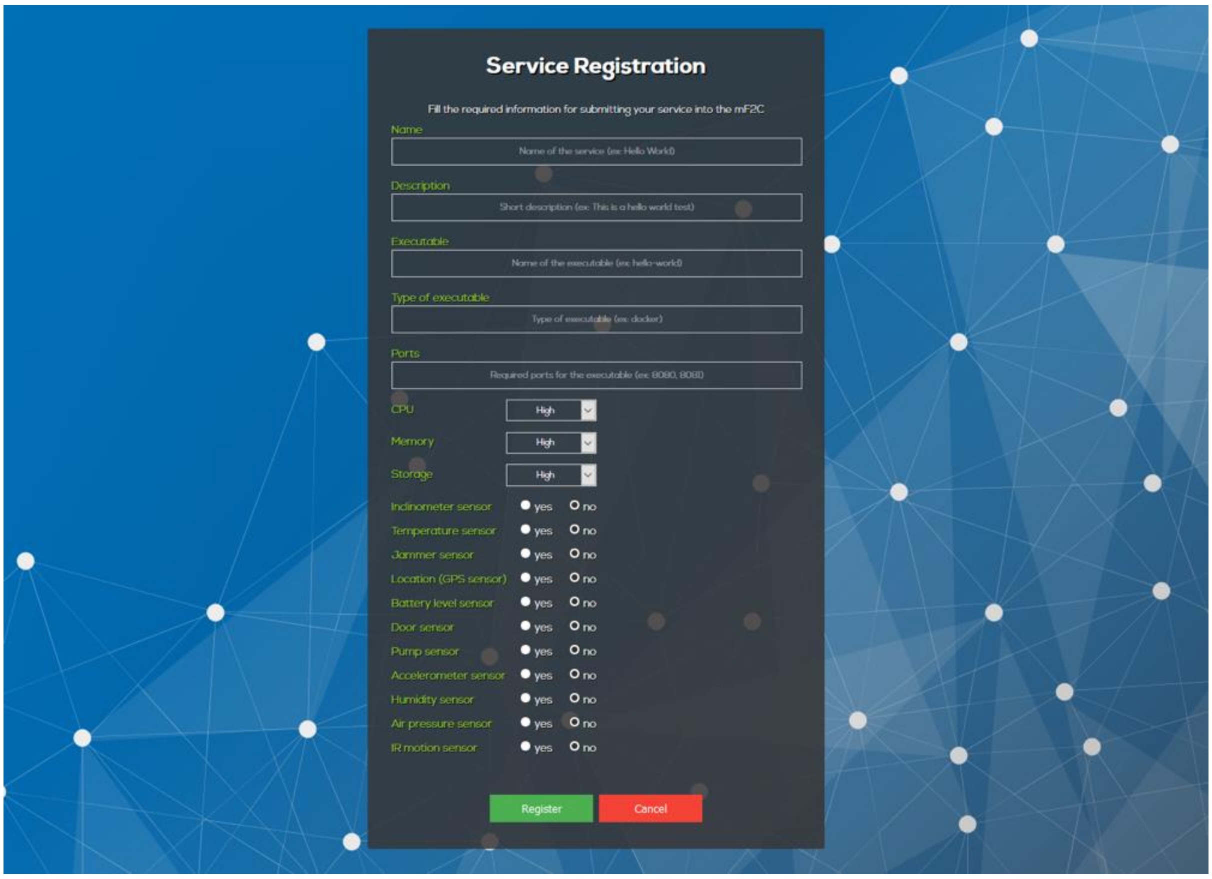
$$r_t = \sum_{n=0}^{N-1} y_n\,(-2x_s + 1) + (1 - y_n)(x_s - 1)$$

where 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 it is 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 the value is used for 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, thus 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 with other blocks.

3.6.1 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 18.



Service Registration

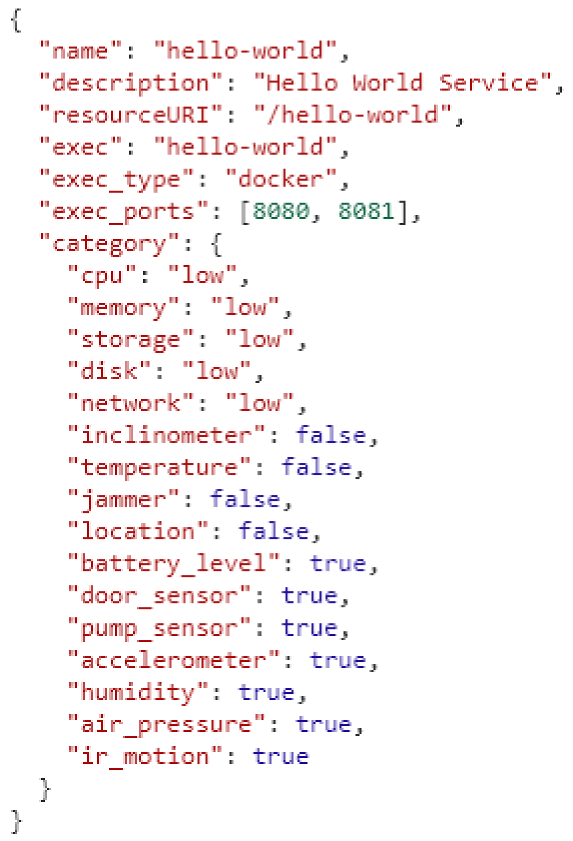
Fill the required information for submitting your service into the mF2C

Name	Name of the service (ex: Hello World)
Description	Short description (ex: This is a hello world test)
Executable	Name of the executable (ex: hello-world)
Type of executable	Type of executable (ex: docker)
Ports	Required ports for the executable (ex: 8080, 8081)
CPU	High
Memory	High
Storage	High
Inclinometer sensor	● yes ○ no
Temperature sensor	● yes ○ no
Jammer sensor	● yes ○ no
Location (GPS sensor)	● yes ○ no
Battery level sensor	● yes ○ no
Door sensor	● yes ○ no
Pump sensor	● yes ○ no
Accelerometer sensor	● yes ○ no
Humidity sensor	● yes ○ no
Air pressure sensor	● yes ○ no
IR motion sensor	● yes ○ no

Figure 18 Registration of a new service

Apart from the 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 include CPU, memory, storage as well as information about any sensors that are necessary for the service. These parameters ensure that services can 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 19.



```
{
  "name": "hello-world",
  "description": "Hello World Service",
  "resourceURI": "/hello-world",
  "exec": "hello-world",
  "exec_type": "docker",
  "exec_ports": [8080, 8081],
  "category": {
    "cpu": "low",
    "memory": "low",
    "storage": "low",
    "disk": "low",
    "network": "low",
    "inclinometer": false,
    "temperature": false,
    "jammer": false,
    "location": false,
    "battery_level": true,
    "door_sensor": true,
    "pump_sensor": true,
    "accelerometer": true,
    "humidity": true,
    "air_pressure": true,
    "ir_motion": true
  }
}
```

Figure 19 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. Figure 20 demonstrates a simple example of a service catalogue without taking a specific user into account. 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'.

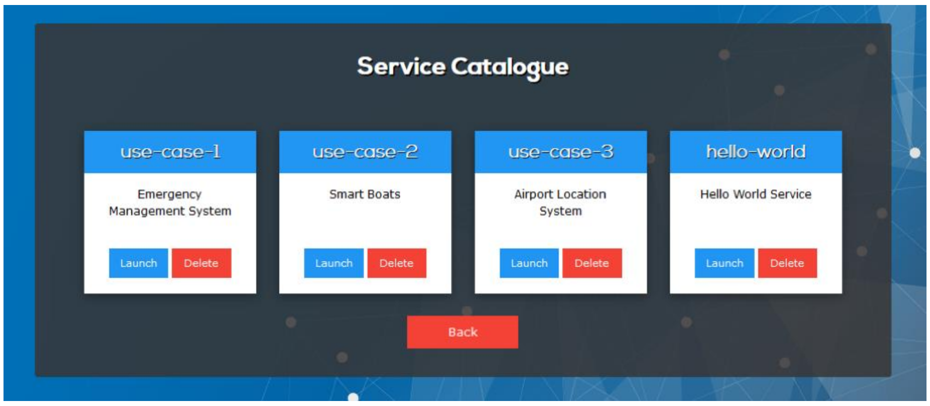


Figure 20 Service Catalogue

3.6.2 Integration tests

Test Case Number Version	**TC-Service registration and Service Level Agreement-based QoS Analysis**
Test Case Title	Service registration and Service Level Agreement-based QoS Analysis
Module tested	QoS Provider, CIMI,
Related requirement(s)	mF2C Agent running on the testbed
Target	In this scenario, the first step is to submit a service to the service manager and afterwards to check the QoS of a specific service. After the service instance is submitted, the mF2C system will run the service and afterwards will generate a service-operation-report with the time spent running the service. For testing purposes, we submit reports specifying a service execution time much longer than the time specified in the agreement in order to artificially create SLA violations. After this submission, the SLA management will generate violations, that the QoS provider block will use to determine which agents can be used for the next execution of the same service. Then, for the service execution, the Lifecycle will check the QoS, specifying the id. The output will be a service instance specifying the agents that can be used again for the service execution.
Initial condition(s)	- No preconditions for Service registration - For QoS analysis, a valid agreement has to be previously linked with service instance
Expected results	- A service arrives into the system and if it is new, it goes through the process of categorization. - A new service is submitted to CIMI. - For a new service instance, the Lifecycle starts the QoS analysis for that service. - The QoS Provider obtains the service instance through CIMI. - For the service instance, the QoS provider gets the SLA agreement information from CIMI (which is previously evaluated by the SLA Management using the stored execution times). - The QoS Provider gets the SLA violation information from CIMI (SLA violations are stored in CIMI).
Status	Completed
Owner	TUBS
Assigned	Francisco Carpio (TUBS)

Steps	1. Submit a service with a POST request to the Service Manager REST API (*/api/service-management/categorizer*) with the required parameters defined later in section 3.6.1. 2. The Service Management Categorizer returns information on *service_id* and categorizes the submitted service. 3. The next step is testing the QoS. For this to be achieved, it is necessary to first submit an agreement and a service instance to CIMI 3.1 Submit an agreement with a POST request to CIMI (*/api/agreement*) with required parameters. The reply is an agreement_id. 3.2 Submit a Service Instance specifying the *<service-id>* and the *<agreement-id>* with a POST request to CIMI (*/api/service-instance*) with required parameters. The reply is service-instance_id. 3.3 After the service instance is submitted, the mF2C system will run the service and afterwards will generate a service-operation-report with the time spent running the service. For testing purposes, we submit with a POST request to CIMI reports specifying a service execution time much longer than that specified in the agreement in order to artificially create SLA violations (*/api/service-operation-report*) 4. After this submission, the SLA management will generate violations, that the QoS provider block will use to determine which agents can be used for the next execution of the same service. Then, for the service execution, the Lifecycle will check the QoS, specifying the id. This is achieved with GET request */api/service-management/qos/<service-instance-id>*. The output will be a service instance specifying the agents that can be used again for the service execution.
Feedback receiver	ATOS
Start date	M13
End date	M17
Passed?	Yes
Bug ID	-
Problems	-
Required changes	-
Comments	Tested with the last versions of these two components.

Table 10. Integrations tests for the Service registration and Service Level Agreement-based QoS Analysis

3.7 Lifecycle Management

Once there is an mF2C cluster consisting of one or more agents, the Lifecycle Management (often shortened to “Lifecycle”) enters the scene. This component is responsible for managing the lifecycle of the services' instances executed by the mF2C platform. This management includes the process of deploying one or more services in the agents (**service initialization**), the termination of these services (**service termination**), and all the basic operations: start / stop a service, and start a job in COMPSs (**service operation**).

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, the Distributed Execution Runtime component (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 initialization of a service instance in an mF2C cluster. This workflow (Figure 21) is started when a user wants to initialize a service to a set of devices. The Lifecycle of the agent that gets this request is responsible for the following workflow.

First, the ***Lifecycle*** processes this request (‘initialize 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* (step 3 in the workflow below), 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. When Recommender receives the request from Lifecycle it forwards it to the Landscaper. The Landscaper returns to the Recommender the list of available devices in the cluster that can satisfy the recipe (step 4 in the workflow below). The Recommender selects from this list the optimal devices (agents) to execute the requested service.

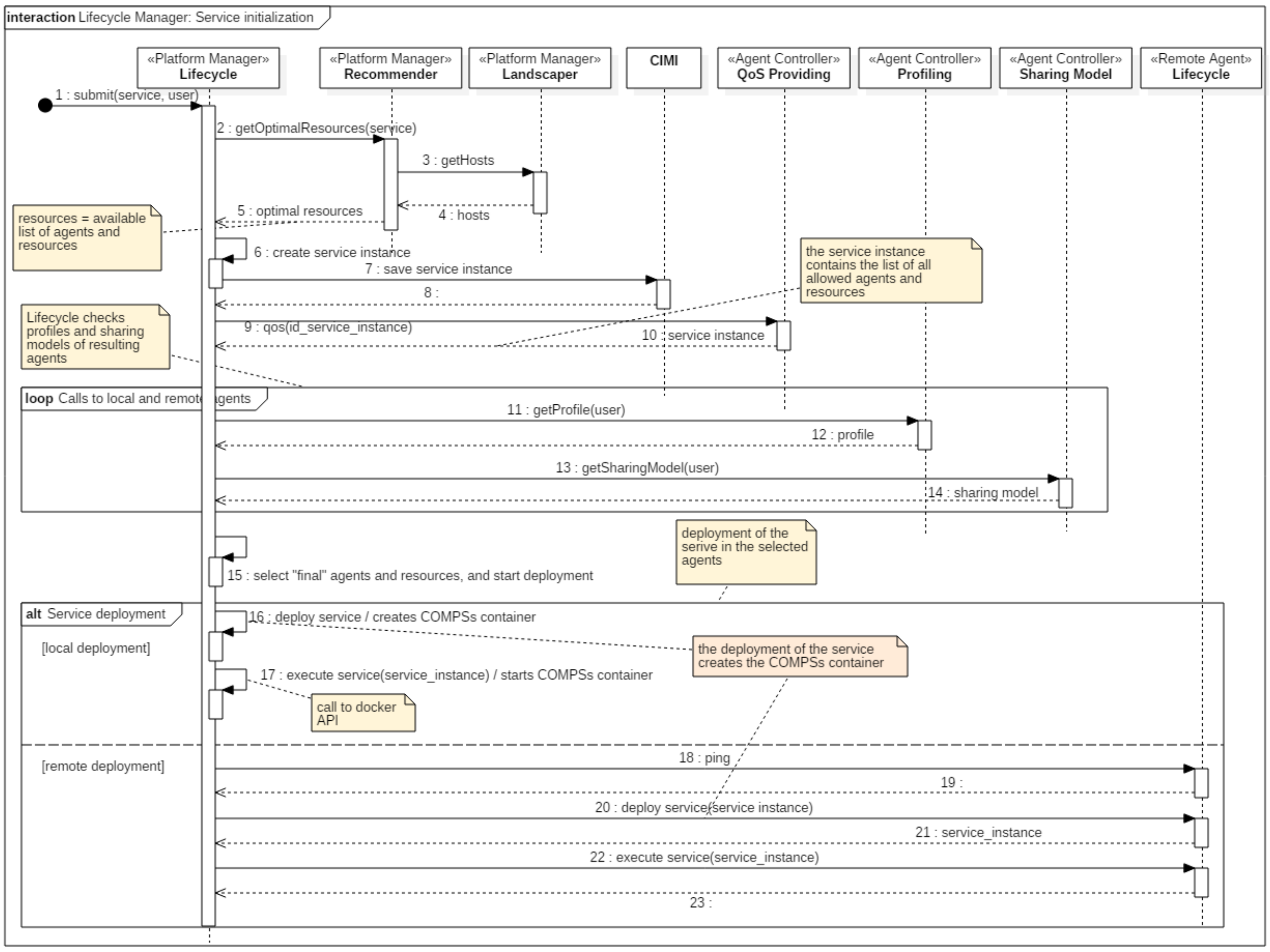

interaction Lifecycle Manager: Service initialization
«Platform Manager» Lifecycle
«Platform Manager» Recommender
«Platform Manager» Landscaper
CIMI
«Agent Controller» QoS Providing
«Agent Controller» Profiling
«Agent Controller» Sharing Model
«Remote Agent» Lifecycle
1 : submit(service, user)
2 : getOptimalResources(service)
3 : getHosts
4 : hosts
5 : optimal resources
resources = available list of agents and resources
6 : create service instance
7 : save service instance
8 :
the service instance contains the list of all allowed agents and resources
9 : qos(id_service_instance)
10 : service instance
Lifecycle checks profiles and sharing models of resulting agents
loop Calls to local and remote agents
11 : getProfile(user)
12 : profile
13 : getSharingModel(user)
14 : sharing model
deployment of the serive in the selected agents
15 : select "final" agents and resources, and start deployment
alt Service deployment
[local deployment]
16 : deploy service / creates COMPSs container
the deployment of the service creates the COMPSs container
17 : execute service(service_instance) / starts COMPSs container
call to docker API
[remote deployment]
18 : ping
19 :
20 : deploy service(service instance)
21 : service_instance
22 : execute service(service_instance)
23 :

Figure 21 Lifecycle Management - Service initialization

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 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 which is included in the service instance object that is created during the service initialization workflow, step 15 in the 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 is included in the initialization phase. This operation includes both a call to CIMI in order to update the status of the service instance object, and a call to the SLA Manager to start the SLA Agreement process.

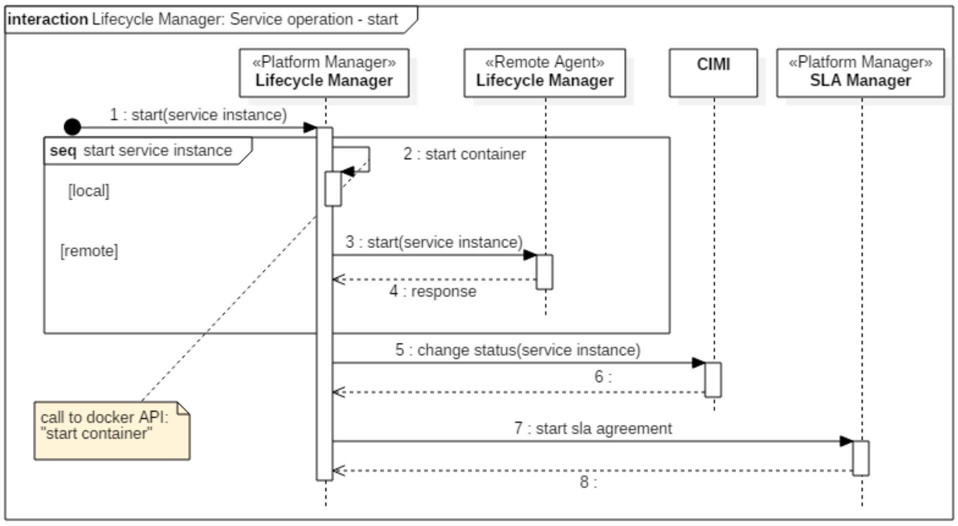

interaction Lifecycle Manager: Service operation - start
«Platform Manager» Lifecycle Manager
«Remote Agent» Lifecycle Manager
CIMI
«Platform Manager» SLA Manager
1 : start(service instance)
seq start service instance
[local]
2 : start container
[remote]
3 : start(service instance)
4 : response
5 : change status(service instance)
6 :
7 : start sla agreement
8 :
call to docker API: "start container"

Figure 22 Start a service

- Stop a service: this operation stops a service, updates the status of the service instance object, and finally stops the SLA Agreement process (see Figure 23).

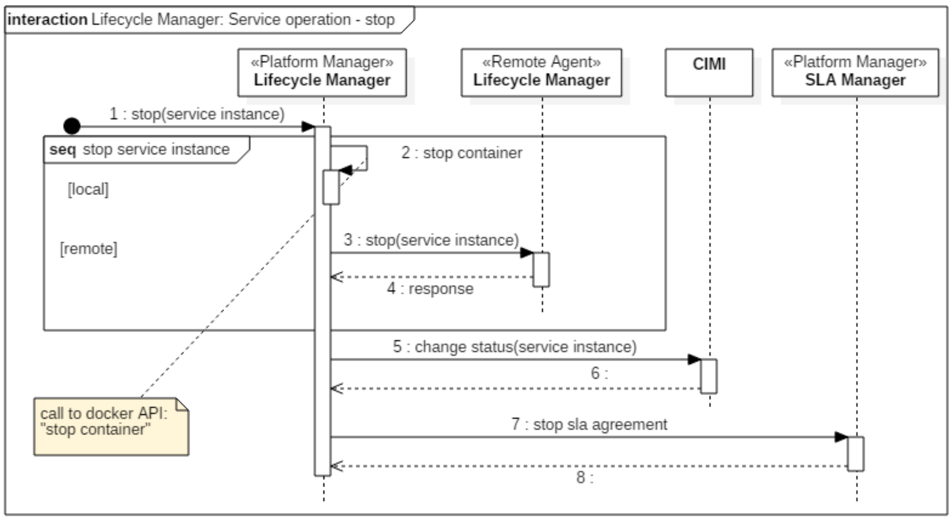

interaction Lifecycle Manager: Service operation - stop
«Platform Manager» Lifecycle Manager
«Remote Agent» Lifecycle Manager
CIMI
«Platform Manager» SLA Manager
1 : stop(service instance)
seq stop service instance
[local]
2 : stop container
[remote]
3 : stop(service instance)
4 : response
5 : change status(service instance)
6 :
7 : stop sla agreement
8 :
call to docker API: "stop container"

Figure 23 Stop a service

- Execute a job in COMPSs: 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 24).

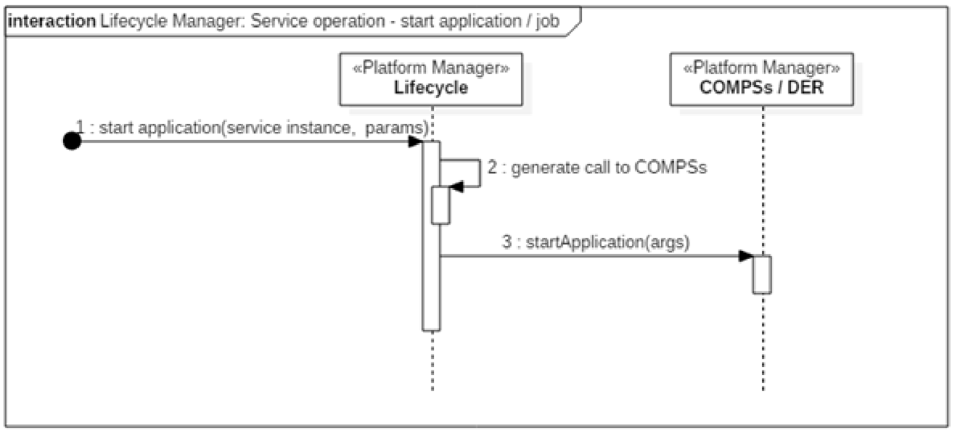

interaction Lifecycle Manager: Service operation - start application / job
«Platform Manager» Lifecycle
«Platform Manager» COMPSs / DER
1 : start application(service instance, params)
2 : generate call to COMPSs
3 : startApplication(args)

Figure 24 Service operation

Service termination

The following workflow (Figure 25) 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.

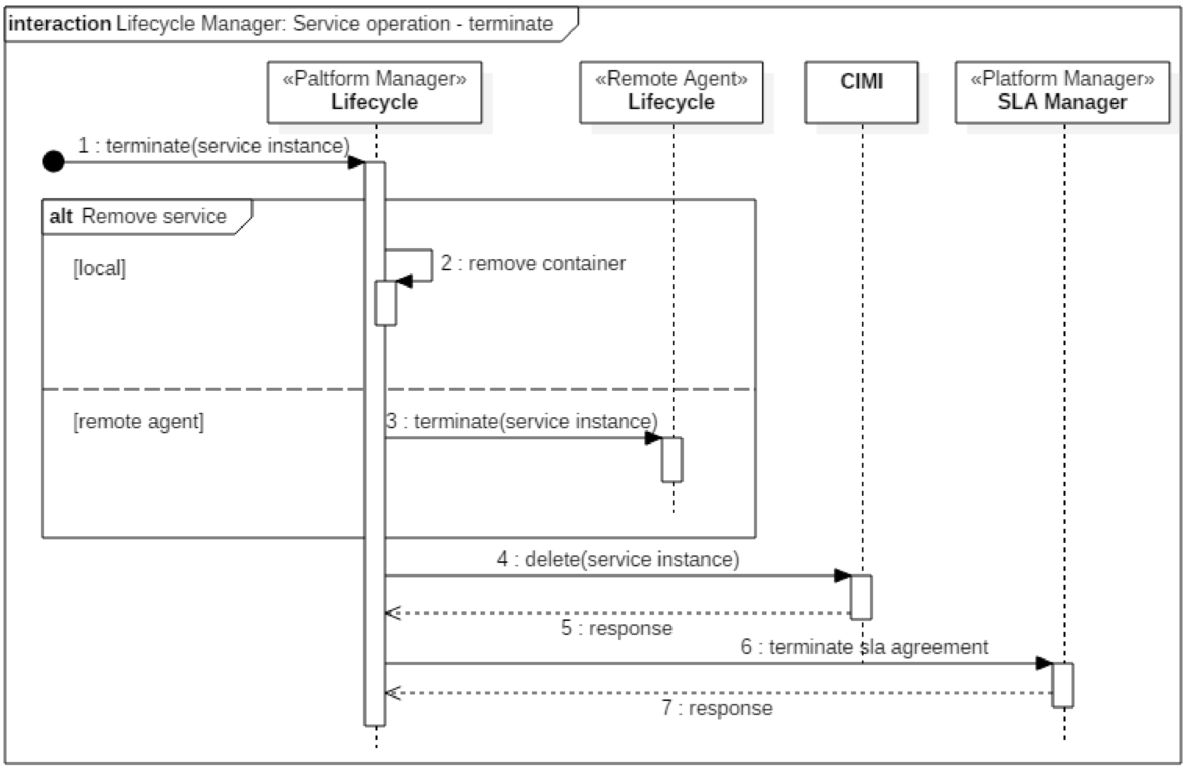

interaction Lifecycle Manager: Service operation - terminate
«Paltform Manager» Lifecycle
«Remote Agent» Lifecycle
CIMI
«Platform Manager» SLA Manager
1 : terminate(service instance)
alt Remove service
[local]
2 : remove container
[remote agent]
3 : terminate(service instance)
4 : delete(service instance)
5 : response
6 : terminate sla agreement
7 : response

Figure 25 Lifecycle Management - Service termination

3.7.1 Integration tests

Test Case Number Version	TC-Lifecycle – Service initialization evaluation (I)

Test Case Title	MF2C (IT-1): Evaluate the deployment of a service based in **COMPSs** in a mF2C cluster
Module tested	Lifecycle, User Management, SLA Manager, Service Manager, Recommender & Landscaper, DER (COMPSs), CIMI
Related requirement(s)	2 or more agents running on the testbed
Target	In this scenario, a COMPSs service will be submitted or deployed in a set of agents that are part of a mF2C cluster. First, the Lifecycle process the 'submission' request and calls the Recommender & Landscaper tools to get a list of the available agents. Second, the Lifecycle calls the Service Manager and the User Management modules to filter this list. Then, the Lifecycle creates a **service instance** object by calling CIMI. And finally, the Lifecycle deploys the service in the resulting agents.
Initial condition(s)	○ The service has been introduced in the system
Expected results	○ A new service instance is created ○ The service is deployed and started in one or more agents ○ The service instance is returned
Status	Completed
Owner	ATOS
Assigned	Roi Sucasas (ATOS)
Steps	1. Send a POST request to the Lifecycle REST API (*/api/v1/lifecycle*) with the required parameters: <u>service identifier</u>, <u>SLA agreement identifier</u>, <u>user identifier</u> 2. The Lifecycle gets, from the Landscaper & Recommender, a list of mF2C agents where this service can be deployed (*docker* containers) 3. The Lifecycle calls the Service Manager to filter this list 4. The Lifecycle calls the User Management of the resulting agents to check them 5. The Lifecycle creates a service instance object 6. The Lifecycle calls all the agents' Lifecycles of this resulting list in order to deploy and start the service 7. The Lifecycle calls the SLA Manager in order to start the corresponding SLA agreement 8. The service instance is updated 9. The Lifecycle returns a response with the results of the operation
Feedback receiver	Roi Sucasas (ATOS), Román Sosa (ATOS), BSC, INTEL, TUBS
Start date	28/05/2018

End date	01/06/2018
Passed?	Passed
Bug ID	-
Problems	-
Required changes	-
Comments	-

Table 11. Integration tests for a COMPSs service initialization on the Lifecycle manager

Test Case Number Version	**TC-Lifecycle – Service initialization evaluation (II)**
Test Case Title	MF2C (IT-1): Evaluate the deployment of a service (docker image or docker-compose) in a mF2C cluster
Module tested	Lifecycle, User Management, SLA Manager, Service Manager, Recommender & Landscaper, CIMI
Related requirement(s)	One or more agents running on the testbed
Target	In this scenario, a service will be deployed and started in a set of agents that are part of a mF2C cluster. First, the Lifecycle process the 'submission' request and calls the Recommender & Landscaper tools to get a list of the available mF2C agents. Second, the Lifecycle calls the Service Manager and the User Management modules to filter this list of agents. Then, the Lifecycle creates a **service instance** object by calling CIMI. And finally, the Lifecycle deploys and starts the service in the resulting agents.
Initial condition(s)	○ The service has been introduced in the system
Expected results	○ A new service instance is created ○ The service instance is deployed and started in **one** mF2C agent ○ The content of the service instance is returned in the response
Status	Completed
Owner	ATOS
Assigned	Roi Sucasas (ATOS)
Steps	1. Send a POST request to the Lifecycle REST API (*/api/v1/lifecycle*) with the required parameters: service identifier, SLA agreement identifier, user identifier

	2. The Lifecycle gets, from the Landscaper & Recommender, a list of mF2C agents where this service can be deployed (*docker* containers) 3. The Lifecycle calls the Service Manager to filter this list 4. The Lifecycle calls the User Management of the resulting agents to check them 5. The Lifecycle creates a service instance object 6. Lifecycle selects the first element (optimal) of the resulting agents list 7. The Lifecycle calls this agent's Lifecycle of in order to deploy and start the service 8. The Lifecycle calls the SLA Manager in order to start the corresponding SLA agreement 9. The service instance is updated 10. The Lifecycle returns a response with the results of the operation
Feedback receiver	Roi Sucasas (ATOS), Román Sosa (ATOS), BSC, INTEL, TUBS
Start date	28/05/2018
End date	01/06/2018
Passed?	Passed
Bug ID	-
Problems	-
Required changes	-
Comments	-

Table 12. Integration tests for a service (Docker container) initialization on the Lifecycle manager

Test Case Number Version	**TC-Lifecycle – Service operation evaluation (I)**
Test Case Title	MF2C (IT-1): Evaluate part of the lifecycle (start, stop) of a service instance
Module tested	Lifecycle, SLA Manager, CIMI
Related requirement(s)	One or more agents and one service instance running on the testbed
Target	In this scenario, a service instance will be stopped / started. First, the Lifecycle will process the request. And then, it calls all the agents' Lifecycles (where this service instance is deployed) in order to stop or start the service instance.
Initial condition(s)	○ A service instance is deployed in one or more mF2C agents
Expected results	○ All the *docker* containers that correspond to this service instance are stopped / started. ○ The SLA agreement is stopped / started

Status	Completed
Owner	ATOS
Assigned	Roi Sucasas (ATOS)
Steps	1. Send a PUT request to the Lifecycle REST API (*/api/v1/lifecycle*) with the required parameters: <u>service instance identifier</u>, <u>operation</u> (start, stop, restart) 2. The Lifecycle gets the list of all the agents where this service instance is deployed (*docker* containers) 3. The Lifecycle calls all these agents in order to stop or start the corresponding *docker* containers 4. The Lifecycle calls the SLA Manager in order to stop or start the corresponding SLA agreement 5. The Lifecycle returns a response with the results of the operation
Feedback receiver	Roi Sucasas (ATOS), Román Sosa (ATOS), BSC
Start date	28/05/2018
End date	01/06/2018
Passed?	Passed
Bug ID	-
Problems	-
Required changes	-
Comments	-

Table 13. Integration tests for the lifecycle management of a service

Test Case Number Version	**TC-Lifecycle – Service operation evaluation (II)**
Test Case Title	MF2C (IT-1): Evaluate the execution of a job in COMPSs
Module tested	Lifecycle, DER (COMPSs), CIMI
Related requirement(s)	One or more mF2C agents and one service instance running on the testbed. This service instance has to correspond to a service based on COMPSs.
Target	In this scenario, a job will be executed in the DER. First, the Lifecycle will process the request. And then, it calls COMPSs to execute a job.
Initial condition(s)	○ A (COMPSs) service instance is running in one or more mF2C agents

Expected results	○ COMPSs (master) executes and distributes the job among all the workers agents
Status	Completed
Owner	ATOS
Assigned	Roi Sucasas (ATOS)
Steps	1. Send a PUT request to the Lifecycle REST API (*/api/v1/lifecycle*) with the required parameters: service instance identifier, operation (*start-job*), and a XML (COMPSs specific) 2. The Lifecycle gets the list of all the agents where this service instance is deployed (*docker* containers) 3. The Lifecycle calls one of these agents in order to communicate with the COMPSs REST API 4. The Lifecycle calls this COMPSs REST API to launch the job 5. The Lifecycle returns a response with the results of the operation
Feedback receiver	Roi Sucasas (ATOS), BSC
Start date	28/05/2018
End date	01/06/2018
Passed?	Passed
Bug ID	-
Problems	-
Required changes	-
Comments	-

Table 14. Integration tests for executing jobs in COMPSs

Test Case Number Version	**TC-Lifecycle – Service termination evaluation**
Test Case Title	MF2C (IT-1): Evaluate the termination of a service instance
Module tested	Lifecycle, SLA Manager, CIMI
Related requirement(s)	One or more agents and one service instance running on the testbed
Target	In this scenario, a service instance will be terminated. First, the Lifecycle will process the 'termination' request. And then, it calls all the agents' Lifecycles (where this service instance is running) in order to stop and terminate the service instance.
Initial condition(s)	○ A service instance is running in one or more mF2C agents

Expected results	○ All the *docker* containers that correspond to this service instance are stopped and removed from the agents. ○ The SLA agreement is terminated ○ The service instance object is deleted by calling CIMI
Status	Completed
Owner	ATOS
Assigned	Roi Sucasas (ATOS)
Steps	1. Send a DELETE request to the Lifecycle REST API (*/api/v1/lifecycle*) with the required parameters: service instance identifier 2. The Lifecycle gets the list of all the agents where this service instance is running or is deployed (*docker* containers) 3. The Lifecycle calls all these agents in order to stop and remove the corresponding *docker* containers 4. The Lifecycle calls the SLA Manager in order to terminate the corresponding SLA agreement 5. The Lifecycle returns a response with the results of the operation
Feedback receiver	Roi Sucasas (ATOS)
Start date	28/05/2018
End date	01/06/2018
Passed?	Passed
Bug ID	-
Problems	-
Required changes	-
Comments	The call to the SLA Manager in order to terminate the SLA agreement has been skipped for simplifying the tests.

Table 15. Integration tests for the termination of a service

3.8 Landscaper

Figure 26 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 Landscaper from DataClay (Device class). Landscaper also requests the list of deployed services from Lifecycle, and 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.

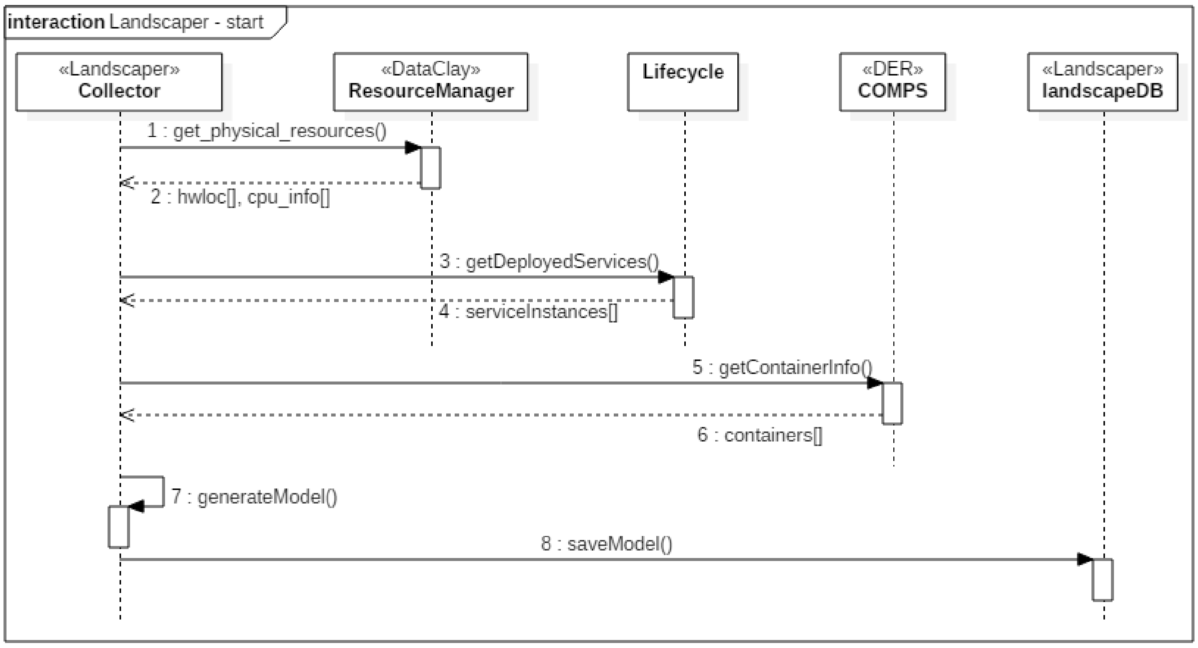


Figure 26 Landscaper start

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.

3.8.1 Integration tests

Test Case Number Version	**TC-Landscaper – Generating the initial model**
Test Case Title	Generate an initial model of devices and services for this mF2C cluster
Module tested	Landscaper
Related requirement(s)	
Target	
Initial condition(s)	CIMI service is started. CIMI contains a number of Device objects Deploy a number of Services/Containers CIMI contains Service_Instance objects for those services
Expected results	- Each device from CIMI is added to the landscape - Each service deployed is added to landscape - Container info for deployed services is added and mapped to the hosts

Status	Completed
Owner	Alec Leckey (Intel)
Assigned	Alec Leckey (Intel)
Steps	Start the Landscaper container
Feedback receiver	None
Start date	1 June
End date	1 June
Passed?	PASS – devices successfully added PASS – Local containers and services added only FAIL – Services instances from CIMI are not mapped to actual services queried from Docker API
Bug ID	
Problems	Only Docker containers and Services deployed locally are added to the Landscape. No containers remotely deployed are added. As such, ServiceInstance objects referring to containers deployed remotely are missing.
Required changes	Integration with a Docker orchestration tool, eg., Docker Swarm or Kubernetes would make it easier to query container metadata.
Comments	Lifecycle is currently manually deploying containers to remote hosts, i.e., Lifecycle Leader → Lifecycle child → deploy container. Possible change would be Lifecycle Parent (eg, through Swarm) → deploy container on child.

Table 16. Integration test for the start of the Landscaper

3.9 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 Providing component to provide better recommendations about the agents to be used on next executions of the same service (see section 3.6). An example of an agreement for IT-1 is shown in Figure 27, while the sequence diagram that details the evaluation process is shown in Figure 28. The evaluation process is executed periodically (e.g., every minute) and gets the monitoring information stored by execution time records which 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 27 Example of SLA agreement

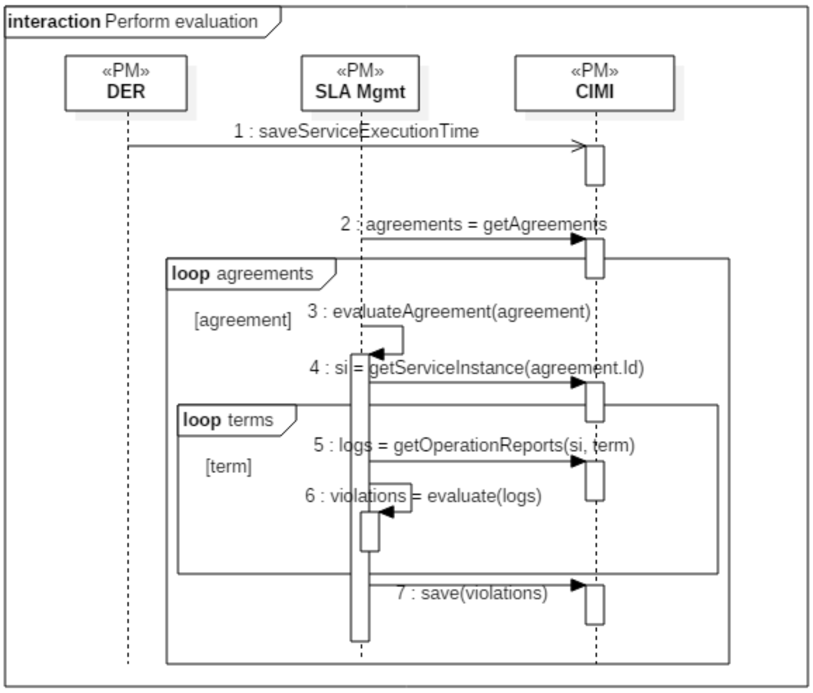

interaction Perform evaluation
«PM» DER
«PM» SLA Mgmt
«PM» CIMI
1 : saveServiceExecutionTime
2 : agreements = getAgreements
loop agreements
[agreement]
3 : evaluateAgreement(agreement)
4 : si = getServiceInstance(agreement.Id)
loop terms
[term]
5 : logs = getOperationReports(si, term)
6 : violations = evaluate(logs)
7 : save(violations)

Figure 28 SLA Management – Evaluate agreement

3.9.1 Integration tests

Test Case Number Version	**TC-SLA Evaluation**
Test Case Title	Evaluate an SLA Agreement on a COMPSs application

Module tested	SLA Management, Lifecycle Management, Distributed Execution Runtime
Related requirement(s)	SLA Management, Lifecycle Management and Service Manager are running on a single agent.
Target	This scenario tests the correct evaluation of SLA agreements of a service that runs a COMPSs application that does not use dataClay. The SLA agreement checks the execution time of one of the operations served by the application. The test also involves the creation and categorization of the service, the instantiation of the service and the deployment on a single agent and the operation call. This means that all these components must be properly integrated for a successful test. This test does not make use of Recommender and Landscaper: the addresses of agents are passed to Lifecycle.
Initial condition(s)	- The service has been introduced in the system and has been properly categorized. - A valid agreement to be assigned to the service instance has been introduced in the system.
Expected results	- The agreement is started - The DER stores operation execution times on CIMI. - SLA Management evaluates the SLA agreement using the stored execution times - SLA Violations are stored on CIMI if execution times do not satisfy the QoS constraint in the agreement.
Status	Done
Owner	Román Sosa (ATOS)
Assigned	Román Sosa (ATOS)
Steps	1. Start service instance 2. Start service operation 3. Wait until operation finishes and SLA evaluation has been performed 4. Check if violation has been generated.
Feedback receiver	Román Sosa, Roi Sucasas (ATOS), Francesc Lordan, Daniele Lezzi (BSC)
Start date	23/05/2018
End date	29/05/2018
Passed?	Yes
Bug ID	-
Problems	None
Required changes	List of changes until test passed:

	- DER needs to accept URL of CIMI repository from service instance initialization - Use non-https CIMI address until we have non-self-signed certificates.
Comments	

Table 17. Integration tests for the SLA manager on a COMPSs application

Test Case Number Version	**TC-SLA Evaluation COMPSs DataClay**
Test Case Title	Evaluate an SLA Agreement on a COMPSs+dataClay application
Module tested	SLA Management, Lifecycle Management, Distributed Execution Runtime, DataClay
Related requirement(s)	SLA Management, Lifecycle Management, DataClay and Service Manager are running on a single agent.
Target	This scenario tests the correct evaluation of SLA agreements of a service that runs a COMPSs application that uses dataClay. The SLA agreement checks the execution time of one of the operations served by the application. The test also involves the creation and categorization of the service, the instantiation of the service and the deployment on two agents and the operation call. This means that all these components must be properly integrated for a successful test. This test does not make use of Recommender and Landscaper: the addresses of agents are passed to Lifecycle.
Initial condition(s)	- The service has been introduced in the system and has been properly categorized. - A valid agreement to be assigned to the service instance has been introduced in the system.
Expected results	- The agreement is started - The DER stores operation execution times on CIMI. - SLA Management evaluates the SLA agreement using the stored execution times - SLA Violations are stored on CIMI if execution times do not satisfy the QoS constraint in the agreement.
Status	Finished
Owner	Román Sosa (ATOS)
Assigned	Román Sosa (ATOS)
Steps	1. Start service instance 2. Start service operation 3. Wait until operation finishes and SLA evaluation has been performed 4. Check if violation has been generated.

Feedback receiver	Román Sosa, Roi Sucasas (ATOS), Francesc Lordan, Daniele Lezzi, Anna Queralt, Jonathan Martí (BSC)
Start date	31/05/2018
End date	05/06/2018
Passed?	Yes on a single agent.
Bug ID	-
Problems	When deploying on more than one agent, the DataClay master needs to be informed about the DataClay slaves. This is not done automatically by the mF2C platform at the moment.
Required changes	-
Comments	Tests passed deploying just on one agent. The multiple-agent DataClay configuration will be setup manually. No other issues are expected to pass the test.

Table 18. Integration tests for the SLA manager evaluation on a COMPSs application using DataClay

3.10 Distributed Execution Runtime

The implementation of the DER component includes a Start App method that requires a list of resources on which 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 postponed to IT-2.

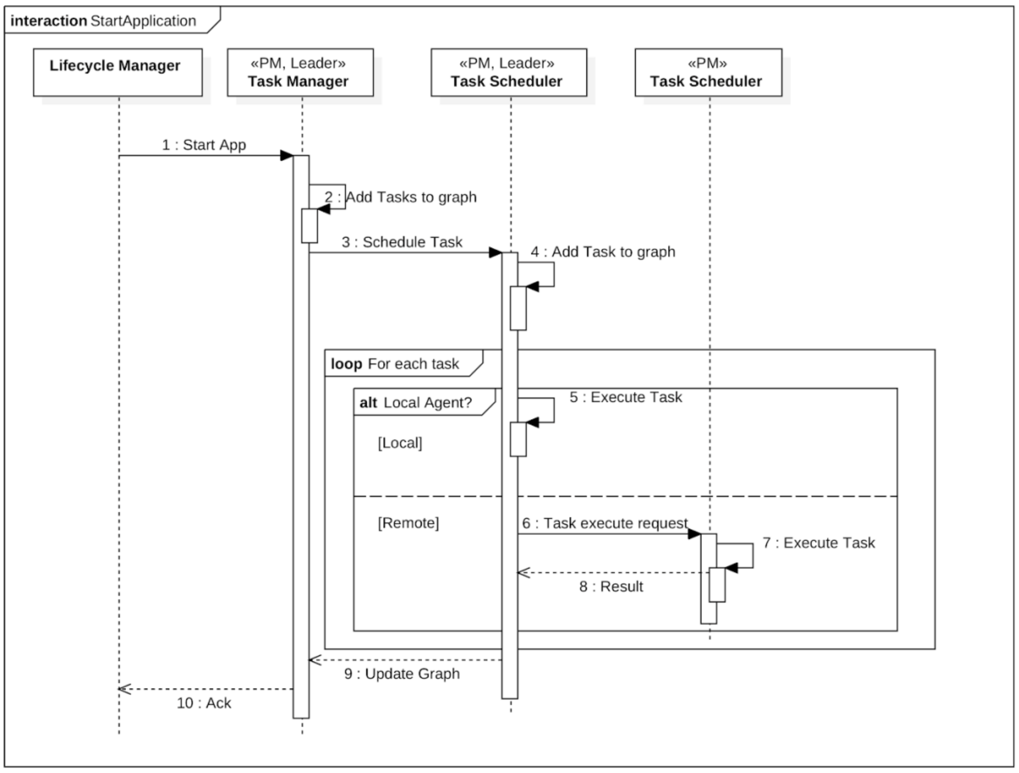
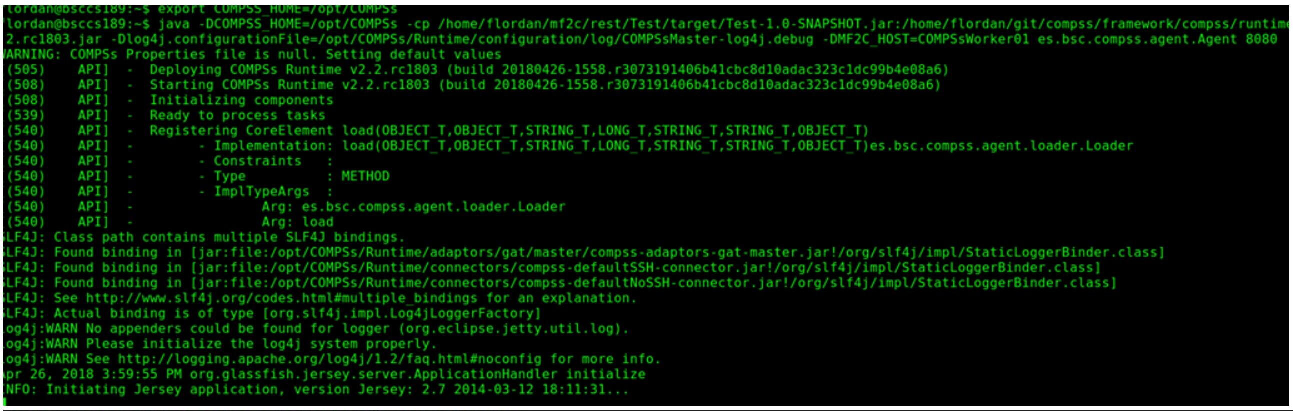
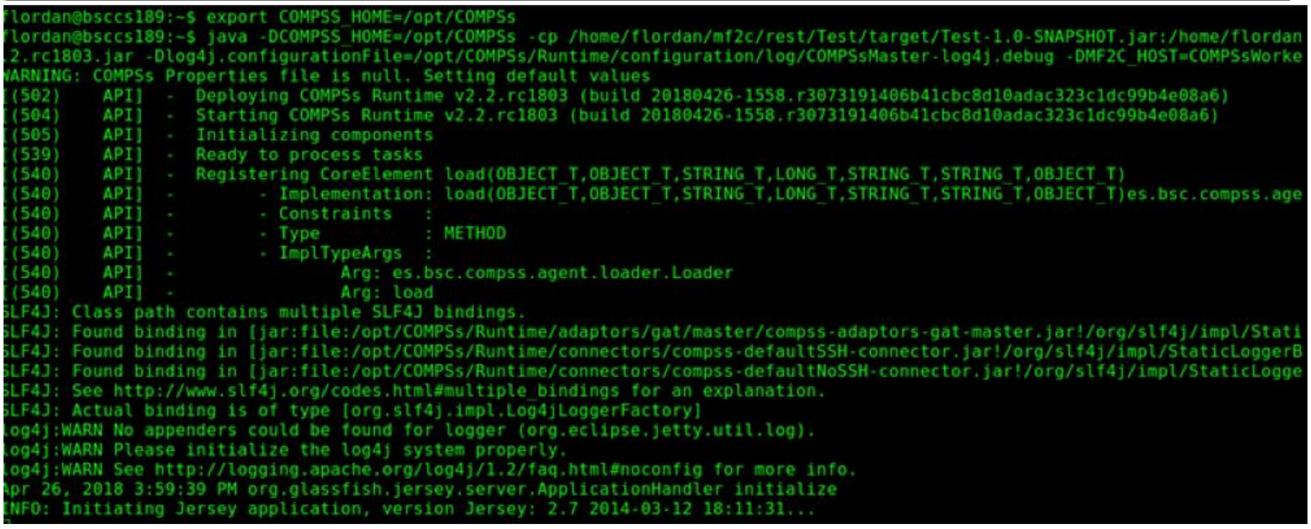


Figure 29 Distributed Execution Runtime – Start application

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 30 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.



```
lordan@bsccs189:~$ export COMPSS_HOME=/opt/COMPSs
lordan@bsccs189:~$ java -DCOMPSS_HOME=/opt/COMPSs -cp /home/flordan/mf2c/rest/Test/target/Test-1.0-SNAPSHOT.jar:/home/flordan/git/compss/framework/compss/runtime
2.rc1803.jar -Dlog4j.configurationFile=/opt/COMPSs/Runtime/configuration/log/COMPSsMaster-log4j.debug -DMF2C_HOST=COMPSsWorker01 es.bsc.compss.agent.Agent 8080
ARNING: COMPSs Properties file is null. Setting default values
(505)    API]  -  Deploying COMPSs Runtime v2.2.rc1803 (build 20180426-1558.r3073191406b41cbc8d10adac323c1dc99b4e08a6)
(508)    API]  -  Starting COMPSs Runtime v2.2.rc1803 (build 20180426-1558.r3073191406b41cbc8d10adac323c1dc99b4e08a6)
(508)    API]  -  Initializing components
(539)    API]  -  Ready to process tasks
(540)    API]  -  Registering CoreElement load(OBJECT_T,OBJECT_T,STRING_T,LONG_T,STRING_T,STRING_T,OBJECT_T)
(540)    API]  -        - Implementation: load(OBJECT_T,OBJECT_T,STRING_T,LONG_T,STRING_T,STRING_T,OBJECT_T)es.bsc.compss.agent.loader.Loader
(540)    API]  -        - Constraints    :
(540)    API]  -        - Type           : METHOD
(540)    API]  -        - ImplTypeArgs   :
(540)    API]  -                Arg: es.bsc.compss.agent.loader.Loader
(540)    API]  -                Arg: load
LF4J: Class path contains multiple SLF4J bindings.
LF4J: Found binding in [jar:file:/opt/COMPSs/Runtime/adaptors/gat/master/compss-adaptors-gat-master.jar!/org/slf4j/impl/StaticLoggerBinder.class]
LF4J: Found binding in [jar:file:/opt/COMPSs/Runtime/connectors/compss-defaultSSH-connector.jar!/org/slf4j/impl/StaticLoggerBinder.class]
LF4J: Found binding in [jar:file:/opt/COMPSs/Runtime/connectors/compss-defaultNoSSH-connector.jar!/org/slf4j/impl/StaticLoggerBinder.class]
LF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
LF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
og4j:WARN No appenders could be found for logger (org.eclipse.jetty.util.log).
og4j:WARN Please initialize the log4j system properly.
og4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
pr 26, 2018 3:59:55 PM org.glassfish.jersey.server.ApplicationHandler initialize
NFO: Initiating Jersey application, version Jersey: 2.7 2014-03-12 18:11:31...
```

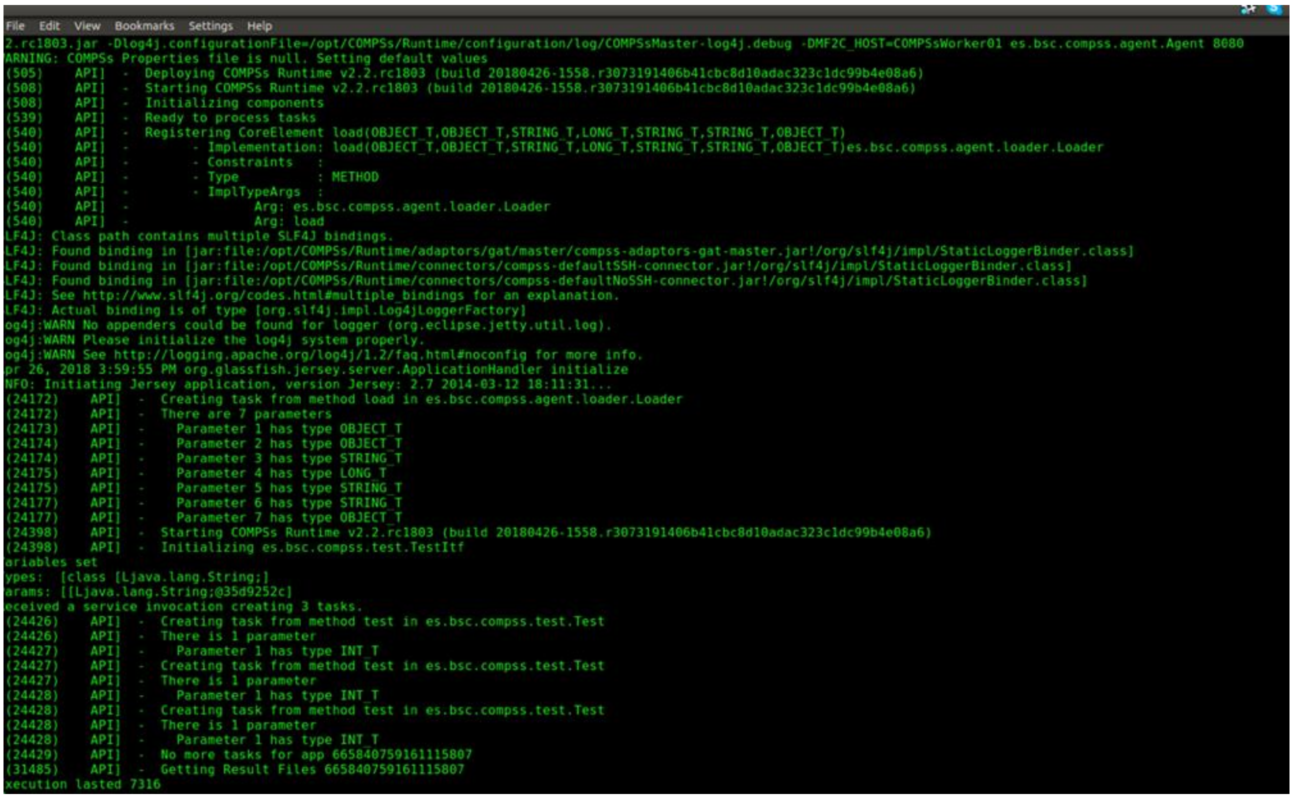


```
flordan@bsccs189:~$ export COMPSS_HOME=/opt/COMPSs
flordan@bsccs189:~$ java -DCOMPSS_HOME=/opt/COMPSs -cp /home/flordan/mf2c/rest/Test/target/Test-1.0-SNAPSHOT.jar:/home/flordan
.2.rc1803.jar -Dlog4j.configurationFile=/opt/COMPSs/Runtime/configuration/log/COMPSsMaster-log4j.debug -DMF2C_HOST=COMPSsWorke
WARNING: COMPSs Properties file is null. Setting default values
[(502)    API]  -  Deploying COMPSs Runtime v2.2.rc1803 (build 20180426-1558.r3073191406b41cbc8d10adac323c1dc99b4e08a6)
[(504)    API]  -  Starting COMPSs Runtime v2.2.rc1803 (build 20180426-1558.r3073191406b41cbc8d10adac323c1dc99b4e08a6)
[(505)    API]  -  Initializing components
[(539)    API]  -  Ready to process tasks
[(540)    API]  -  Registering CoreElement load(OBJECT_T,OBJECT_T,STRING_T,LONG_T,STRING_T,STRING_T,OBJECT_T)
[(540)    API]  -        - Implementation: load(OBJECT_T,OBJECT_T,STRING_T,LONG_T,STRING_T,STRING_T,OBJECT_T)es.bsc.compss.age
[(540)    API]  -        - Constraints    :
[(540)    API]  -        - Type           : METHOD
[(540)    API]  -        - ImplTypeArgs   :
[(540)    API]  -                Arg: es.bsc.compss.agent.loader.Loader
[(540)    API]  -                Arg: load
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/COMPSs/Runtime/adaptors/gat/master/compss-adaptors-gat-master.jar!/org/slf4j/impl/Stati
SLF4J: Found binding in [jar:file:/opt/COMPSs/Runtime/connectors/compss-defaultSSH-connector.jar!/org/slf4j/impl/StaticLoggerB
SLF4J: Found binding in [jar:file:/opt/COMPSs/Runtime/connectors/compss-defaultNoSSH-connector.jar!/org/slf4j/impl/StaticLogge
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Log4j:WARN No appenders could be found for logger (org.eclipse.jetty.util.log).
Log4j:WARN Please initialize the log4j system properly.
Log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Apr 26, 2018 3:59:39 PM org.glassfish.jersey.server.ApplicationHandler initialize
INFO: Initiating Jersey application, version Jersey: 2.7 2014-03-12 18:11:31...
```

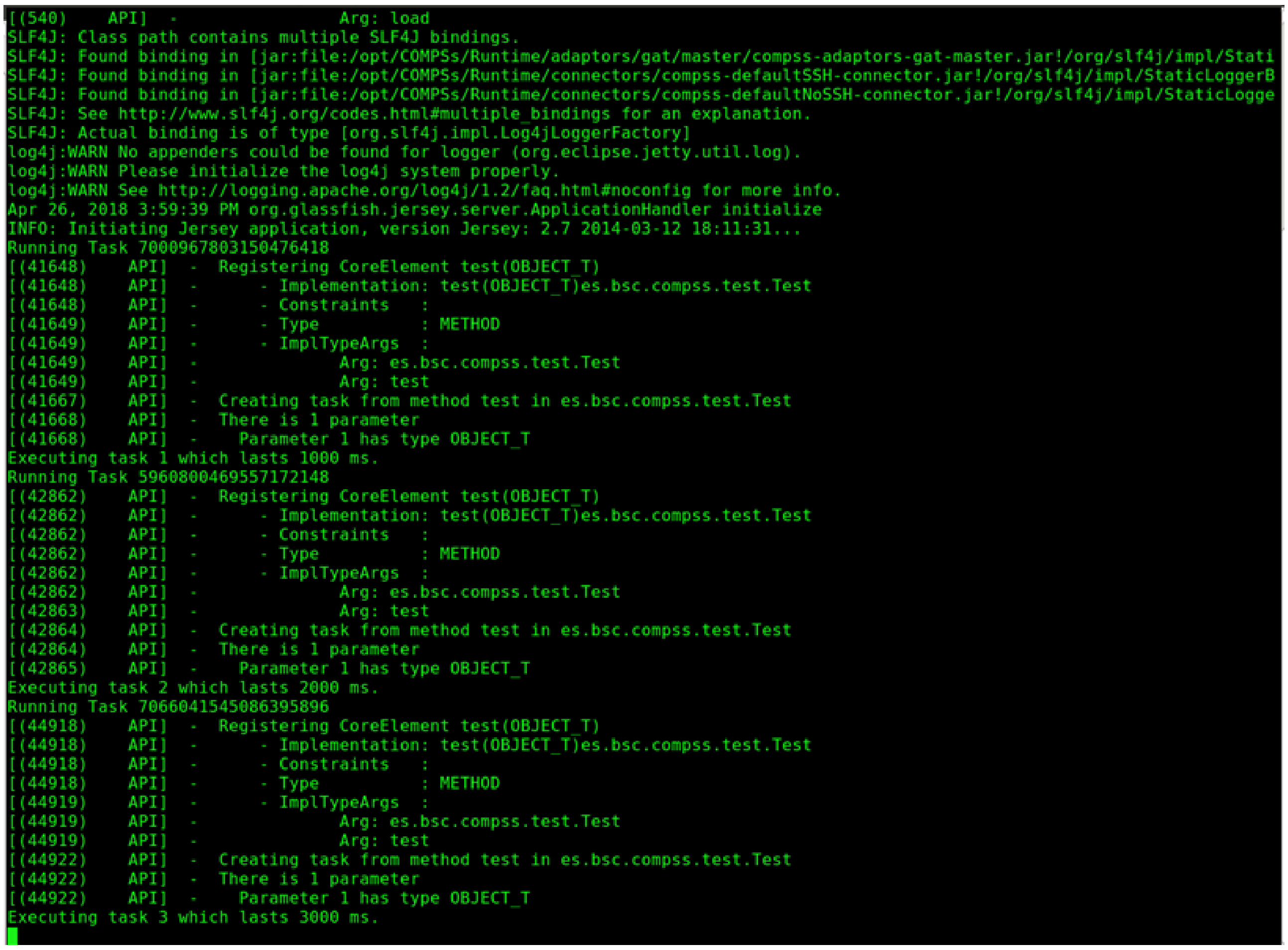
Figure 30 Deployment of DER

Figure 31 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.

Annex 1 contains the content of the Start Application request.



```
2.rc1803.jar -Dlog4j.configurationFile=/opt/COMPSs/Runtime/configuration/log/COMPSsMaster-log4j.debug -DMF2C_HOST=COMPSsWorker01 es.bsc.compss.agent.Agent 8080
ARNING: COMPSs Properties file is null. Setting default values
(505)    API]  -  Deploying COMPSs Runtime v2.2.rc1803 (build 20180426-1558.r3073191406b41cbc8d10adac323c1dc99b4e08a6)
(508)    API]  -  Starting COMPSs Runtime v2.2.rc1803 (build 20180426-1558.r3073191406b41cbc8d10adac323c1dc99b4e08a6)
(508)    API]  -  Initializing components
(539)    API]  -  Ready to process tasks
(540)    API]  -  Registering CoreElement load(OBJECT_T,OBJECT_T,STRING_T,LONG_T,STRING_T,STRING_T,OBJECT_T)
(540)    API]  -        - Implementation: load(OBJECT_T,OBJECT_T,STRING_T,LONG_T,STRING_T,STRING_T,OBJECT_T)es.bsc.compss.agent.loader.Loader
(540)    API]  -        - Constraints    :
(540)    API]  -        - Type           : METHOD
(540)    API]  -        - ImplTypeArgs   :
(540)    API]  -               Arg: es.bsc.compss.agent.loader.Loader
(540)    API]  -               Arg: load
LF4J: Class path contains multiple SLF4J bindings.
LF4J: Found binding in [jar:file:/opt/COMPSs/Runtime/adaptors/gat/master/compss-adaptors-gat-master.jar!/org/slf4j/impl/StaticLoggerBinder.class]
LF4J: Found binding in [jar:file:/opt/COMPSs/Runtime/connectors/compss-defaultSSH-connector.jar!/org/slf4j/impl/StaticLoggerBinder.class]
LF4J: Found binding in [jar:file:/opt/COMPSs/Runtime/connectors/compss-defaultNoSSH-connector.jar!/org/slf4j/impl/StaticLoggerBinder.class]
LF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
LF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
og4j:WARN No appenders could be found for logger (org.eclipse.jetty.util.log).
og4j:WARN Please initialize the log4j system properly.
og4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
pr 26, 2018 3:59:55 PM org.glassfish.jersey.server.ApplicationHandler initialize
NFO: Initiating Jersey application, version Jersey: 2.7 2014-03-12 18:11:31...
(24172)    API]  -  Creating task from method load in es.bsc.compss.agent.loader.Loader
(24172)    API]  -  There are 7 parameters
(24173)    API]  -    Parameter 1 has type OBJECT_T
(24174)    API]  -    Parameter 2 has type OBJECT_T
(24174)    API]  -    Parameter 3 has type STRING_T
(24175)    API]  -    Parameter 4 has type LONG_T
(24175)    API]  -    Parameter 5 has type STRING_T
(24177)    API]  -    Parameter 6 has type STRING_T
(24177)    API]  -    Parameter 7 has type OBJECT_T
(24398)    API]  -  Starting COMPSs Runtime v2.2.rc1803 (build 20180426-1558.r3073191406b41cbc8d10adac323c1dc99b4e08a6)
(24398)    API]  -  Initializing es.bsc.compss.test.TestItf
ariables set
ypes:  [class [Ljava.lang.String;]
arams: [[Ljava.lang.String;@35d9252c]
eceived a service invocation creating 3 tasks.
(24426)    API]  -  Creating task from method test in es.bsc.compss.test.Test
(24426)    API]  -  There is 1 parameter
(24427)    API]  -    Parameter 1 has type INT_T
(24427)    API]  -  Creating task from method test in es.bsc.compss.test.Test
(24427)    API]  -  There is 1 parameter
(24428)    API]  -    Parameter 1 has type INT_T
(24428)    API]  -  Creating task from method test in es.bsc.compss.test.Test
(24428)    API]  -  There is 1 parameter
(24428)    API]  -    Parameter 1 has type INT_T
(24429)    API]  -  No more tasks for app 665840759161115807
(31485)    API]  -  Getting Result Files 665840759161115807
xecution lasted 7316
```



```
[(540)    API]  -               Arg: load
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/COMPSs/Runtime/adaptors/gat/master/compss-adaptors-gat-master.jar!/org/slf4j/impl/Stati
SLF4J: Found binding in [jar:file:/opt/COMPSs/Runtime/connectors/compss-defaultSSH-connector.jar!/org/slf4j/impl/StaticLoggerB
SLF4J: Found binding in [jar:file:/opt/COMPSs/Runtime/connectors/compss-defaultNoSSH-connector.jar!/org/slf4j/impl/StaticLogge
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
log4j:WARN No appenders could be found for logger (org.eclipse.jetty.util.log).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Apr 26, 2018 3:59:39 PM org.glassfish.jersey.server.ApplicationHandler initialize
INFO: Initiating Jersey application, version Jersey: 2.7 2014-03-12 18:11:31...
Running Task 7000967803150476418
[(41648)    API]  -  Registering CoreElement test(OBJECT_T)
[(41648)    API]  -       - Implementation: test(OBJECT_T)es.bsc.compss.test.Test
[(41648)    API]  -       - Constraints    :
[(41649)    API]  -       - Type           : METHOD
[(41649)    API]  -       - ImplTypeArgs   :
[(41649)    API]  -               Arg: es.bsc.compss.test.Test
[(41649)    API]  -               Arg: test
[(41667)    API]  -  Creating task from method test in es.bsc.compss.test.Test
[(41668)    API]  -  There is 1 parameter
[(41668)    API]  -    Parameter 1 has type OBJECT_T
Executing task 1 which lasts 1000 ms.
Running Task 5960800469557172148
[(42862)    API]  -  Registering CoreElement test(OBJECT_T)
[(42862)    API]  -       - Implementation: test(OBJECT_T)es.bsc.compss.test.Test
[(42862)    API]  -       - Constraints    :
[(42862)    API]  -       - Type           : METHOD
[(42862)    API]  -       - ImplTypeArgs   :
[(42862)    API]  -               Arg: es.bsc.compss.test.Test
[(42863)    API]  -               Arg: test
[(42864)    API]  -  Creating task from method test in es.bsc.compss.test.Test
[(42864)    API]  -  There is 1 parameter
[(42865)    API]  -    Parameter 1 has type OBJECT_T
Executing task 2 which lasts 2000 ms.
Running Task 7066041545086395896
[(44918)    API]  -  Registering CoreElement test(OBJECT_T)
[(44918)    API]  -       - Implementation: test(OBJECT_T)es.bsc.compss.test.Test
[(44918)    API]  -       - Constraints    :
[(44918)    API]  -       - Type           : METHOD
[(44919)    API]  -       - ImplTypeArgs   :
[(44919)    API]  -               Arg: es.bsc.compss.test.Test
[(44919)    API]  -               Arg: test
[(44922)    API]  -  Creating task from method test in es.bsc.compss.test.Test
[(44922)    API]  -  There is 1 parameter
[(44922)    API]  -    Parameter 1 has type OBJECT_T
Executing task 3 which lasts 3000 ms.
```

Figure 31 Execution of an application

3.10.1 Integration tests

Test Case Number Version	**TC-DER Deployment**
Test Case Title	Validation of the deployment of the DER in the agent
Module tested	Distributed Execution Runtime
Related requirement(s)	
Target	Validate that the runtime is up and listening on the expected port when an application is deployed
Initial condition(s)	The mf2c/compss-test image available in the Docker Hub
Expected results	Master and worker instances, both deploying the COMPSs runtime, up and listening for execution requests
Status	Passed
Owner	Daniele Lezzi, Francesc Lordan (BSC)
Assigned	Daniele Lezzi, Francesc Lordan (BSC)
Steps	1. Instantiate a Docker container that plays the role of worker *docker run --rm -it --env MF2C_HOST=172.17.0.2 --env DEBUG=debug --name worker mf2c/compss-test:latest* 2. Instantiate a Docker container that plays the role of master. The container is listening on port 46100, it is important to redirect the port of the container host. *docker run --rm -it --env MF2C_HOST=172.17.0.3 -p46100:46100 --env DEBUG=debug --name master mf2c/compss-test:latest*
Feedback receiver	
Start date	M12
End date	M16
Passed?	Yes
Bug ID	-
Problems	
Required changes	
Comments	

Table 19. Integration tests for the Distributed Execution Runtime deployment workflow

3.11 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 behave 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 shown in the corresponding integration 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; the workflow is triggered by an earlier request from some mF2C component through CIMI.

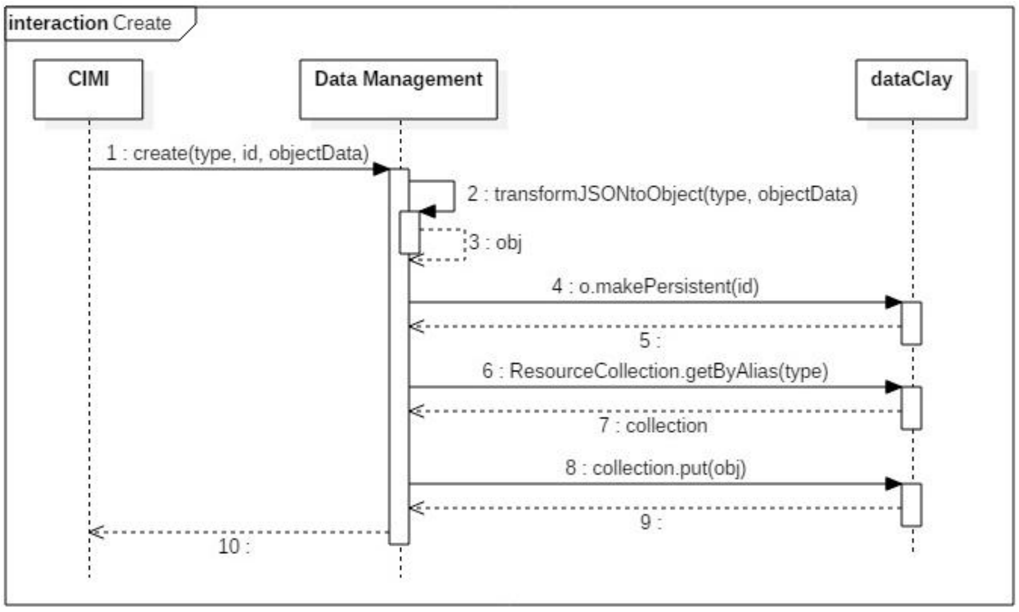


Figure 32 Data Management – Create

When a CREATE request is received from CIMI, the resource metadata 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’.

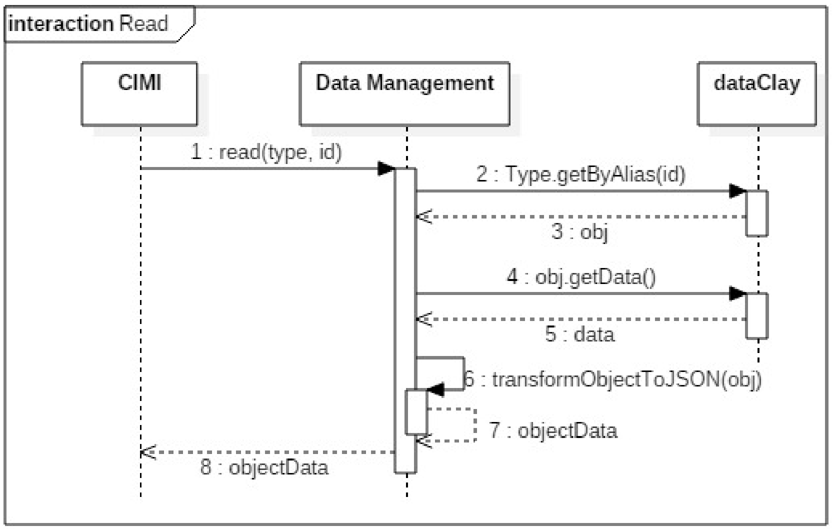


Figure 33 Data Management – Read

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 33), and serialized 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.

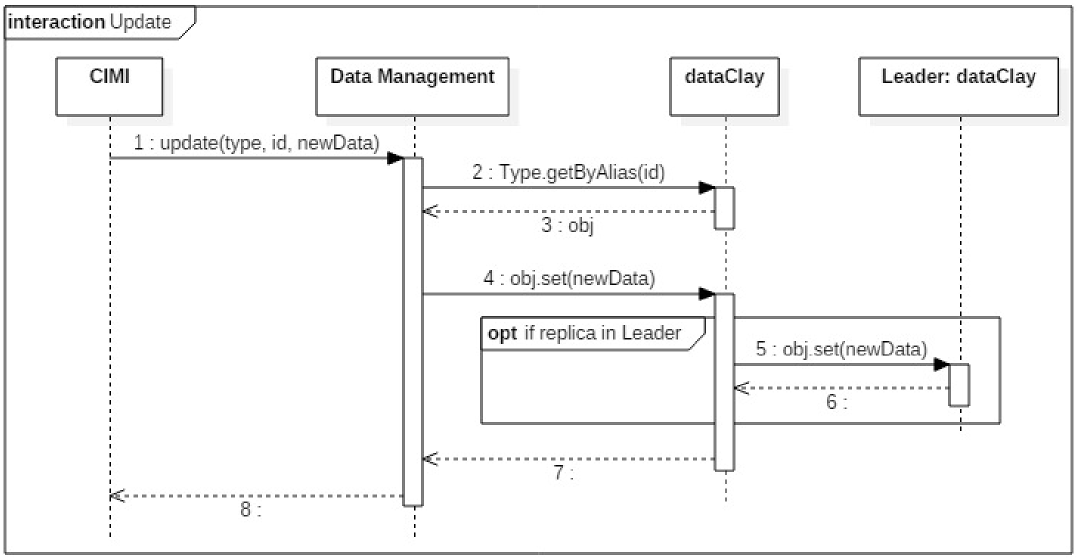


Figure 34 Data Management – Update

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.

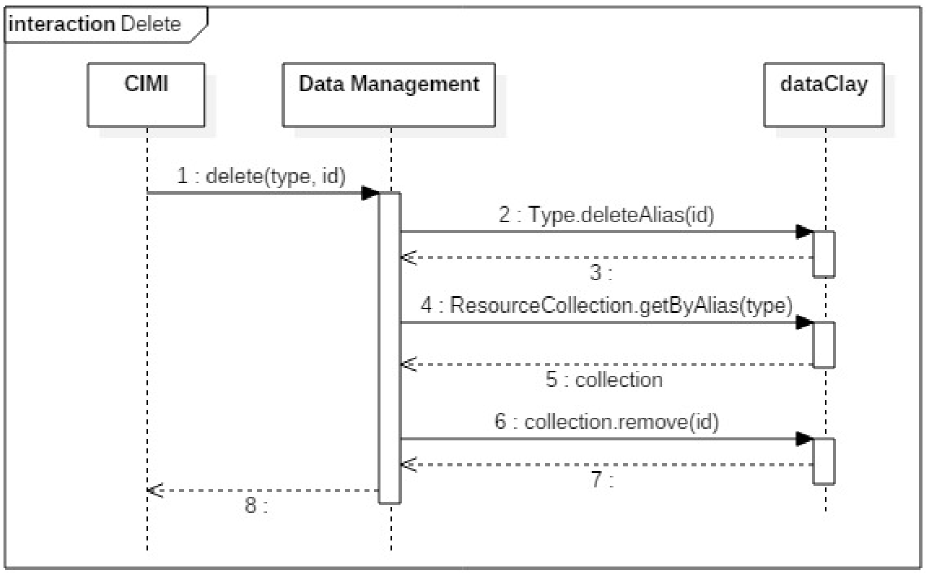


Figure 35 Data Management – Delete

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. Eventually, the garbage collector in dataClay will remove the object.

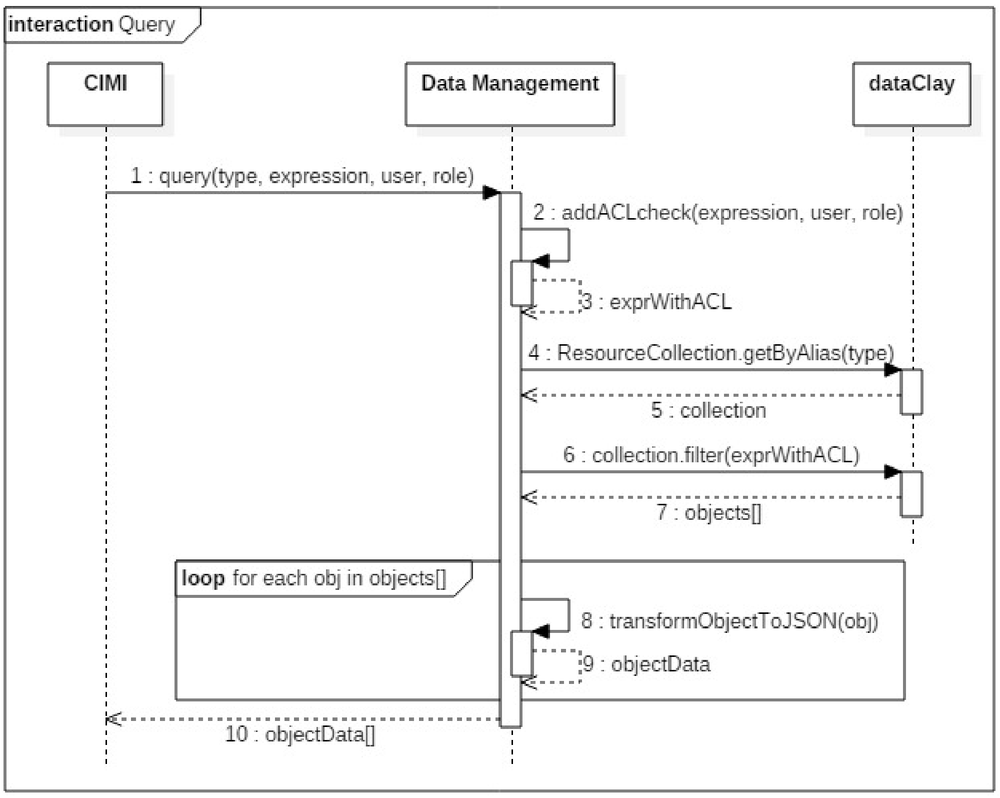
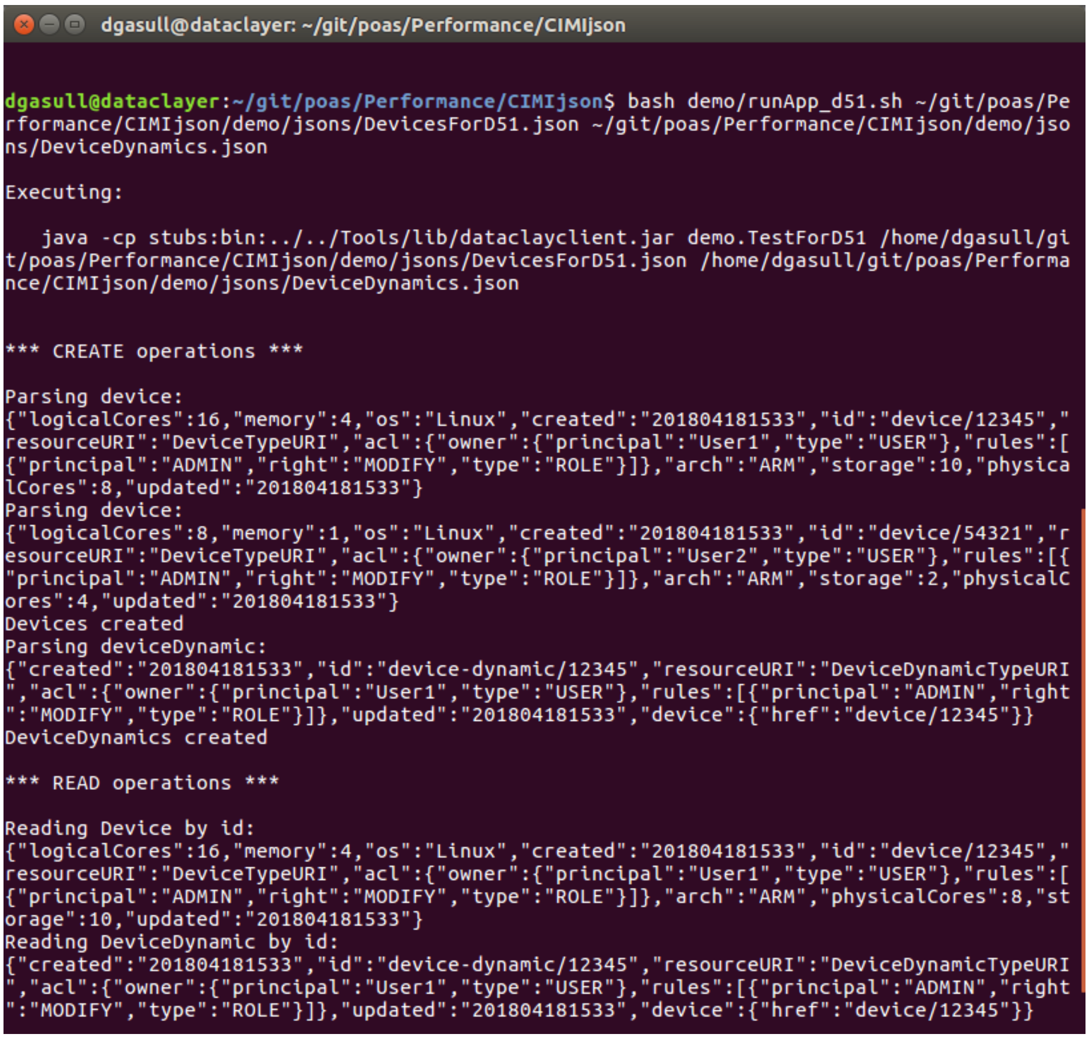

interaction Query
CIMI
Data Management
dataClay
1 : query(type, expression, user, role)
2 : addACLcheck(expression, user, role)
3 : exprWithACL
4 : ResourceCollection.getByAlias(type)
5 : collection
6 : collection.filter(exprWithACL)
7 : objects[]
loop for each obj in objects[]
8 : transformObjectToJSON(obj)
9 : objectData
10 : objectData[]

Figure 36 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 their 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.

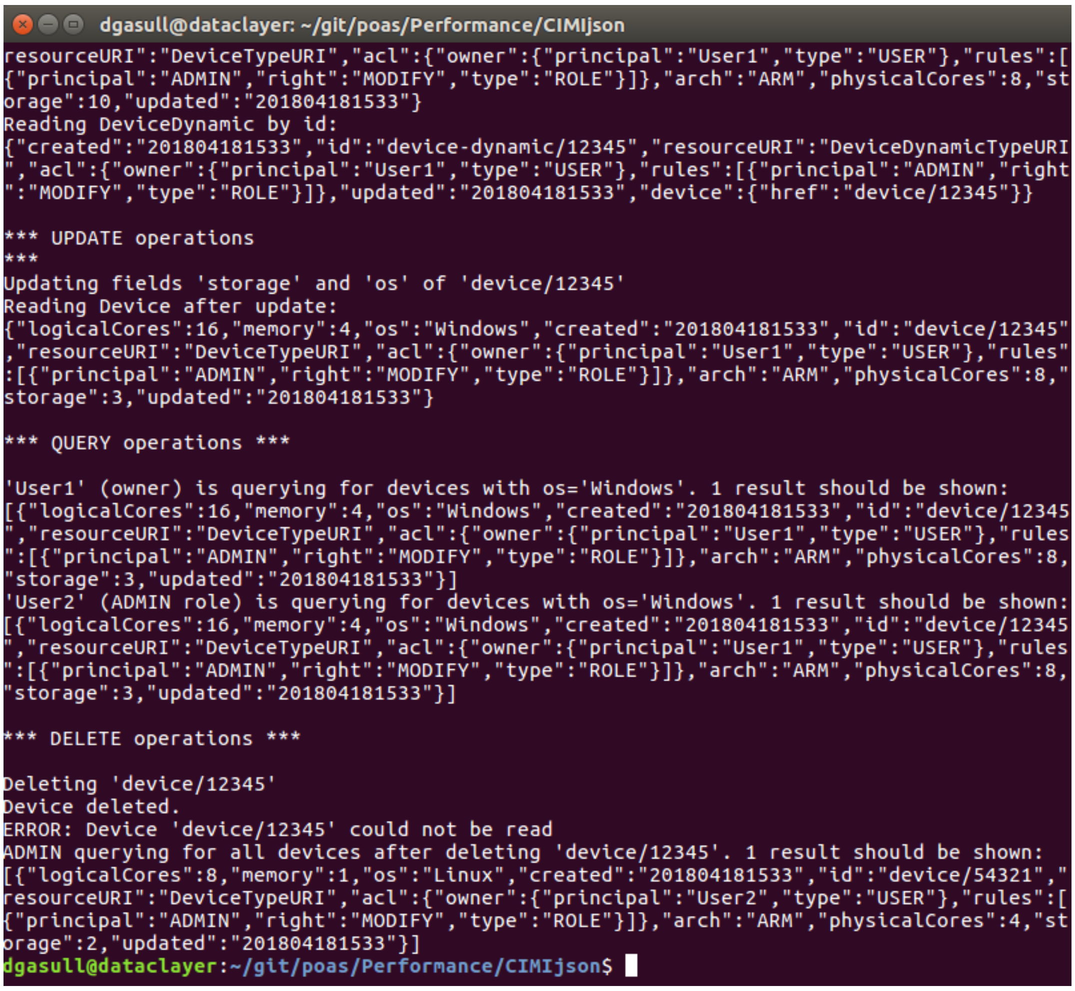
The application starts by creating a set of resources, reading their data from a set of JSON files provided as input. These files have the same format as the ones that would be received from CIMI.



```
dgasull@dataclayer: ~/git/poas/Performance/CIMIjson

dgasull@dataclayer:~/git/poas/Performance/CIMIjson$ bash demo/runApp_d51.sh ~/git/poas/Pe
rformance/CIMIjson/demo/jsons/DevicesForD51.json ~/git/poas/Performance/CIMIjson/demo/jso
ns/DeviceDynamics.json

Executing:

   java -cp stubs:bin:../../Tools/lib/dataclayclient.jar demo.TestForD51 /home/dgasull/gi
t/poas/Performance/CIMIjson/demo/jsons/DevicesForD51.json /home/dgasull/git/poas/Performa
nce/CIMIjson/demo/jsons/DeviceDynamics.json


*** CREATE operations ***

Parsing device:
{"logicalCores":16,"memory":4,"os":"Linux","created":"201804181533","id":"device/12345","
resourceURI":"DeviceTypeURI","acl":{"owner":{"principal":"User1","type":"USER"},"rules":[
{"principal":"ADMIN","right":"MODIFY","type":"ROLE"}]},"arch":"ARM","storage":10,"physica
lCores":8,"updated":"201804181533"}
Parsing device:
{"logicalCores":8,"memory":1,"os":"Linux","created":"201804181533","id":"device/54321","r
esourceURI":"DeviceTypeURI","acl":{"owner":{"principal":"User2","type":"USER"},"rules":[{
"principal":"ADMIN","right":"MODIFY","type":"ROLE"}]},"arch":"ARM","storage":2,"physicalC
ores":4,"updated":"201804181533"}
Devices created
Parsing deviceDynamic:
{"created":"201804181533","id":"device-dynamic/12345","resourceURI":"DeviceDynamicTypeURI
","acl":{"owner":{"principal":"User1","type":"USER"},"rules":[{"principal":"ADMIN","right
":"MODIFY","type":"ROLE"}]},"updated":"201804181533","device":{"href":"device/12345"}}
DeviceDynamics created

*** READ operations ***

Reading Device by id:
{"logicalCores":16,"memory":4,"os":"Linux","created":"201804181533","id":"device/12345","
resourceURI":"DeviceTypeURI","acl":{"owner":{"principal":"User1","type":"USER"},"rules":[
{"principal":"ADMIN","right":"MODIFY","type":"ROLE"}]},"arch":"ARM","physicalCores":8,"st
orage":10,"updated":"201804181533"}
Reading DeviceDynamic by id:
{"created":"201804181533","id":"device-dynamic/12345","resourceURI":"DeviceDynamicTypeURI
","acl":{"owner":{"principal":"User1","type":"USER"},"rules":[{"principal":"ADMIN","right
":"MODIFY","type":"ROLE"}]},"updated":"201804181533","device":{"href":"device/12345"}}
```

Figure 37 Data Management – Tests (part 1)

As can be seen in the output shown in Figure 37, 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 from the database objects by their identifier, are executed afterwards. We submit a request for Device “device/12345” and also for “device-dynamic/12345”, and return all the data they contain in the form of a JSON document. The test continues in the next figure, where we show the UPDATE, QUERY and DELETE operations. We UPDATE a couple of properties of “device/12345”, and read the device object using the previous operation to show that the values in “storage” and “os” have been correctly modified.



```
dgasull@dataclayer: ~/git/poas/Performance/CIMIjson
resourceURI":"DeviceTypeURI","acl":{"owner":{"principal":"User1","type":"USER"},"rules":[
{"principal":"ADMIN","right":"MODIFY","type":"ROLE"}]},"arch":"ARM","physicalCores":8,"st
orage":10,"updated":"201804181533"}
Reading DeviceDynamic by id:
{"created":"201804181533","id":"device-dynamic/12345","resourceURI":"DeviceDynamicTypeURI
","acl":{"owner":{"principal":"User1","type":"USER"},"rules":[{"principal":"ADMIN","right
":"MODIFY","type":"ROLE"}]},"updated":"201804181533","device":{"href":"device/12345"}}

*** UPDATE operations
***
Updating fields 'storage' and 'os' of 'device/12345'
Reading Device after update:
{"logicalCores":16,"memory":4,"os":"Windows","created":"201804181533","id":"device/12345"
,"resourceURI":"DeviceTypeURI","acl":{"owner":{"principal":"User1","type":"USER"},"rules"
:[{"principal":"ADMIN","right":"MODIFY","type":"ROLE"}]},"arch":"ARM","physicalCores":8,"
storage":3,"updated":"201804181533"}

*** QUERY operations ***

'User1' (owner) is querying for devices with os='Windows'. 1 result should be shown:
[{"logicalCores":16,"memory":4,"os":"Windows","created":"201804181533","id":"device/12345
","resourceURI":"DeviceTypeURI","acl":{"owner":{"principal":"User1","type":"USER"},"rules
":[{"principal":"ADMIN","right":"MODIFY","type":"ROLE"}]},"arch":"ARM","physicalCores":8,
"storage":3,"updated":"201804181533"}]
'User2' (ADMIN role) is querying for devices with os='Windows'. 1 result should be shown:
[{"logicalCores":16,"memory":4,"os":"Windows","created":"201804181533","id":"device/12345
","resourceURI":"DeviceTypeURI","acl":{"owner":{"principal":"User1","type":"USER"},"rules
":[{"principal":"ADMIN","right":"MODIFY","type":"ROLE"}]},"arch":"ARM","physicalCores":8,
"storage":3,"updated":"201804181533"}]

*** DELETE operations ***

Deleting 'device/12345'
Device deleted.
ERROR: Device 'device/12345' could not be read
ADMIN querying for all devices after deleting 'device/12345'. 1 result should be shown:
[{"logicalCores":8,"memory":1,"os":"Linux","created":"201804181533","id":"device/54321","
resourceURI":"DeviceTypeURI","acl":{"owner":{"principal":"User2","type":"USER"},"rules":[
{"principal":"ADMIN","right":"MODIFY","type":"ROLE"}]},"arch":"ARM","physicalCores":4,"st
orage":2,"updated":"201804181533"}]
dgasull@dataclayer:~/git/poas/Performance/CIMIjson$ 
```

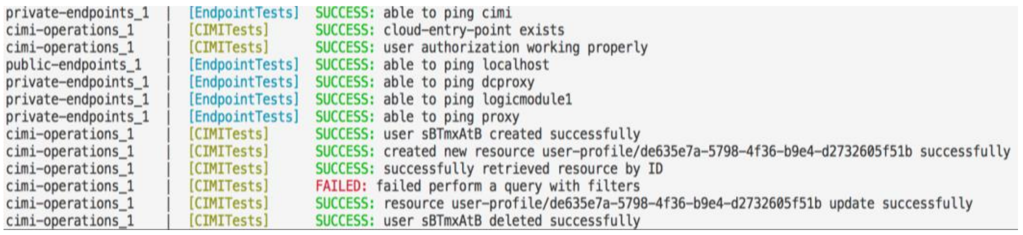
Figure 38 Data Management – Tests (part 2)

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 in the ACL 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.

3.11.1 Integration tests

Test Case Number Version	**TC-Data Management**
Test Case Title	Validation of the Data Management workflow
Modules tested	CIMI, DataClay
Related requirement(s)	CIMI and DataClay Docker containers running on a device (i.e. laptop) (https://github.com/mF2C/cimi/blob/master/_demo/README.md)
Target	In this scenario, an unregistered user will use CIMI to self-register into the deployed infrastructure using the CLI. As a result, the user shall receive a confirmation email (which also happens for the registration workflow shown in Figure 2), login and make sure different system resources can be managed through the interface. This process is manual as it implies verifying the user's email inbox in order to validate the user and continue with the test. On a more automated fashion, CRUD operations and searching capabilities will also be tested to validate the readiness of the CIMI-DataClay binding. These two components are the responsible for all the interactions between users, components and the database.
Initial condition(s)	1. CIMI and DataClay deployed from scratch, through a single compose file. 2. CIMI's interface must be published into the tester's host machine, so it can be reached at *localhost.* 3. The tester's device is equipped with *curl* (or similar). 4. CIMI must be configured to accept anonymous requests with escalated privileges, in order to perform ADMIN operations from within an automated script.
Expected results	The user can self-register with the exposed interface, and follow up the registration by validating his/her newly created username in the database. After login, the user can also perform CRUD operation on all the user-writable system resources. Using the automated integration test, we should observe a set of service endpoints and functionality tests being done in parallel by separate micro-services. The final result should be a table with one row per test, having all rows marked as successful.
Status	Finished
Owner	Cristovao Cordeiro (SixSq)
Assigned	Cristovao Cordeiro (SixSq)
Steps	1. Make sure CIMI (with DataClay) has been deployed locally and it is running 2. Issue a POST request to self-register a new user 3. Check the email inbox for a validation email 4. Follow the validation link

	5. Login, by creating a new session, using the user credentials from step 2. 6. Run the automated integration test (instruction at https://github.com/mF2C/cimi/blob/master/test-integration/README.md). This test will: a. Ping CIMI, DataClay, the DataClay Proxy and the CIMI reverse proxy b. Ping *localhost* to make sure the API has been published correctly c. Verify if CIMI's special resource *cloud-entry-point* has been created and is available d. Verify that basic ACLs and authorization handling are in place e. Create a test user f. Create a test resource g. Get a specific resource by ID h. Perform a query with filters i. Update an existing resource j. Delete a resource
Feedback receiver	Cristovao Cordeiro (SixSq), Anna Queralt (BSC)
Start date	M10
End date	M16

Passed?	The manual steps 1, 2, 3 and 4 from above passed successfully. Step 5 however failed due to a missing DataClay model for the respective CIMI *session-template*:
	`POST https://localhost/api/session @regularUser.json`
	`{`
	`  "status" : 400,`
	`  "message" : "requested href not found: session-template/internal"`
	`}`
	The automated tests from step 6 onwards returned the following results:
	
	`private-endpoints_1  | [EndpointTests] SUCCESS: able to ping cimi`
	`cimi-operations_1    | [CIMITests]     SUCCESS: cloud-entry-point exists`
	`cimi-operations_1    | [CIMITests]     SUCCESS: user authorization working properly`
	`public-endpoints_1   | [EndpointTests] SUCCESS: able to ping localhost`
	`private-endpoints_1  | [EndpointTests] SUCCESS: able to ping dcproxy`
	`private-endpoints_1  | [EndpointTests] SUCCESS: able to ping logicmodule1`
	`private-endpoints_1  | [EndpointTests] SUCCESS: able to ping proxy`
	`cimi-operations_1    | [CIMITests]     SUCCESS: user sBTmxAtB created successfully`
	`cimi-operations_1    | [CIMITests]     SUCCESS: created new resource user-profile/de635e7a-5798-4f36-b9e4-d2732605f51b successfully`
	`cimi-operations_1    | [CIMITests]     SUCCESS: successfully retrieved resource by ID`
	`cimi-operations_1    | [CIMITests]     FAILED: failed perform a query with filters`
	`cimi-operations_1    | [CIMITests]     SUCCESS: resource user-profile/de635e7a-5798-4f36-b9e4-d2732605f51b update successfully`
	`cimi-operations_1    | [CIMITests]     SUCCESS: user sBTmxAtB deleted successfully`
	which indicates all tests passed except step 6-h, where it was not possible to perform a CIMI search using filters.
	
	The outcome is that 15 tests passed out of **17**.

Bug ID	https://github.com/mF2C/dataClay/issues/12 https://github.com/mF2C/dataClay/issues/13
Problems	The session-template type "*internal*" is missing from the DataClay Java models. This should be created to match the existing CIMI session schemas.

	The second failed test is related with the API querying capabilities, which are not being processed. The source of the problem is yet to be identified.
Required changes	-
Comments	This integration test should be executed whenever there will be changes to the respective modules, as it is expected that the observed issues will disappeared throughout the upcoming releases.

Table 20. Integration tests for the data management operations and components availability

3.12 Analytics

The implementation of Service Analysis and the characterization of services have not changed much. For a previously deployed service, a subgraph (of nodes/edges representing the service, virtualized infrastructure and physical hosts) 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.

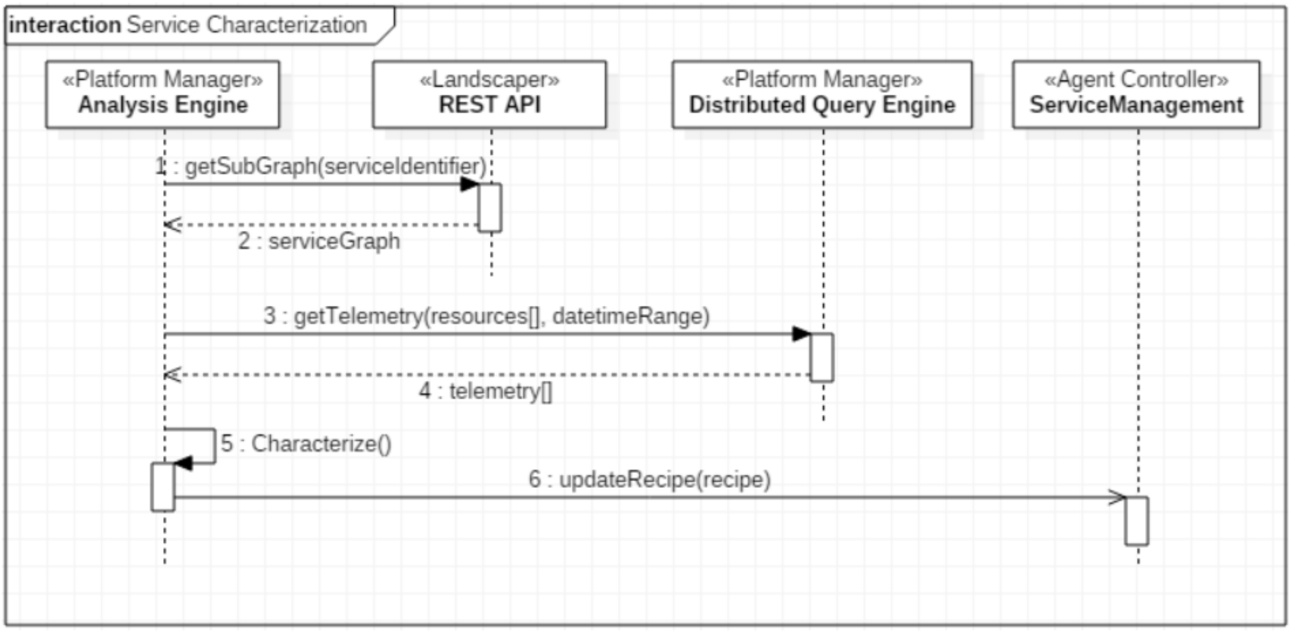

interaction Service Characterization
«Platform Manager» Analysis Engine
«Landscaper» REST API
«Platform Manager» Distributed Query Engine
«Agent Controller» ServiceManagement
1 : getSubGraph(serviceIdentifier)
2 : serviceGraph
3 : getTelemetry(resources[], datetimeRange)
4 : telemetry[]
5 : Characterize()
6 : updateRecipe(recipe)

Figure 39 Service characterization

The real-time Service Performance analysis follows the same flow as Figure 39 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. Subsequently, in the future, the Lifecycle Manager should be notified that a newer version of the recipe is available if it wants to issue service replacement.

3.12.1 Integration tests

Test Case Number Version	**TC-Analytics – Service characterization**
Test Case Title	Perform a characterization of a service and update the service description (recipe) in the catalogue
Module tested	Analytics
Related requirement(s)	

Target	
Initial condition(s)	Analytics, Telemetry and Landscaper containers must be started on the device
Expected results	- A characterization of the service is generated for resources: CPU, RAM, Disk, Network - The Service's model (recipe) is updated in the Service Catalogue
Status	Completed
Owner	Alec Leckey (Intel)
Assigned	Alec Leckey (Intel)
Steps	Call the /analyse method of the REST API, passing the service_instance to be analysed
Feedback receiver	None
Start date	1 June
End date	1 June
Passed?	FAIL
Bug ID	
Problems	If the service instance supplied is a remotely running service, then it won't be found in the landscape (due to the way landscaper is adding remote services/containers). Analysis of local service instances generate performance characterization. Service object model is updated
Required changes	Fix to landscaper to add remote services/containers
Comments	

Table 21. Integration tests for the service characterization from the analytics component

4. PoC Description

This section introduces the first prototype delivered within the mF2C project, in terms of a Proof of Concept (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 (for example, categories to be included may be: IoT, data, smart cities…). 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.
- 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 be very basic
- Clustering of nodes policy set manually at bootstrap
- All services fill a single container each
- A "recipe", defined as the recommended devices on which a service can run, consists of (cpu, memory, disk and network)
- 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 SLA management block detects violation.
- The dynamic resources information is obtained from COMPSs notifying the Lifecycle Manager, when a task is done and the latter reporting to dataClay through CIMI. 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 recommended through the Lifecycle Manager
- 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
- During app execution it is assumed that leader does not fail, and if it does, no backup is provided.
- 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. 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 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 into 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 broadcasted 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 fog-to-cloud environment.

4.1.1.3 Leader Failure

This functionality refers to the handover process when a leader goes down or becomes 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 checks 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 IoT devices and agents. 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.

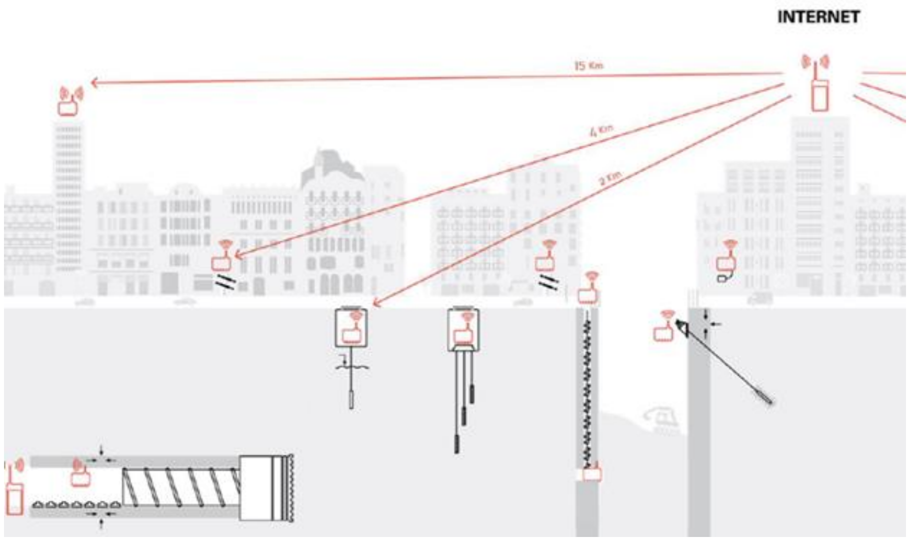


Figure 40 LoadSensing for Infrastructure monitoring

The use case consists of analyzing sensor measurements. The tilt-meter sends inclination information to the Gateway, which relays this information to the cloud. This data is both analyzed by the agent 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 for service described in a) 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 the second part (b) of the emergency service. 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.

The data flow for services described in b), launching the emergency service when the emergency message is triggered from part (a), is the following:

- An ambulance is requested.
- A fire engine is requested.
- The best paths are computed, from the location of the ambulance and the fire engine to the building where the inclination is detected.
- Traffic lights are changed from red to green in the computed paths.
- The ambulance and the fire engine move automatically to the building through the computed paths.

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 for its 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 the use case are grouped into five major user functionalities of the planned end product, where only two will be demonstrated for IT-1:

- Continuous Boat Monitoring
- Sensor Control

Both boat monitoring and sensor control are 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 one considers 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).

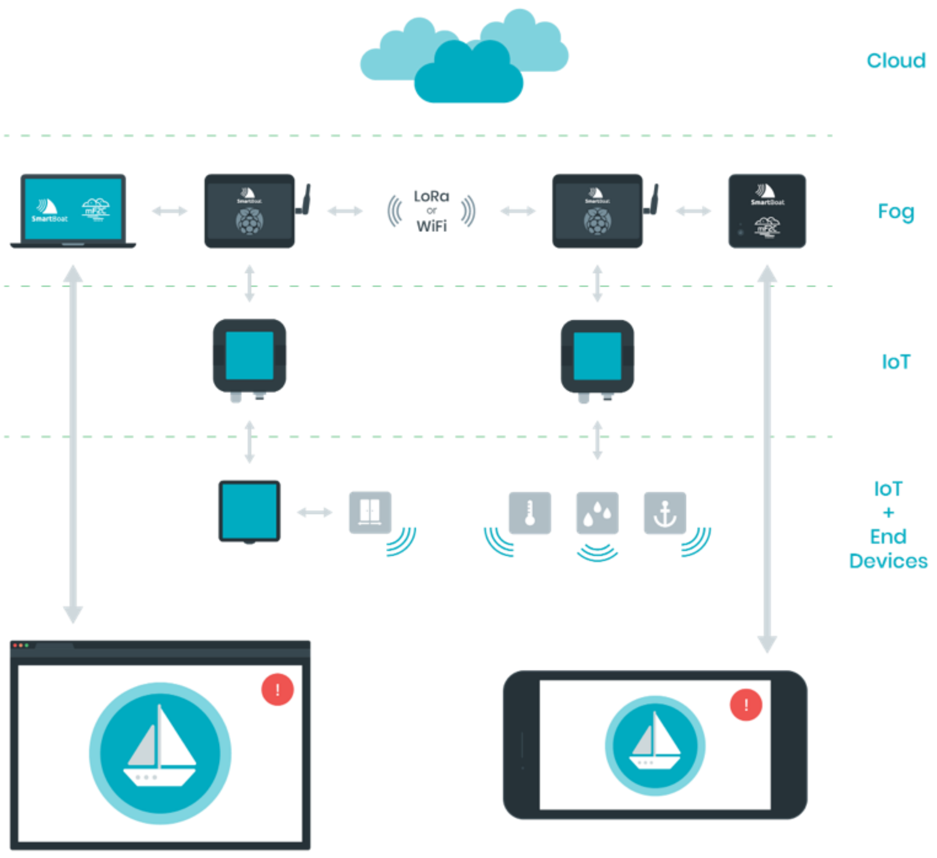


Figure 41 Use case 2 system architecture

Figure 41 presents Smart Boat deployment including multiple layers in the mF2C scheme, depending on the number of 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 the final deployment, most of the fog processing will be done by a Raspberry Pi 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 Raspberry Pi that manages the fog fleet and sensor data collection and control. The demo will have two Raspberry Pi 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 Raspberry Pi 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 will be used 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 Raspberry Pi, 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 Raspberry Pi.
- 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 sent to a second Raspberry Pi from the application in the laptop that uses the LoRa component of the first Raspberry Pi via gRPC. The second Raspberry Pi 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 of multiple boats in the fleet in the actual environment) and reports an anomaly if the difference is too big. The LoRa module will be finalized in IT2, currently it is in a proof of concept stage.

4.2.3 Use Case 3

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 Raspberry Pi, 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 (leader), 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 Raspberry Pi3 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 generators.

The mF2C agent runs in all Cloud and Fog elements; this is not currently supported on Raspberry Pi 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.

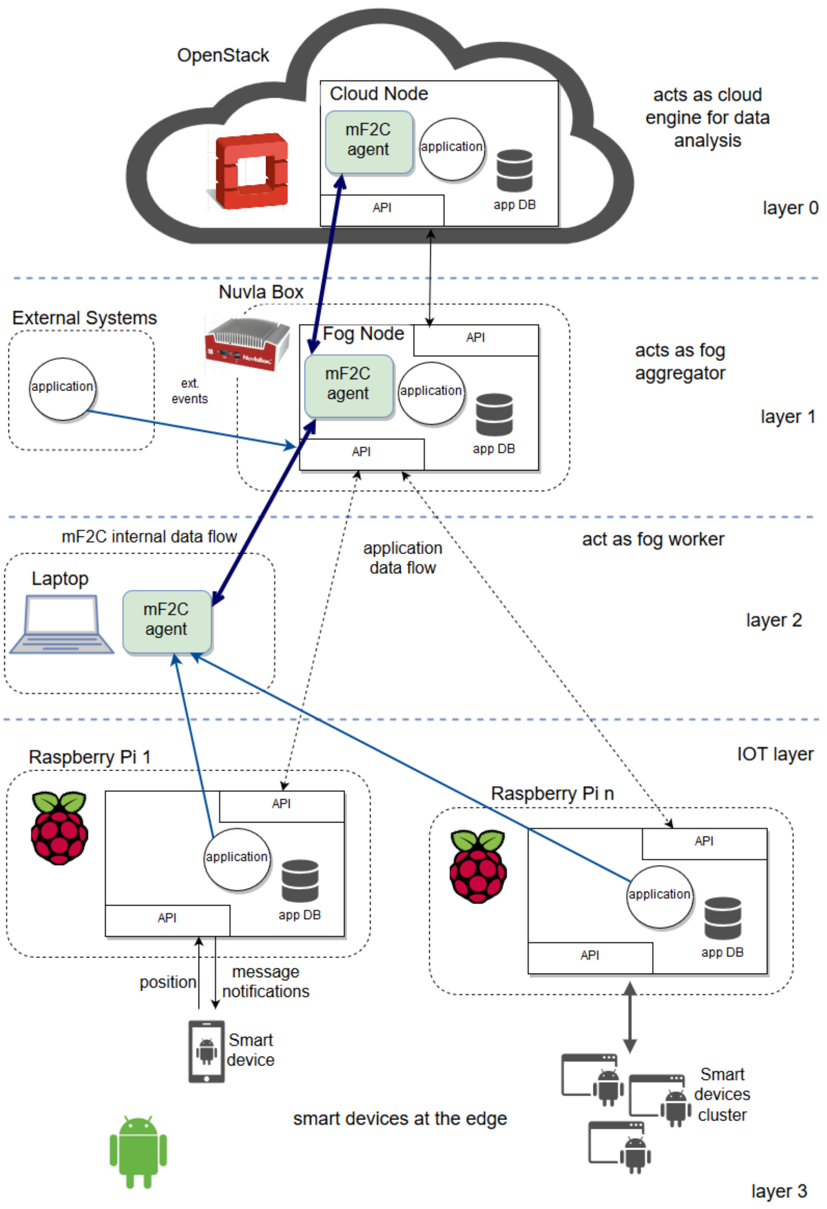


Figure 42 Use Case 3 system architecture

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)

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 use case 1 (emergency management). This service is developed in a Smart City scenario with mF2C capacity (Figure 43), 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.

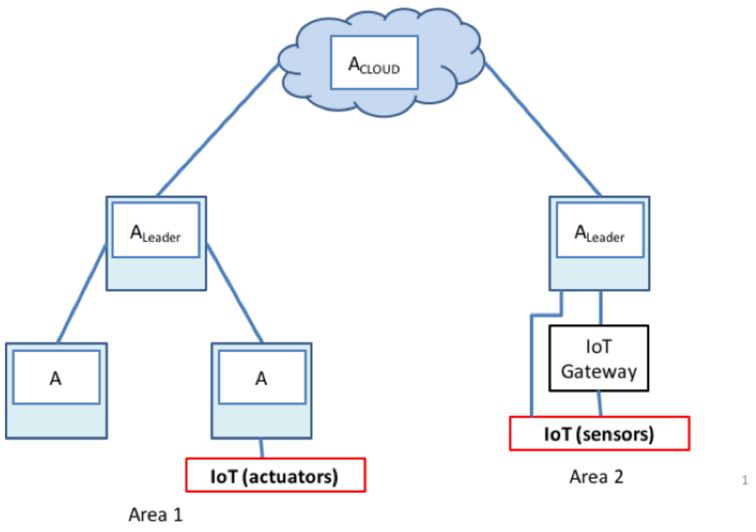


Figure 43 Topology of use case 1

The set of agents that make up the topology has a set of connected IoT devices that enable a response to an emergency service. The emergency service is emulated in the testbed shown in Figure 44, where 3 buildings are shown (red, pink and blue boxes):

- The sensorized building is to be monitored (red one in Figure 44).
- Hospital with the ambulance inside (pink one in Figure 44).
- Fire station with the fire engine inside (blue one in Figure 44).

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.

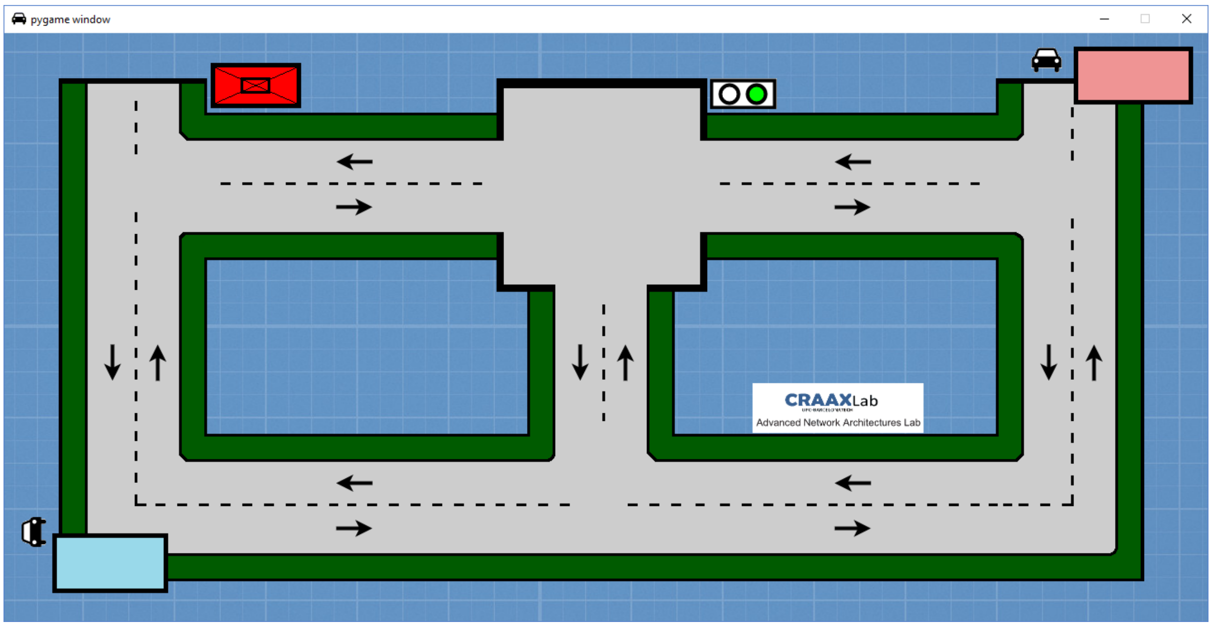


Figure 44 CRAAX Testbed

The two areas of action, shown in Figure 44, represent two separated zones of a Smart City, as shown in Figure 45.

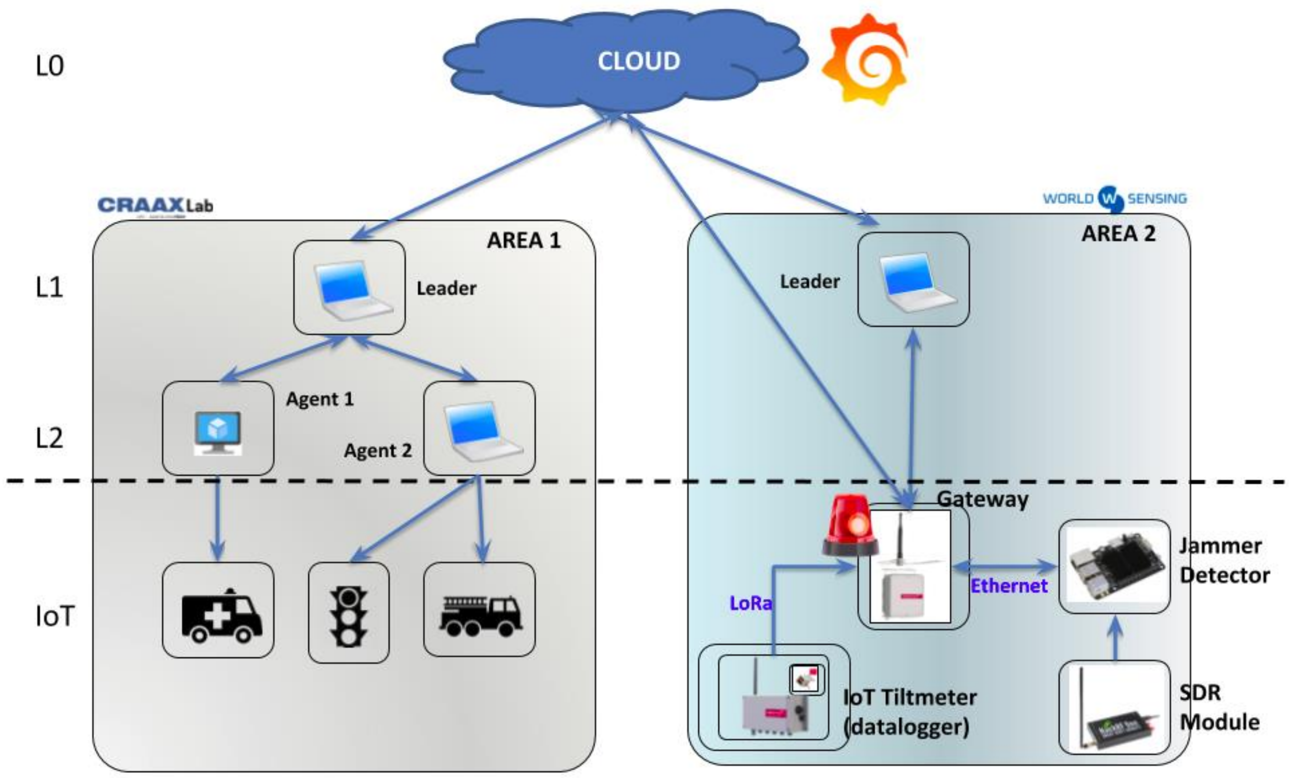


Figure 45 Infrastructure of use case 1

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 (PC)
- Agent 1 (VM)→ connected to an ambulance and to a fire engine.
- Agent 2 (VM)→ to be used as a computation element.

The emergency event detection itself is located on the Cloud and also in 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 IoT devices connected to the leader through a gateway,

- Cloud: is used to receive and store the data coming from the IoT sensors, coming from the Gateway. It also provides real time visualization tools. This software is called the **Monitoring Software** and is in charge of receiving sensor data and controlling the alert cases. In case of an emergency situation, it will contact the Gateway to start the physical alarms and will start the emergency actions.
- Leader (laptop) will be connected to the Gateway and will run the **Monitoring Software** as a backup in order to provide more reliability and improve QoS as well as latency when treating emergencies.
- The Gateway is connected to the following IoT devices:

- through wire (Ethernet) to the Jammer detector.
- through LoRa to the dataloggers, receiving inclination measurements wirelessly..
- to a temperature sensor. In IT-2 we will use this sensor to demonstrate the detection of 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 (IoT)
- LoadSensing Gateway (IoT)

Jammer Detector's group:

- Jammer Detector (IoT)

WOS Leader:

- Laptop - Leader Agent (L1 Fog)

Cloud Server:

- Cloud Virtual Machine – Cloud Agent (L0 Cloud)

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 are intentionally 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 a 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 Raspberry Pi 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. 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 of this deliverable.

About 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” if a component is already considered secure and suitable for use, potentially even in a hostile environment.

The expected outcome of the security test is therefore:

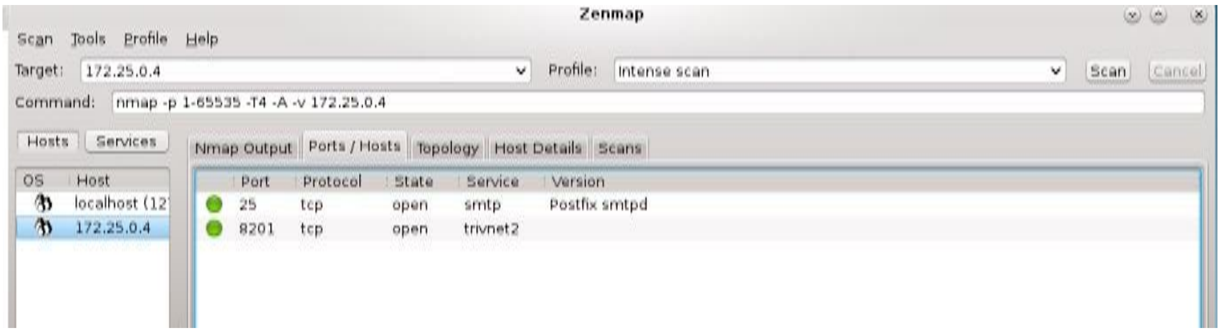
- Discovery of known security vulnerabilities, because:
  - The protection is out of scope for IT-1, or
  - The security feature has not been implemented yet.
- Potential discovery of unknown or unexpected vulnerabilities.

- 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 46 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.



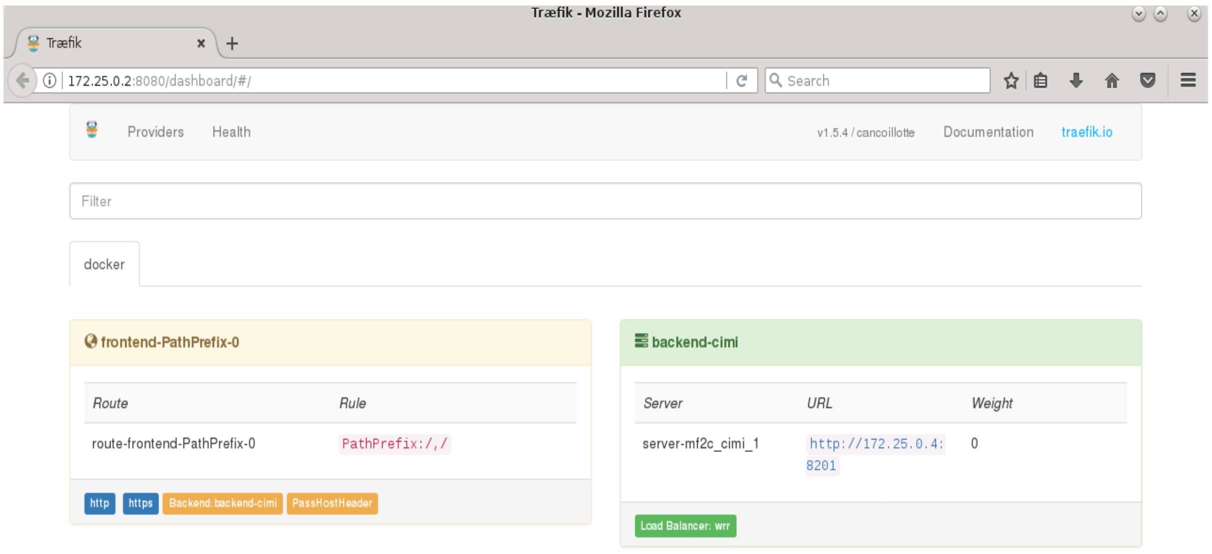
Target: 172.25.0.4
Profile: Intense scan
Command: nmap -p 1-65535 -T4 -A -v 172.25.0.4

OS	Host
	localhost (12
	172.25.0.4

	Port	Protocol	State	Service	Version
●	25	tcp	open	smtp	Postfix smtpd
●	8201	tcp	open	trivnet2	

Figure 46 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.



frontend-PathPrefix-0

Route	Rule
route-frontend-PathPrefix-0	PathPrefix:/,/

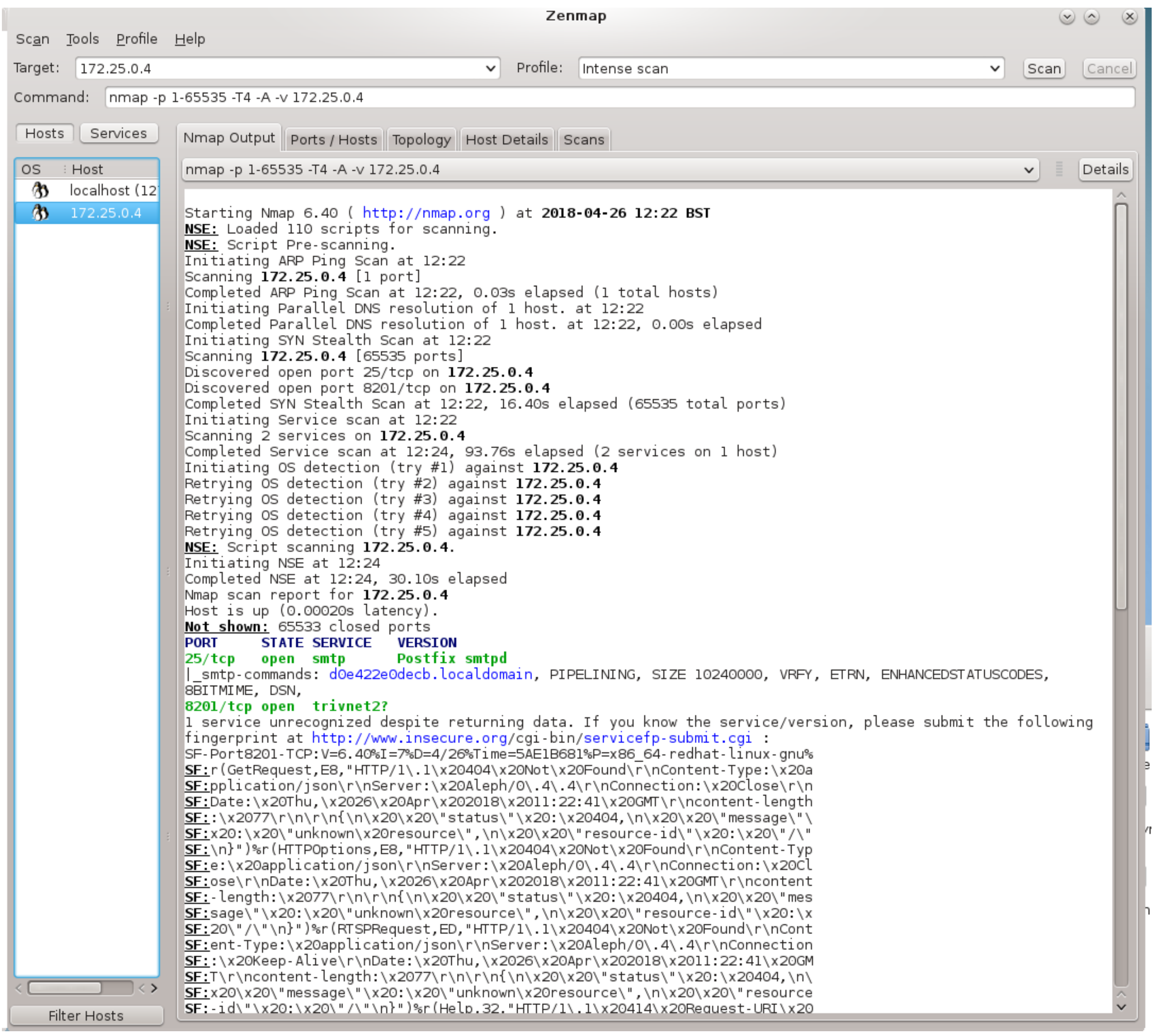
http https Backend:backend-cimi PassHostHeader

backend-cimi

Server	URL	Weight
server-mf2c_cimi_1	http://172.25.0.4:8201	0

Load Balancer: wrr

Figure 47 Traefik monitoring page exposed



```
Starting Nmap 6.40 ( http://nmap.org ) at 2018-04-26 12:22 BST
NSE: Loaded 110 scripts for scanning.
NSE: Script Pre-scanning.
Initiating ARP Ping Scan at 12:22
Scanning 172.25.0.4 [1 port]
Completed ARP Ping Scan at 12:22, 0.03s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 12:22
Completed Parallel DNS resolution of 1 host. at 12:22, 0.00s elapsed
Initiating SYN Stealth Scan at 12:22
Scanning 172.25.0.4 [65535 ports]
Discovered open port 25/tcp on 172.25.0.4
Discovered open port 8201/tcp on 172.25.0.4
Completed SYN Stealth Scan at 12:22, 16.40s elapsed (65535 total ports)
Initiating Service scan at 12:22
Scanning 2 services on 172.25.0.4
Completed Service scan at 12:24, 93.76s elapsed (2 services on 1 host)
Initiating OS detection (try #1) against 172.25.0.4
Retrying OS detection (try #2) against 172.25.0.4
Retrying OS detection (try #3) against 172.25.0.4
Retrying OS detection (try #4) against 172.25.0.4
Retrying OS detection (try #5) against 172.25.0.4
NSE: Script scanning 172.25.0.4.
Initiating NSE at 12:24
Completed NSE at 12:24, 30.10s elapsed
Nmap scan report for 172.25.0.4
Host is up (0.00020s latency).
Not shown: 65533 closed ports
PORT     STATE SERVICE   VERSION
25/tcp   open  smtp      Postfix smtpd
|_smtp-commands: d0e422e0decb.localdomain, PIPELINING, SIZE 10240000, VRFY, ETRN, ENHANCEDSTATUSCODES,
8BITMIME, DSN,
8201/tcp open  trivnet2?
1 service unrecognized despite returning data. If you know the service/version, please submit the following
fingerprint at http://www.insecure.org/cgi-bin/servicefp-submit.cgi :
SF-Port8201-TCP:V=6.40%I=7%D=4/26%Time=5AE1B681%P=x86_64-redhat-linux-gnu%
SF:r(GetRequest,E8,"HTTP/1\.1\x20404\x20Not\x20Found\r\nContent-Type:\x20a
SF:pplication/json\r\nServer:\x20Aleph/0\.4\.4\r\nConnection:\x20Close\r\n
SF:Date:\x20Thu,\x2026\x20Apr\x202018\x2011:22:41\x20GMT\r\ncontent-length
SF::\x2077\r\n\r\n{\n\x20\x20\"status\"\x20:\x20404,\n\x20\x20\"message\"\
SF:x20:\x20\"unknown\x20resource\",\n\x20\x20\"resource-id\"\x20:\x20\"/\"
SF:\n}")%r(HTTPOptions,E8,"HTTP/1\.1\x20404\x20Not\x20Found\r\nContent-Typ
SF:e:\x20application/json\r\nServer:\x20Aleph/0\.4\.4\r\nConnection:\x20Cl
SF:ose\r\nDate:\x20Thu,\x2026\x20Apr\x202018\x2011:22:41\x20GMT\r\ncontent
SF:-length:\x2077\r\n\r\n{\n\x20\x20\"status\"\x20:\x20404,\n\x20\x20\"mes
SF:sage\"\x20:\x20\"unknown\x20resource\",\n\x20\x20\"resource-id\"\x20:\x
SF:20\"/\"\n}")%r(RTSPRequest,ED,"HTTP/1\.1\x20404\x20Not\x20Found\r\nCont
SF:ent-Type:\x20application/json\r\nServer:\x20Aleph/0\.4\.4\r\nConnection
SF::\x20Keep-Alive\r\nDate:\x20Thu,\x2026\x20Apr\x202018\x2011:22:41\x20GM
SF:T\r\ncontent-length:\x2077\r\n\r\n{\n\x20\x20\"status\"\x20:\x20404,\n\
SF:x20\x20\"message\"\x20:\x20\"unknown\x20resource\",\n\x20\x20\"resource
SF:-id\"\x20:\x20\"/\"\n}")%r(Help,32,"HTTP/1\.1\x20414\x20Request-URI\x20
```

Figure 48 Example 2 of use of nmap

Figure 47 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] 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 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 or have already been deployed in the most recent container images. 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 any handling personal data should be done with extra care.

5.4 Testers Task Force

Testing mF2C components has been considered a primary task. Testing helps by increasing the quality of software components and their 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 22. Test case table

So, once each mF2C component was released, the corresponding content was downloaded 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. The tested components have been chosen either due to availability at the time of testing and/or relevance to the use cases.

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 must 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 must 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 must 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 executed 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 must 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 must 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 executed in correct way just for the installation of "realpath" command for un external repository but failed for another error	passed

DataClay	v.1	30/jan/2018	Paolo Cocco	Installation test on the dataClay must verify the correct installation for the application. Centos7 with Docker, java 8, python installed, have java class People, Person and Hello People described in the manual	passed
DataClay	v.1	30/jan/2018	Paolo Cocco	Installation test on the dataClay must 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 must 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 must verify the correct installation and dockerizing the application. Ubuntu 16.04 vanilla with Docker installed, and java8	passed

Table 23. 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 to 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.Raspberry Pi.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.

Annex 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>
```

Annex 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 annex.

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 in the browser.
web_spider	Crawl the web application.

Annex 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 Annex 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.

Annex 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 Annex 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

Annex 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 Annex 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 Annex 4 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

Annex 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.

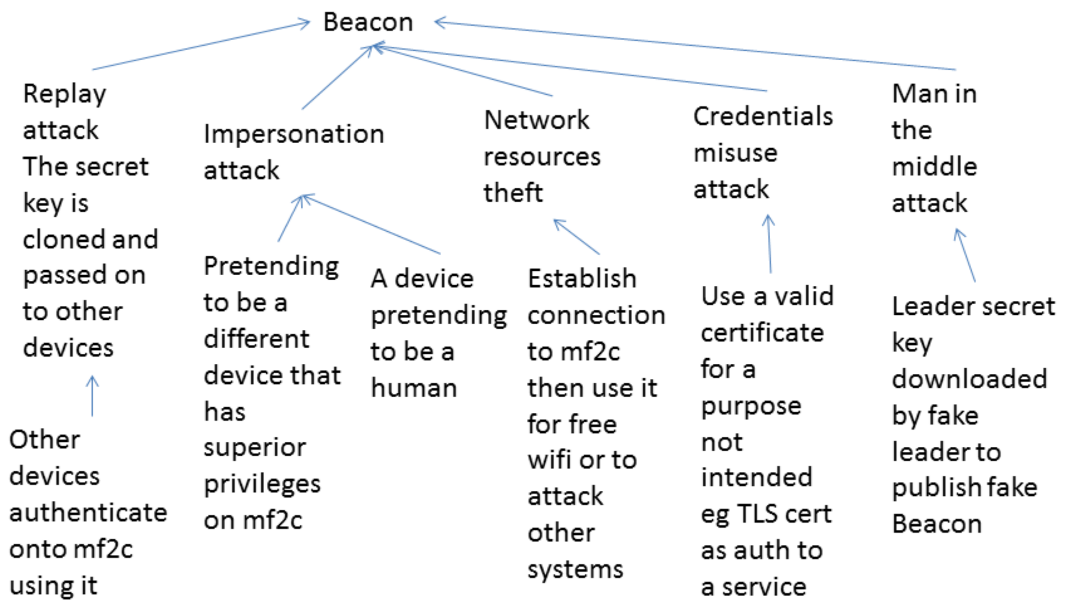
We shall not produce a complete model, only a representative model, because of the amount of work involved. Below is shown a very small part of the threat model so far produced.

[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


Beacon
Replay attack The secret key is cloned and passed on to other devices
Other devices authenticate onto mf2c using it
Impersonation attack
Pretending to be a different device that has superior privileges on mf2c
A device pretending to be a human
Network resources theft
Establish connection to mf2c then use it for free wifi or to attack other systems
Credentials misuse attack
Use a valid certificate for a purpose not intended eg TLS cert as auth to a service
Man in the middle attack
Leader secret key downloaded by fake leader to publish fake Beacon

Annex 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.

Annex 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.

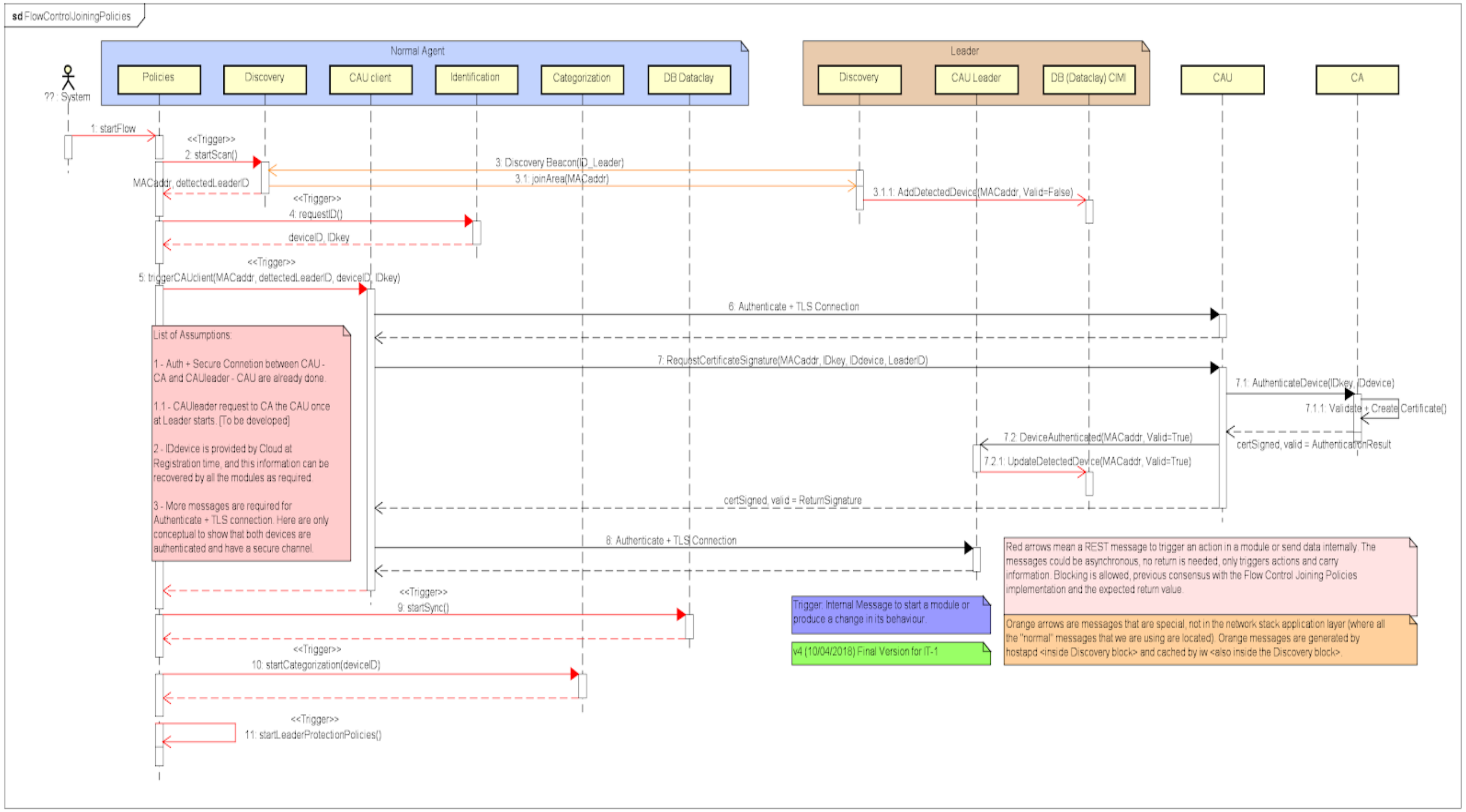
Devices are assumed to be Internet of Things devices and smartphones. They are very likely to hold PII. We shall not consider them here but will do so with the Use Case applications in Deliverable D5.3.

Results

For a description of the column headings refer to Annex 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			

Annex 9: Discovery, authentication and categorization workflow (large)


sd FlowControlJoiningPolicies
Normal Agent
Leader
?? System
Policies
Discovery
CAU client
Identification
Categorization
DB Dataclay
Discovery
CAU Leader
DB (Dataclay) CIMI
CAU
CA
1: startFlow
<<Trigger>>
2: startScan()
3: Discovery.Beacon(ID_Leader)
3.1: joinArea(MACaddr)
3.1.1: AddDetectedDevice(MACaddr, Valid=False)
MACaddr, detectedLeaderID
<<Trigger>>
4: requestID()
deviceID, IDkey
<<Trigger>>
5: triggerCAUclient(MACaddr, detectedLeaderID, deviceID, IDkey)
6: Authenticate + TLS Connection
7: RequestCertificateSignature(MACaddr, IDkey, IDdevice, LeaderID)
7.1: AuthenticateDevice(IDkey, IDdevice)
7.1.1: Validate + Create Certificate()
certSigned, valid = AuthenticationResult
7.2: DeviceAuthenticated(MACaddr, Valid=True)
7.2.1: UpdateDetectedDevice(MACaddr, Valid=True)
certSigned, valid = ReturnSignature
8: Authenticate + TLS Connection
<<Trigger>>
9: startSync()
<<Trigger>>
10: startCategorization(deviceID)
<<Trigger>>
11: startLeaderProtectionPolicies()
List of Assumptions:
1 - Auth + Secure Connetion between CAU - CA and CAUleader - CAU are already done.
1.1 - CAUleader request to CA the CAU once at Leader starts. [To be developed]
2 - IDdevice is provided by Cloud at Registration time, and this information can be recovered by all the modules as required
3 - More messages are required for Authenticate + TLS connection. Here are only conceptual to show that both devices are authenticated and have a secure channel.
Red arrows mean a REST message to trigger an action in a module or send data internally. The messages could be asynchronous, no return is needed, only triggers actions and carry information. Blocking is allowed, previous consensus with the Flow Control Joining Policies implementation and the expected return value.
Orange arrows are messages that are special, not in the network stack application layer (where all the "normal" messages that we are using are located). Orange messages are generated by hostapd <inside Discovery block> and cached by iw <also inside the Discovery block>.
Trigger: Internal Message to start a module or produce a change in its behaviour.
v4 (10/04/2018) Final Version for IT-1