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

D5.4 mF2C solution demonstration and field trial results (validation IT-2)

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

Work Package	WP5, mF2C – PoC integration and Demonstration strategy
Due Date:	M36
Submission Date:	29/12/2019
Version:	1.6
Status	Final
Author(s):	Andrea Bartoli, Denis Guilhot, Alejandro Lampropulos, Francisco Hernandez (WOS), Sašo Stanovnik, Kristian Žarn, Matija Cankar (XLAB), Antonio Salis, Roberto Bulla (ENG), Eva Marín, Xavier Masip (UPC)
Reviewer(s)	Jasenka Dizdarevic (TUBS) Ana Juan Ferrer (ATOS)

Keywords
PoC, integration, demonstration, testbeds, iteration 2 results

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

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

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	15/05/2019	First ToC circulated	Denis Guilhot (WOS)
0.1-XLABv7.4.9-gamma3	11/11/2019	First draft of the UC2 section.	Sašo Stanovnik, Kristian Žarn, Matija Cankar (XLAB)
0.1-ENG	11/11/2019	First draft of 2.4, 6.5	Antonio Salis, Roberto Bulla (ENG)
0.4	4/12/2019	Final outline and partial UC1 input	Denis Guilhot (WSL), Alejandro Lampropoulos (WSL), Andrea Bartoli (WOS), Francisco Hernandez (WOS)
0.5	4/12/2019	ToC changes and UPUC contribution	Eva Marin (UPC), Xavier Masip (UPC)
0.6	5/12/2019	UC1 inputs	Denis Guilhot (WSL)
1.0	16/12/2019	UPC section 3 input and final draft version	Eva Marin (UPC), Xavier Masip (UPC), Denis Guilhot (WSL), Francisco Hernandez (WOS)
1.1	17/12/2019	First review	Admela Jukan (TUBS)
1.2	17/12/2019	Answers to first review	Sašo Stanovnik (XLAB), Antonio Salis (ENG), Alejandro Lampropoulos (WSL)
1.3	18/12/2019	Second review	Ana Juan Ferrer (ATOS)
1.4	19/12/2019	Answers to second review	Antonio Salis (ENG), Eva Marin (UPC), Sašo Stanovnik (XLAB), Denis Guilhot (WSL)
1.5	19/12/2019	Quality assessment	Maria Teresa Garcia (ATOS)
1.6	29/12/2019	Final version ready to be submitted.	Denis Guilhot (WSL)

Table of Contents

List of figures

List of tables

Executive Summary

This deliverable is the logical continuation of D5.2 mF2C reference architecture (Integration IT-2). Both deliverables are also the equivalent of D5.1 mF2C reference architecture (integration IT-1) [1] and D5.3 mF2C solution demonstration and field trial results (validation IT-1) for IT-2 [2], as those two deliverables dealt with the same concepts, but for IT-1. D5.2 deals with the interaction between the mF2C components in the second iteration of the project in order to provide the final integrated version of the project reference architecture.

The present deliverable describes the deployment of the final proof-of-concept of the mF2C architecture in three real-world use cases implemented in three testbeds developed specifically for this project. Three scenarios are conceptualized:

- **Emergency Situation Management in Smart City**, in which an alarm is simulated, and the intervention of the emergency services managed in an optimised fashion.
- **Smart Boat,** in which monitoring of a fleet and a control system for boat owners and users are simulated using an extensible module add-on architecture.
- **Smart Fog-Hub Service**, which proposes a proximity marketing and topics adviser system for travellers in crowded places such as airports, railway stations or shopping centres.

This deliverable reports on the challenges faced during the demonstration deployments as well as the assumptions and modifications that resulted to be necessary in order to perform these real-world tests. The tasks realised allowed deeper understanding of the mF2C architecture and allowed the identification of small issues with the agent, the testbed and the integration of both components, which were solved in biweekly iterations, as well as conceptual improvements that could further improve the ecosystem.

In-depth characterisation has been performed and the results obtained are also presented, which demonstrate the relevance of the proposed architecture in the use cases selected and illustrate the benefits of using the technology developed in the mF2C project in each specific situation, as well as generic advantages demonstrated through common KPI. For instance, reduction of the services' latency is demonstrated in both the Emergency Situation Management and Smart Fog-Hub services whilst reliability is demonstrated in the Emergency Situation Management and Smart Boat use cases. Finally, an increase in coverage is demonstrated for the Smart Boat use case.

1. Introduction

The objective of this deliverable is to describe the design, implementation and evaluation of the use case scenarios that allowed to validate the mF2C management architecture developed in this project. The three selected scenarios are presented, as well as the deployment of the mF2C architecture on each of the testbeds, and the final results detailed.

The deliverable D2.7 mF2C Architecture (IT-2) [3] presented in M25 the final architecture for the second implementation of the project (IT-2), while each of the architecture blocks were described in more detail in deliverables D3.4 Design of the mF2C Agent Controller Block IT-2 [4] and D4.4 Design of the mF2C Platform Manager block components and microagents IT-2 [5]. D5.2 mF2C reference architecture (Integration IT-2) reports on the modifications of functionalities supported by the mF2C blocks for the final version of the architecture. This document builds on all those previous deliverables.

1.1. Purpose

This deliverable is structured in 7 sections:

Section 1. The current section introduces this document and presents the scope that will be described in the following sections.

Section 2. describes the proof-of-concept (PoC) that has been developed during the project and introduces the technical and scientific challenges that are illustrated in this project.

Sections 3., 4. and 5. detail the three proposed use cases, the testbeds they use, the structures that are being used for this first iteration, the data flow diagrams, the experimentation set-ups and the results obtained together with numerical KPI. Finally, business considerations for each one of the scenarios are introduced, that are developed in more detail in Work Package 6.

Sections 6. deals with the standalone demo, developed in order to facilitate the understanding of initial configuration steps of mF2C platform and complementary to the project use cases.

Finally, section 7. will present the conclusions reached during the work presented in this deliverable.

1.2. Structure of the document

This deliverable is structured in 7 sections:

Section 1. The current section introduces this document and presents the scope that will be described in the following sections.

Section 2. describes the proof-of-concept (PoC) that has been developed during the project and introduces the technical and scientific challenges that are illustrated in this project.

Sections 3., 4. and 5. detail the three proposed use cases, the testbeds they use, the structures that are being used for this first iteration, the data flow diagrams, the experimentation set-ups and the results obtained together with numerical KPI. Finally, business considerations for each one of the scenarios are introduced, that are developed in more detail in Work Package 6.

Sections 6. deals with the standalone demo, developed in order to facilitate the understanding of initial configuration steps of mF2C platform and complementary to the project use cases.

Finally, section 7. will present the conclusions reached during the work presented in this deliverable.

1.3. Glossary of Acronyms

Acronym	Definition
API	Application programming interface
CA	Certificate Authority
CAU	Control Area Unit
CAPEX	Capital expenditures
CIMI	Cloud Infrastructure Management Interface
CSR	Certificate Sign Request
DER	Distributed Execution Runtime
DoA	Description of the Action
DoS	Denial of Service
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HDMI	High-Definition Multimedia Interface
I2C	Inter-Integrated Circuit
IP	Internet Protocol
IMA	Industrial Monitoring Application
IoT	Internet of Things
IT-#	Iteration 1 or Iteration 2
JVM	Java Virtual Machine
LoRa	Long range LPWAN
LPWAN	Low-Power Wide-Area Network
MAC	Media Access Control
MQTT	Message Queue Telemetry Transport
NUC	Next Unit of Computing
OPEX	Operating Expenses
PCs	Personal Computers
PoC	Proof of Concept
POI	Points of Interest
QoS	Quality of Service
REST	Representational State Transfer
SDK	Software development kit
SDR	Software Defined Radio
SSL	Secure Sockets Layer
Timeout status	When a device, for example, a sensor, is not sending any data or status information.
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
USFF	Ultra-small form factor
VM	Virtual Machine

Table 1. Acronyms

2. PoC and Application description

2.1. mF2C PoC implementation

The main objective of the PoC was to implement all the functionalities and workflows in deliverables D2.7 [3], D3.3 [6] and D4.3 [7] which describe the final design and architecture of mF2C. This prototype must fit all the needs and functionalities required for use cases in IT-2. After the first prototype in IT-1 for which some components were postponed, the final architecture was refined and some components relocated as shown in Figure 1, corresponding to Figure 8 of deliverable D2.7 [3].

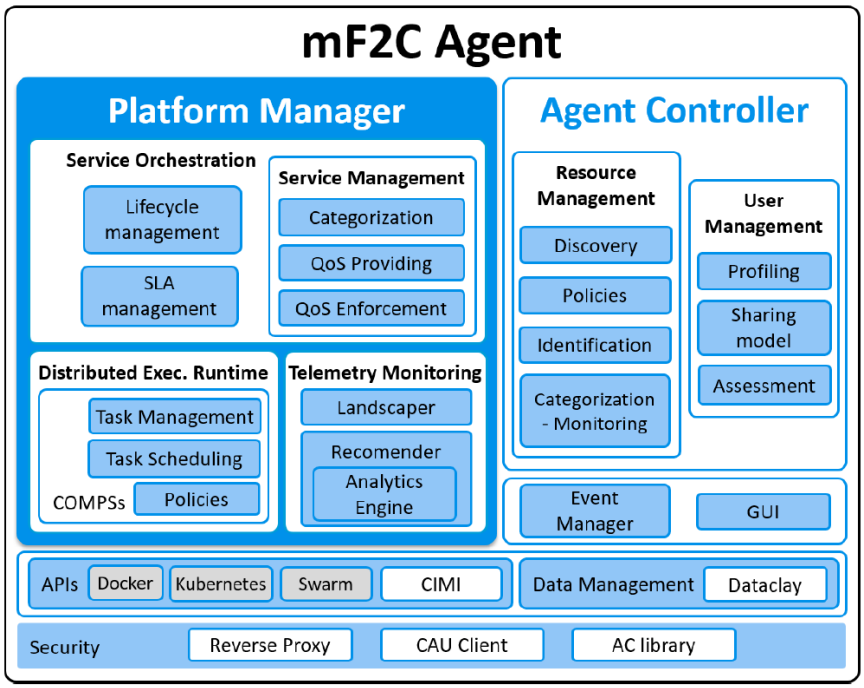

Figure 1 mF2C Agent architecture

The assumptions for integration in IT-2 were the same as described in D5.1 [1], that is, the use of docker, where the mF2C system was orchestrated through a single YAML file (Compose file) in which all the components were referenced, configured and automatically linked with each other, under an isolated local network.

As we did for IT-1, we performed a set of integration tests for IT-2, described in D5.2. We also performed some functional tests as described for IT-1 in D5.3 [2], in order to test all the mF2C functionalities, not only those involved in the use cases, but trying to cover all of them. In this sense, apart from the integration of the PoC in each one of the use cases described below, the developers' team has prepared two generic applications to be run on the system:

- The Hello-World app -> using COMPSs to deploy the app in different agents, that run the different tasks of the application in parallel.
- The Docker Hello-World -> which deploys a docker in one of the devices in the cluster, either microagent or agent.

More information can be found in D5.2.

2.2. Map of Scientific and technical challenges to the use cases

The mF2C architecture addresses several technical challenges which are being illustrated in the use cases. To simplify the matching of challenges and their demonstration, the following matrix maps the scientific and technical challenges to the use cases. These challenges identify additional research challenges complementary to mF2C and how use cases could or do respond to these beyond mF2C implementation.

Challenges	Reflection in use cases
Networking is a grand challenge in fog-to-cloud computing, which has stayed widely unaddressed in the research community. Since fog computing especially is expected to deploy a variety of connectivity solutions, ranging from high speed links to wireless connections, delay tests need to be factored in, along with a communication protocol of choice.	Use cases have taken into consideration the issue of the delay in one of the KPI implemented, the calculation of the latency, which compares run times using the platforms without mF2C and with mF2c and gives a numerical comparison of the tests related to the delay of the network.
In mf2C, different use cases use either only Wi-Fi or Ethernet for their networks. Agent however does not support Ethernet because the leader creates a hot spot to which the other agents connect to – which is an open challenge.	The use cases have avoided the use of Ethernet wherever possible, although the third use case illustrates it perfectly, since the mF2C Leader Wi-Fi beacon cannot be used due to airport on-site constraints, thus the use of wired network is mandatory.
Telemetry Monitoring has a large potential to making use of ML in fog-to-cloud systems. To efficiently deploying ML at scale, the number of services needs to be comparatively large. In this way, joint scalability and ML studies can be conducted while increasing the number of services (e.g., services registering into mF2C). When Reinforcement Learning is used, as it is the case with QoS provider in mF2C, data needs to be collected in use cases scenarios for longer periods of time to reaching meaningful conclusions.	Unfortunately, machine learning has not yet been used in the use cases because the number of runs is not sufficient for such elaborated techniques. Nevertheless, use case 1 for instance has collected more than 15.000 measurements in under a month and ML could be applied to higher number of instances of the use case.
An mf2C system would ideally be working with different and technically heterogeneous devices and systems, ranging from resource constrained RPis to more powerful PCs in the fog computing layer. There is open challenge however of implementing the full agent functionality on constrained devices. A software engineering challenge is to create elegant solutions of different versions of agents that would be supported by a variety of technologies and devices, including software and hardware technologies of the said device; for instance, a new agent implementation would run on a smartphone. Migration to a new and perhaps more powerful potential edge devices should be seamless, for instance by inclusion of edge devices with GPU accelerators.	The use cases are using different hardware to show the versatility of the agent, as can be seen in sections 3.1.1, 4.1.1 and 5.1.1. The hardware includes PC, virtual machines and cloud virtual machines, including the new ATOS edge server. Even Raspberry pies and Nuvla Nanos have been used for the micro agent. Nevertheless, the challenge of deploying the agent on gateways and constrained devices remains open.

Table 2 Scientific and Technical challenges mapping

3. Use Case #1: Emergency Situation Management in Smart Cities (ESM)

The first use case, led by Worldsensing, intends to explore the use of the mF2C technology in the context of smart/connected IoT and Industry 4.0 applications where continuous wireless communication takes place among different assets (autonomous or semi-autonomous) and the control centre, through a more efficient and flexible emergency management service.

The Emergency Situation Management applies to the infrastructure construction and monitoring in Smart Cities. It will analyse the flow of assets within smart infrastructures to provide useful information to infrastructure operators, detect potential emergency scenarios in a real-time approach and decrease the necessary resources in terms of energy, latency, etc. to efficiently respond to these situations in accordance to the applications' requirements.

Five main functionalities are required to efficiently implement the solution, as illustrated in Figure 2:

1. Monitoring: The monitoring has to be able to show real-time data of the emergencies and the location of the closest worker when relevant.
2. Detection: The detection functionality is responsible for handling the inclination data as well as for raising an alarm when the threshold is reached.
3. Positioning: The construction workers will be equipped with location devices. This position will be used to identify which construction workers are closest to the emergency being reported so they can be contacted to check what is really happening and report their diagnostic to validate or disprove the alert.
4. Scanning: The scanning is performed by the proprietary sensor used for this use case when a communication error is reported, to identify whether there is an alert there
5. Intervention: The intervention functionality corresponds to the intervention of the closest construction worker intervention reaction and also to the emergency vehicle path calculation, intervention, and the action on the traffic lights for clearance of the "green corridor" to optimise their intervention time.

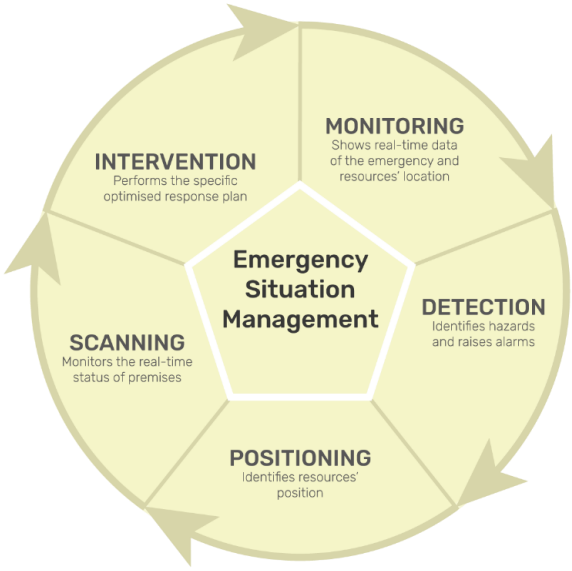

Figure 2 Emergency Situation Management Lifecycle

3.1. Implementing the mF2C Architecture in Worldsensing's use case

The following diagram represents the high-level architecture to be implemented in the use case of Worldsensing, including a demonstration. We consider two areas of the Smart City: Area 1 where the intervention means are located and Area 2 where the emergency situation occurs.

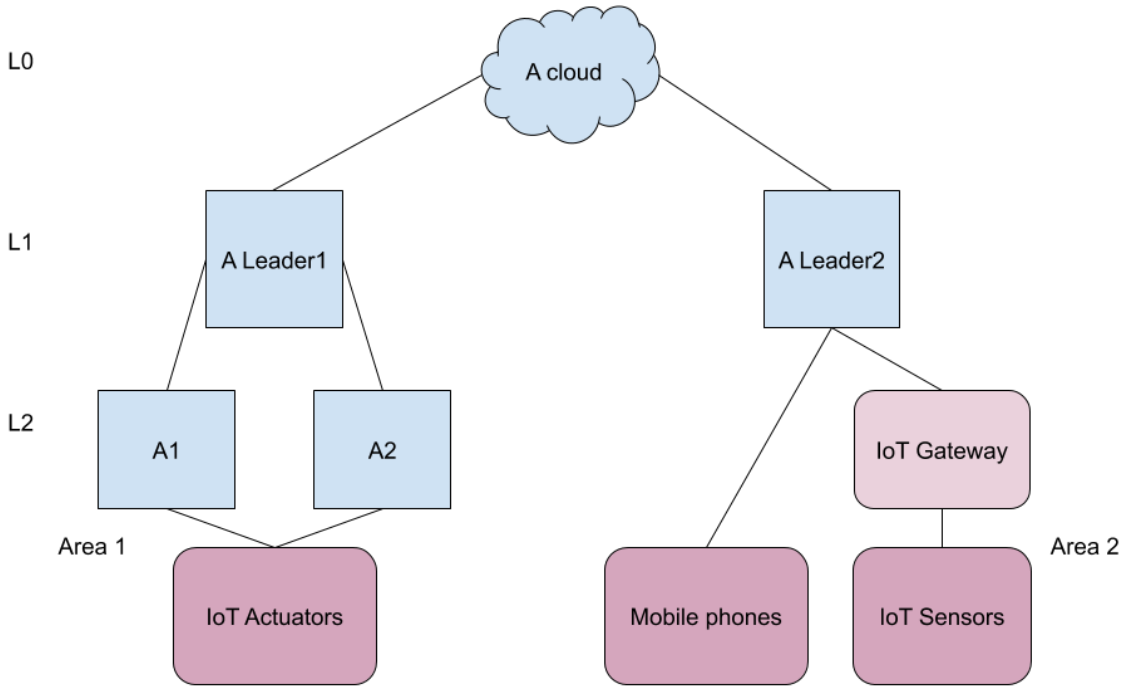

Figure 3 Complete Architecture of the Use Case

The following three F2C levels are installed in the final version of the pilot:

- Layer 0 Cloud
  - Cloud Virtual Machine: this virtual machine runs on the cloud and is used to receive and store the data coming from the IoT sensors through the Gateway. It also provides real time visualization tools, business rules that are applied to the data in order to trigger alerts and response plans associated, which respond to those alerts. This software is called the Industrial Monitoring Application (IMA). In case of an emergency, once the alert is confirmed on site, an automatic response plan will be started, calling the emergency services in Area 1. A cloud agent is running on this VM.
- Layer 1 Fog
  - We have two leaders, one in each area.
  - A Leader2 (right) is installed on a machine that runs the Lightweight IMA, allowing a Local Response of the emergency. Having this software on the Fog is essential to provide more reliability and improve delays regarding the control of emergency situations. In addition, it allows us to have more reactive on-site confirmation of the alert situations before triggering a response plan.

  - A Leader1 (left) is installed on a machine running only as a coordinator (leader) for Agent1 and Agent2
- Layer 2 Fog
  - In area 1, there are two agents, whereas in area 1 there are no agents in this layer L2.
- IoT and others
  - Gateway: used as a gateway for LoRa packets coming from dataloggers to be transmitted to the IMA. The Gateway needs to be capable of turning on and off the physical alarms.
  - Mobile phones: if these run the Industrial App, they can connect to IMA and IMA Light to provide confirmation of the alerts and on-site notifications. The application tracks the position of the phone, notifying the closest one of a potential alert so an emergency situation can be confirmed or cancelled.
  - Datalogger with tiltmeters: this IoT device measures the inclination on a structure and communicates it to the gateway via LoRa.

The previous IoT devices are sensing devices, and the IoT actuators are:

  - Ambulance connected to its corresponding agent through a Raspberry Pi.
  - Fire engine connected to its corresponding agent through a Raspberry Pi.
  - Traffic light connected to its corresponding agent through a Raspberry Pi.

More information can be found in section 4.4.

3.1.1.Current Hardware and base Software

The list of devices for the demonstration of use case 1 are the following:

Edge IoT devices (simple sensors and actuators)	
Mobile phone: Google Pixel 2	
Hardware	Software
Processor: Octa-core (4x2.35 GHz Kryo & 4x1.9 GHz Kryo) Volatile memory: 4GB RAM Non-volatile memory: 64GB flash Communication interfaces: 4G, WiFi (802.11mc compatible), Bluetooth, USB (type-C), NFC	Operating system: Android 9.0 (Pie), with 802.11mc support. Software packages: 802.11mc Round Trip Time calculation library. Worldsensing's Industrial App.
Loadsensing Tiltmeter's Inclinometer including an integrated LoadSensing datalogger	
Hardware	Software
Processor: ARM Cortex M4 48MHz Volatile memory: 32 kB Non-volatile memory: 256 kB + 4 MB flash Communication interfaces: I2C, UART, USB, LoRa modem (Semtech 1276) Sensor: Murata's SCA103T differential Inclinometer series	Operating system: FreeRTOS Software packages: sx1276 library Loadsensing application (112 kB flash, 24 kB ram, 4 MB storage)
LoRa Gateway	
Hardware	Software
Processor: Intel® Atom™ E3940 Processor Volatile memory: 2GB LPDDR4	Operating system: Ubilinux v4 LoadSensing firmware

Non-volatile memory: 32GB eMMC Communication interfaces: USB, Ethernet, LoRa, Bluetooth, 3G/4G, WiFi	
Ambulance	
Hardware	Software
Processor: Quad Core 1.2GHz Broadcom BCM2837 64bit Volatile Memory: 1GB Non-volatile memory: 16GB Communication interfaces: USB, Ethernet / BCM43438 wireless LAN	Operating system: Raspbian Jessie (A Debian Linux operating system for RaspberryPI) Software Packages: Python, Car_Controller
Fire engine	
Hardware	Software
Processor: Quad Core 1.2GHz Broadcom BCM2837 64bit Volatile Memory: 1GB Non-volatile memory: 16GB Communication interfaces: USB, Ethernet / BCM43438 wireless LAN	Operating system: Raspbian Jessie (A Debian Linux operating system for RaspberryPI) Software Packages: Python, Car_Controller
Traffic light	
Hardware	Software
Processor: Quad Core 1.2GHz Broadcom BCM2837 64bit Volatile Memory: 1GB Non-volatile memory: 16GB Communication interfaces: USB, Ethernet / BCM43438 wireless LAN	Operating system: Raspbian Jessie (A Debian Linux operating system for RaspberryPI) Software Packages: Python, Traffic_Light_Controller
L2 Fog-capable devices	
Virtual machines (VMs) (UPC)	
Hardware	Software
Processor: Core i5 Volatile memory: 12 GB DDRAM Non-volatile memory: 256 GB HD Communication interfaces: USB, Ethernet	Operating System: Ubuntu 16.04 LTS Software Packages: Docker, docker-compose, Emergency_resolution_service
L1 Fog-capable devices	
WOS Leader (PC-1)	
Hardware	Software
Processor: Core i5 Volatile memory: 16 GB DDRAM Non-volatile memory: 256 GB HD Communication interfaces: USB, Ethernet, WiFi	Operating system: Ubuntu 16.04 Software packages: Docker, InfluxDB Time-Series Database, Kapacitor Worldsensing's IMA Light
UPC Leader (Virtual machines - VM)	
Hardware	Software
Processor: Core i5 Volatile memory: 12 GB DDRAM Non-volatile memory: 256 GB HD Communication interfaces: USB, Ethernet	Operating System: Ubuntu 16.04 LTS Software Packages: Docker, docker-compose, Emergency_resolution_service

L0 Cloud-capable devices	
Cloud Machine	
Hardware	Software
Processor: Core i5-4570 CPU @ 3.20GHz x 4 Volatile Memory: 16 GB DDRAM Non-volatile memory: 1 TB HDD Communication interfaces: USB, Ethernet	Operating System: Ubuntu 16.04 Software Packages: Docker, docker-compose, Worldsensing's IMA

Table 3 List of devices for the demonstration of UC1 in IT-1

For clarity purposes, we group the different set of devices listed above in three categories:

1. LoadSensing's group
    1. Inclinometer (IoT), integrating a datalogger, called Tiltmeter
    2. LoRa Gateway (IoT)

b. Local response group:
    a. Mobile phones

a. Actuator group
b. L2- group- VMs connected to actuators and used for computation purposes.
c. L1- group machines acting as leaders
d. Cloud Server
    a. (L0: Cloud)

The way they are coordinated and work together is:

- Worldsensing's IMA Light is started by the mF2C System on an agent with enough resources to run it. Once it starts, it synchronizes its configuration (sensors onboarding, business rules, response plans, etc) with the Cloud IMA.
- Worldsensing's IMA Light periodically receives the position of the workers running the Industrial App.
- When (L1 Fog) Worldsensing IMA Light, through a (IoT) Tiltmeter + Gateway, detects an inclination value that exceeds a safe threshold (business rule), it will start the siren alarm and notify the closest worker to the threat, so that they confirm the alert situation.
- The closest worker confirms the threat using the Industrial App, thus the Cloud IMA is notified.
- Cloud IMA triggers the response plan, calling the mF2C agent to start the emergency services.
- The emergency services triggered by the detection of the alert are in fact three different sub-services:
    - An ambulance is searched.
    - A fire engine is searched.
        - 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 (and blue indicating an emergency) in the computed paths.
        - The ambulance and the fire engine move automatically to the building through the computed paths.

3.1.2.Software Responsibilities

There are software functionalities in all the layers of the infrastructure, as will be explained below. The functionalities of this Cloud and Fog infrastructure will be extended to execute other tasks that would fulfil better the Use Case objectives. Current and new functionalities are described below:

Cloud Software - Industrial Monitoring Application

Worldsensing's IMA

1. Receive LoadSensing's sensing data through the Gateway:
    1. Receive data from the dumb sensors, for example the Tiltmeter, through the LoadSensing datalogger, via the LoRa Gateway.

b. Check sensor thresholds:
    a. If sensor thresholds are exceeded, the software creates an alert
    b. If the alert is confirmed, the software starts the emergency service system, as a response plan.

a. Call emergency services
b. Show sensor measurements in real time
c. Show infrastructure alerts.
d. Manage sensor onboarding, creation of business rules and response plans

Leader 2

Worldsensing's Industrial Monitoring Application Lightweight:

1. Synchronizes with the Cloud IMA when started
2. Periodically tracks the position of the mobile phones running the Industrial App (even indoors)
3. Receive LoadSensing's sensing data through the Gateway:
    a. Receive data from the dumb sensors, for example the Tiltmeter, through the LoadSensing datalogger, via the LoRa Gateway.

a. Check sensor thresholds:
    a. Apply the configured business rules to detect if a threshold is exceeded. If it is, generate an alert to be confirmed

a. Notify closest mobile phone to an alert situation to confirm the alert

Mobile phones

Worldsensing's Industrial App

1. Calculate indoor position of the device
2. Periodically communicate the position to the IMA Light
3. Receive notifications from the IMA Light, regarding potential alert situations to be confirmed

LoadSensing Gateway

1. Receive sensor data from Loadsensing dataloggers
2. Provide sensor data to the Cloud (IMA) and the Fog (IMA Light):
    a. All sensor data is forwarded

a. Turn on physical alarm (siren) when there is an emergency

LoadSensing Datalogger Firmware

1. Read LoadSensing tiltmeter (inclinometer) values.
2. Transmit inclinometer values to LoRa Gateway wirelessly.

Leader 1

This software executes the functionalities corresponding to the emergency resolution service:

1. Receive the request from the IMA (Cloud).
2. Compute the paths for the vehicles from the current location to the specific destination (collapsing building).
3. Send to the vehicles the path to follow.
4. Calculate the state of the traffic lights in the computed paths and send the required actions to the involved traffic lights.

Virtual machines (Agent 1 and Agent 2)

This software executes the functionalities corresponding to the emergency resolution service:

1. Receive the request from the IMA (Cloud).
2. Compute the paths for the vehicles from the current location to the specific destination (collapsing building).
3. Send to the vehicles the path to follow.
4. Calculate the state of the traffic lights in the computed paths and send the required actions to the involved traffic lights.

Note: Both, agents in layer fog 2 and leader have the same software, allowing to allocate the different subservices (corresponding to functionalities) in the different agents, if necessary.

Ambulance

This software executes the following functionality:

1. Drive and control the vehicle to reach the specified destination.

Fire engine

This software executes the following functionality:

1. Drive and control the vehicle to reach the specified destination.

Traffic light

This software executes the following functionality:

1. Change the state of the traffic light to allow pass the vehicles and clear the street according to the signal sent by the emergency resolution service.

3.1.3.Tasks to be executed by the mF2C Agent

The main outcome of this use case will be taking decisions according to inclination sensor (tiltmeter) measurements to monitor emergencies in infrastructures. In the present illustration of the use case, a tiltmeter (IoT device) will send measurements to the Gateway, which will forward them to the IMA (Cloud and Fog) instances. IMA will check the thresholds and decide whether those thresholds are within the acceptable limits or if a dangerous situation is detected, in which case alerts will be created and an emergency service will be triggered.

The mF2C agent will enhance the solution by executing IMA Light on shared resources on the fog. If the resources are enough, even a backup instance could be started. Furthermore, mF2C agents can be called to execute the emergency services, as long as these are registered on the mF2C system.

Having an mF2C infrastructure that supports running IMA services on the Fog (IMA Light) will provide more reliability and QoS to the solution, while keeping hardware cheaper since the Light version requires less computing power and can be deployed on a standard, low-end hardware. In addition,

time response to emergency situations will decrease if the services are run locally, rather than on the cloud.

3.2. Use Case Storyline

The storyline of the use case for IT-2 is as follows: the location of construction workers can be displayed on dashboard and maps and stored for use in the event of an emergency. Both these cases are evaluated to ensure compliance with the GDPR, options include requiring consent forms, using only tracking mobile devices provided by the company, not displaying the location except in case of an emergency and not associating a specific worker to a specific device, since the only requirement is locating the device closest to the emergency. Obviously, several combinations of these potential solutions are also contemplated. The final decision is still being studied by the legal department at this stage. The tiltmeter (also proprietary sensor) regularly sends data to the Gateway, that forwards it to the Monitoring Software, which shows the data on the dashboard and maps. Meanwhile, a jammer detector is idle. Once the Monitoring Software detects that a tiltmeter threshold was exceeded, the response plan is activated. First, the sirens are started. Then, the closest worker of the use case is identified through the location information stored previously. A message is sent requiring the worker to go and check the emergency situation. The mobile device can be used to report to confirm or cancel the emergency. The authorities and the relevant actors are alerted, and the alerts visualized on the dashboard and maps in real time. The warnings are emitted, and the emergency vehicles are sent, through the optimum path calculated at that moment. Finally, the traffic lights are changed to green on the path of the emergency vehicles in order to optimise the intervention time.

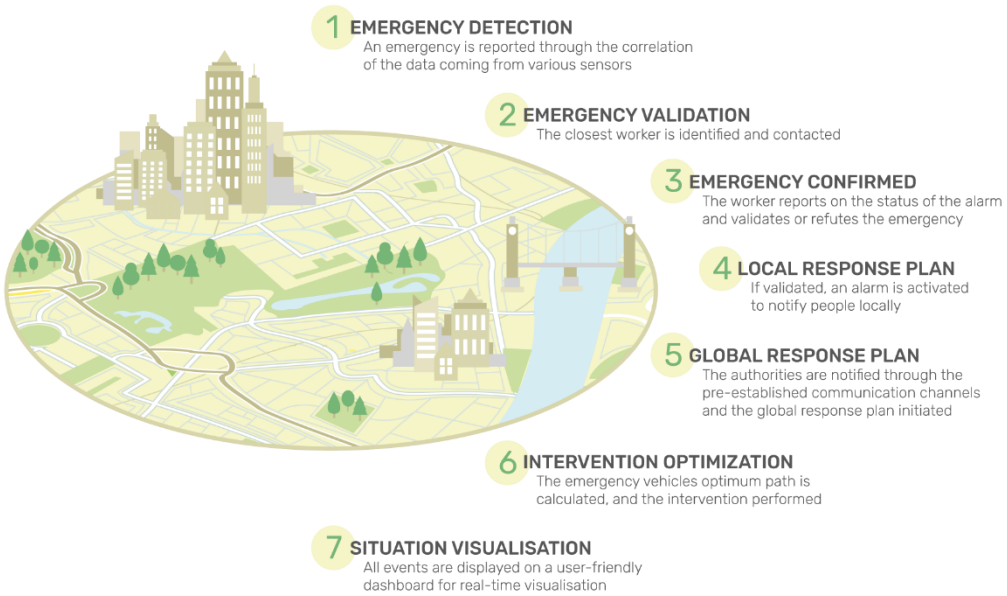

Figure 4 Emergency Situation Management Scenario

In a more technical perspective, the data collected by the tiltmeters is reported periodically to the Gateway and these data are forwarded to the Cloud, where IMA provides visualization in real time. IMA also provides methods to onboard these sensors, as well as to create business rules and response

plans related to the alerts generated by those rules. There is also the Industrial App running on the workers' mobile phones. This application calculates the indoor position of the workers and communicates it periodically to IMA. It can also receive notification messages and provides an interface to confirm alerts.

If the inclination exceeds a certain threshold (business rule), it will be detected by IMA and a message will be sent to the Gateway to trigger the siren. The closest mobile phone to the threat will be notified for its user to confirm or cancel the alert. If the alert is confirmed, the response plan will be started. This response plan will call the emergency services, so that an ambulance and fire engine are sent for help and the traffic lights will be smartly activated to allow an optima intervention time of the vehicles.

Having mF2C agents working together adds the possibility to locally start IMA Light, which is a lightweight version of the IMA. This application can run on any device in the Fog that can share enough resources for it. Once it is started, it retrieves all necessary information from the Cloud IMA regarding the onboarded sensors, business rules and response plans. If there is an alert situation, it will detect it locally, start the siren and notify the closest worker, so that they can confirm the alert.

On the other hand, the Cloud IMA will trigger a response plan by calling the mF2C agent to start emergency services to call an ambulance, a fire engine and manage the traffic lights. If the services are known to the agent, it will smartly trigger them on the corresponding machine. The Cloud Agent will call Agent 1 to trigger all three services (see Figure 5). Agent 1 will start the ambulance service on itself and will automatically go through the Leader 1 agent to trigger the other two services on Agent 2. These services will calculate shortest paths and send the vehicles, as well as activate the smart traffic lights.

3.3. Data Flow Diagrams

3.3.1. IoT sensors to Monitoring Software (no mF2C)

The following data flow diagrams do not specifically include the modules in mF2C agents, or the architectural benefits of mf2C. This is in fact the normal flow of data among the different modules and actors in a standard installation with minimum requirements. Instead, we show the use case specific flow and operation.

3.3.1.1. Normal Operation

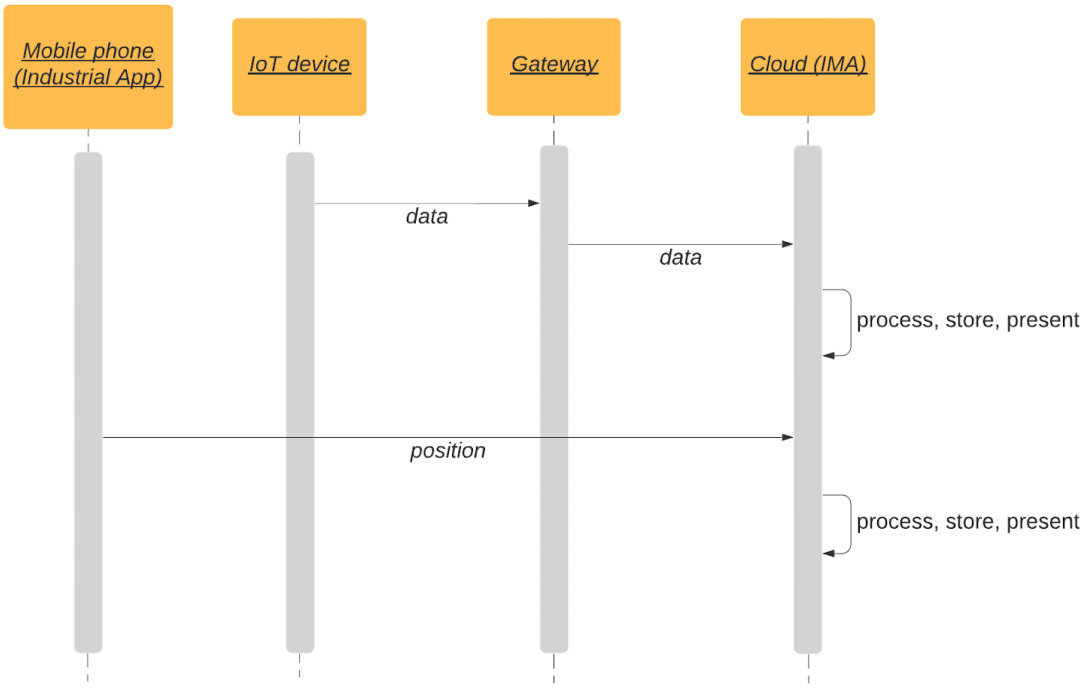

Figure 5 Normal operation (no mF2C) data flow diagram

We can see that data measured by the IoT device (in this case, a Tiltmeter) is communicated to the Gateway and later to the Cloud, which is the module where the data is processed to detect alarms, it is stored and it can be presented to users in real time. The Industrial App provides location of the workers periodically.

3.3.1.2. Inclination threshold

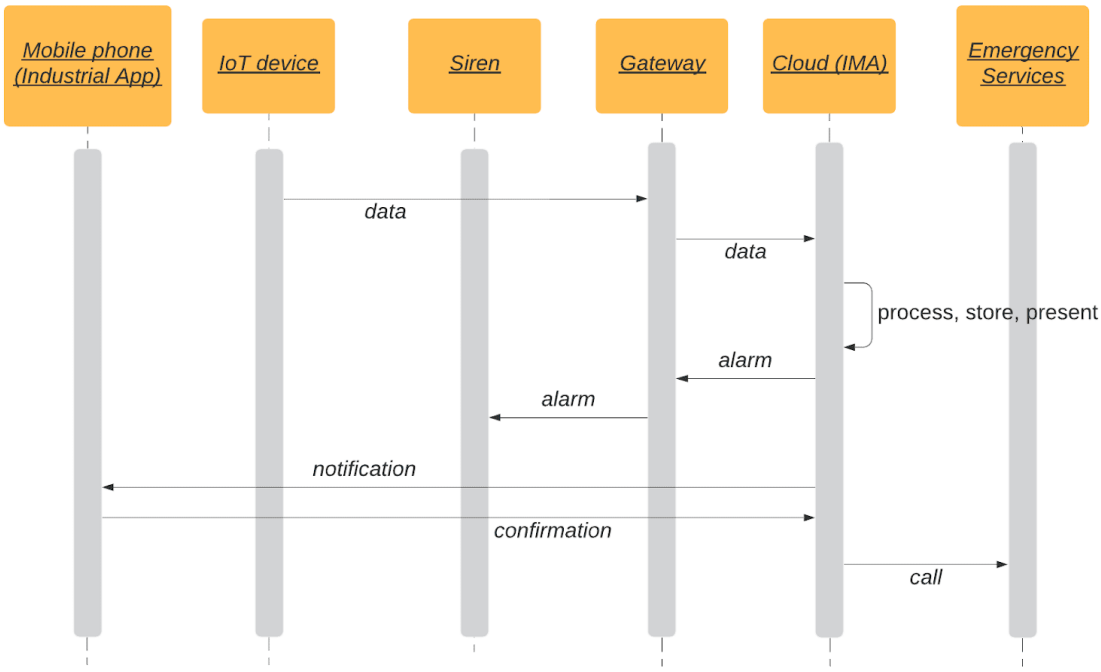

Figure 6 Inclination threshold (no mF2C) data flow diagram

In case a measurement exceeds a threshold, IMA on the Cloud will get back to the Gateway to start the physical alarm and also notify the closest worker to the potentially critical situation. Once the worker confirms the alert, the response plan is triggered, calling the emergency services directly.

3.3.2.IoT sensors to Monitoring Software (with mF2C)

This scenario considers the use of mF2C agents in Fog and Cloud. If the Fog environment has enough resources to share, the mF2C System can be called to start IMA Light Services. When started, these services need to synchronize with the Cloud to retrieve information about sensors in the system, business rules, etc. Once everything is set, the advantage will be to have a faster response locally, independently of the internet connection.

3.3.2.1. Normal operation

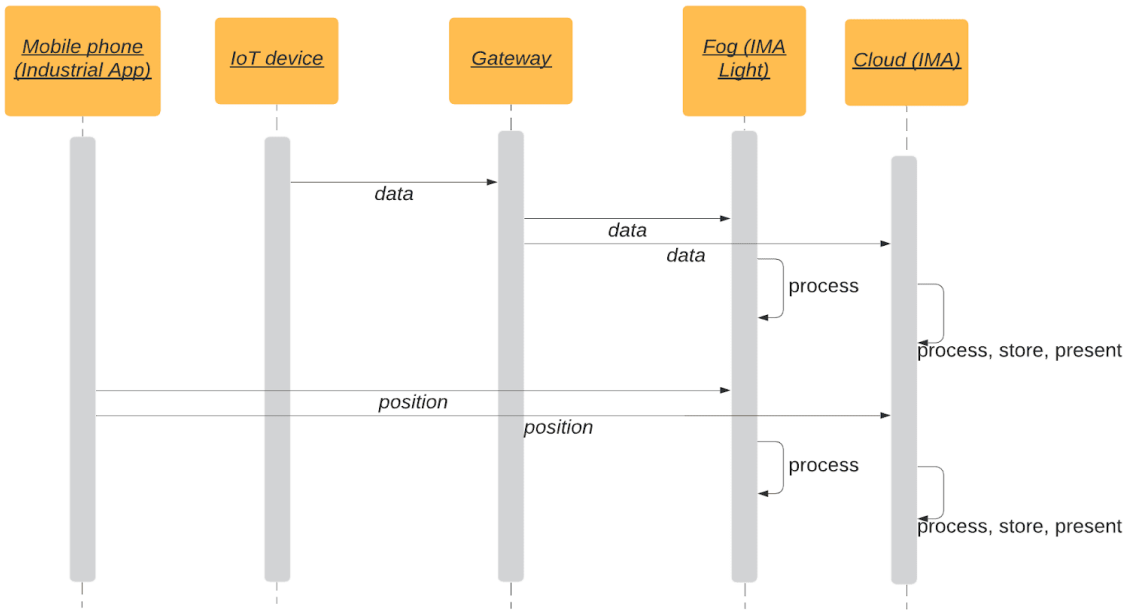

Figure 7 Normal operation (mF2C) data flow diagram

In this case, the IMA Light running on the Fog processes all data and locations coming from IoT devices and the Industrial App instances, respectively. It will analyse the data to detect exceeded thresholds and update the positions of the workers periodically.

3.3.2.2. Inclination threshold

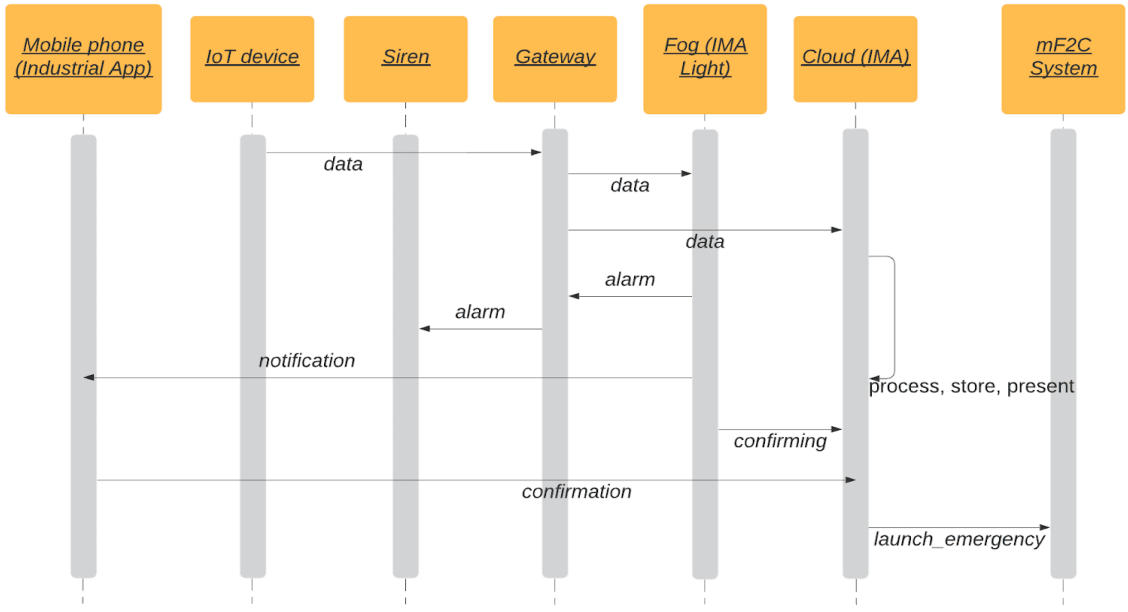

Figure 8 Inclination threshold (mF2C) data flow diagram

In case of an emergency, IMA Light will detect the alert faster and activate the physical alarm. It will also notify the closest worker to confirm the alert and send a message to the Cloud IMA in order to inform that the alert is being confirmed. Once the alert is confirmed, the response plan is activated in

the Cloud. In this case, the mF2C System is called to launch emergency services. The mF2C will smartly decide where each service will take place.

3.3.3. Emergency Resolution Service

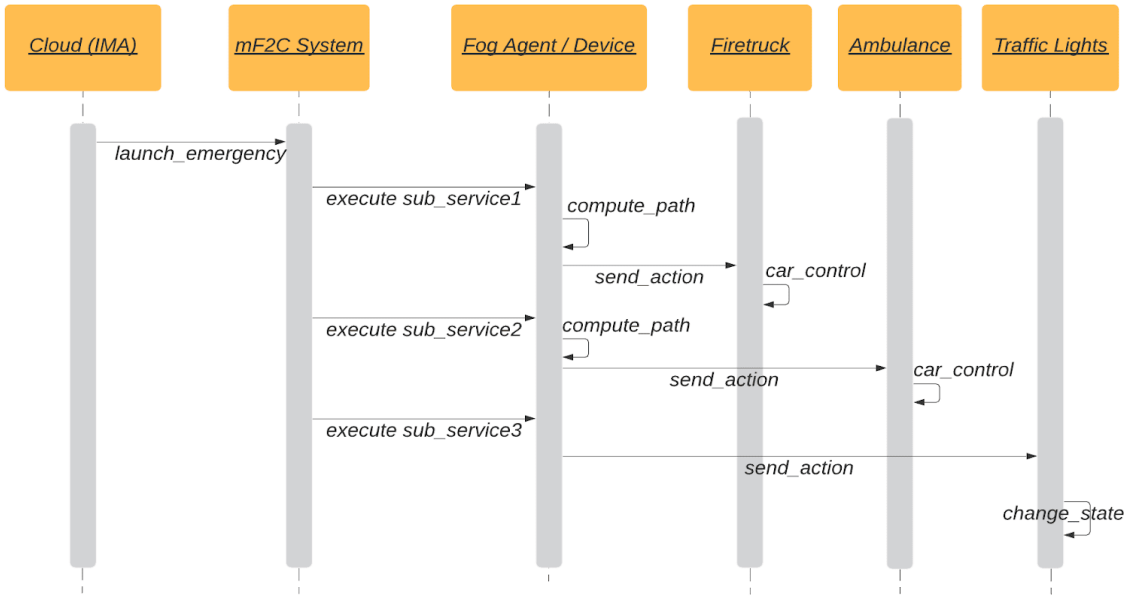

Figure 9 Emergency resolution service (mF2C) data flow diagram

The monitoring software sends the request to execute each one of the subservices to the Lifecycle (mF2C system), the Lifecycle looks for suitable agents where executing the requested subservices, then allocates and launches them in the corresponding agents.

3.4. Experimental Set UP

The test bed is deployed at the UPC laboratories in Vilanova I la Geltrù and the following diagram represents the whole use case actors and their interactions, according to the actual implementation for testing and demo purposes:

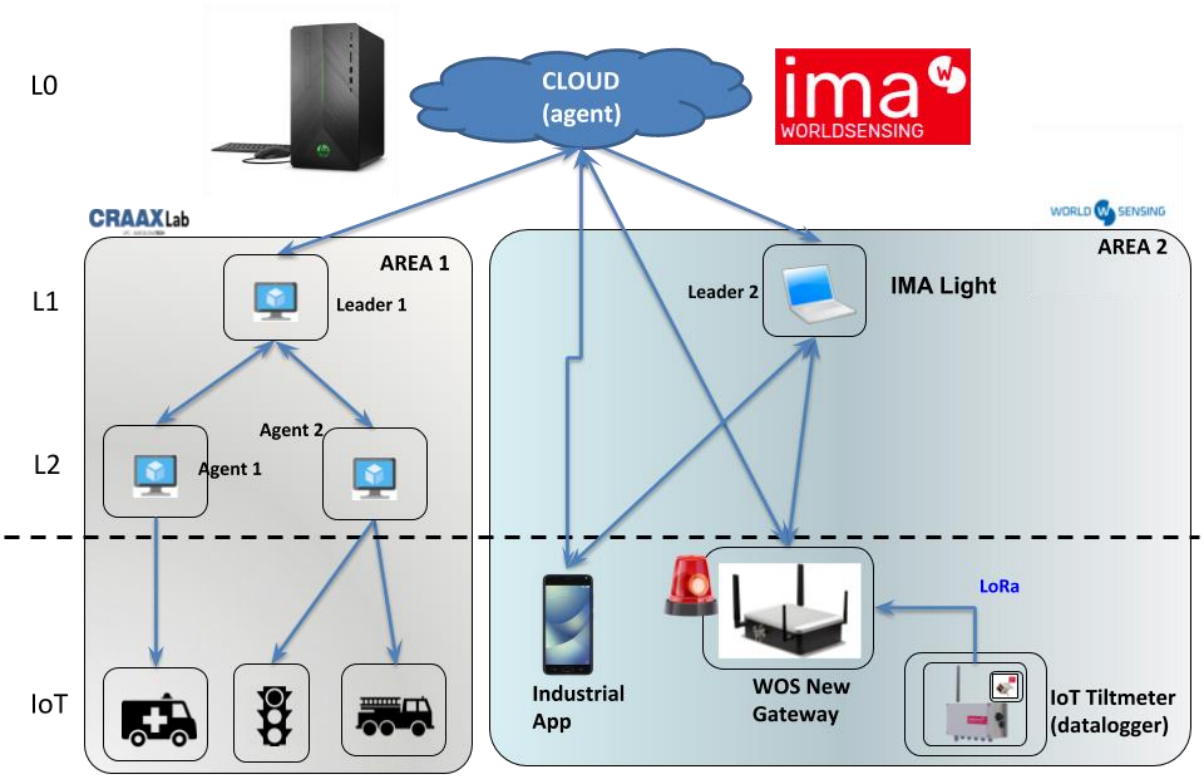

Figure 10 Use Case 1 experimental set-up

Worldsensing (Area 2):

IoT and others:

- non-mF2C capable LS-G6 Tiltmeter: measures inclination.
- non-mF2C capable LS-G6 Gateway (enhanced): forwards sensor data and activates physical alarms.
- non-mF2C capable Google Pixel 2 mobile phone: runs Industrial App (locates device, communicates location, receives notifications, confirms alerts).

L1 (Fog):

- mF2C Ubuntu HP-Laptop - Leader 2 (PC): runs mF2C Leader and IMA Light.

UPC (Area 1):

L0 (Cloud):

- mF2C capable Ubuntu Desktop (PC): runs mF2C Cloud Agent and IMA.

L1 (Fog):

- mF2C capable Ubuntu Linux - Leader 1 (VM): runs mF2C Leader.

L2 (Fog):

- mF2C capable Ubuntu Linux - Agent 1 (VM): runs mF2C Agent and executes Ambulance emergency service.
- mF2C capable Ubuntu Linux - Agent 2 (VM): runs mF2C Agent and executes both Traffic Lights and Fire Truck emergency services.

IoT:

- non-mF2C capable Raspberry PI (Ambulance): drives the ambulance to the emergency site.

- non-mF2C capable Raspberry PI (Firemen vehicle): drives the vehicle to the emergency site.
- non-mF2C capable Raspberry PI (Traffic Lights): turns the traffic light colors to allow emergency vehicles to circulate.

As mentioned before, the mF2C topology is the following: in Area 1, there are two regular agents (Agent 1 and 2) connected with the Leader 1 Agent. All of them run on virtual machines and are capable of running the emergency services if required. Nevertheless, for the tests and demo, we have set a limit to Agent 1 to run only 1 service at a time, simulating a high load on its resources. In this case, the mF2C ecosystem will decide to forward the request of the other 2 services, which will be finally executed in Agent 2, as shown in the figure.

Then, there is the Cloud Agent, running on a desktop machine, along with Cloud IMA. On Area 2, there is a second Leader Agent, running on a laptop machine, along with the IMA Light. The IMA Light Service is started by the Leader 2 Agent.

On the other hand, there are other devices in the IoT layer, which do not run any mF2C agents since they are too small, but are needed to complete the use case demonstration, as contemplated since the early stages of the project. On Area 1, these are the IoT devices that drive the ambulance, fire engine and traffic lights. On Area 2, we have the IoT Gateway, Tiltmeter and Mobile phones (running the Industrial App).

3.5. Results and KPI measurements

To quantify the performance, and provide KPI measurements, we integrated the above use case with the mF2C service management. In this case, the agent is capable of launching the IMA Light Service on the Fog on any device that has enough resources to run it. In addition, the Cloud IMA was configured to have a response plan that calls the mF2C Agent whenever an emergency service must be started. This allows us the flexibility to run those services on the corresponding agents seamlessly and profit from the distributed platform.

The whole sequence involves the Agents starting up and running the IMA Light Services on the Fog. These services will synchronize with the Cloud instance in order to get the sensors, business rules and response plans configuration. Then, when a Tiltmeter's inclination is more than the established threshold for the business rule, the IMA Light contacts the Gateway to start the siren and notifies the closest worker to confirm the alert. Once the confirmation is done, Cloud IMA's response plan is activated, calling the Agent to start the 3 different emergency services. These services will be launched each one on a different Agent machine, considering their resource limitations. When executing, services will calculate the paths of the vehicles and start them, as well as contacting the traffic lights device to change them.

Figure 11 Pictures of the testbed

Having the possibility to deploy the IMA Light on mF2C capable Fog devices is an advantage, as it provides better reliability and improves the latency of a response to an alert situation, when the software runs on the Fog. Moreover, it allows manufacturing more economical hardware, which is not necessarily intended to run the monitoring itself but to relay that aspect to other Fog or Cloud machines.

3.5.1.KPI measurements

As mentioned, latency can be improved significantly by running the IMA Light in the Fog, as the local network will introduce less delays than the Cloud itself. In order to prove this, we have measured the time it takes to the physical alarm to be started, from the moment that the Tiltmeter measurement reaches the Gateway. The following diagram explains the sequence and the delays measured.

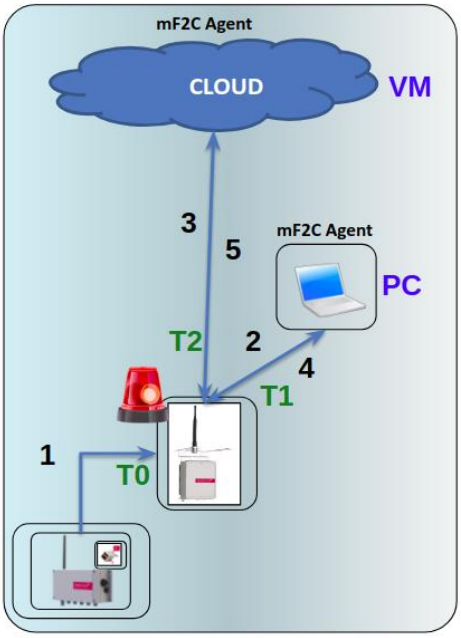

Figure 12 Latency KPI measurement strategy

The Tiltmeter sends message 1 to the Gateway. This gateway receives the packet (T0), processes it and publishes it on an MQTT Broker. Both the Lightweight IMA and Cloud IMA subscribe to that queue and obtain the packet (messages 2 and 3). Both will process it, evaluate the thresholds and send an alert message (4 and 5) back to the Gateway so that the physical alarm is activated. Messages 4 and 5 will define T1 and T2, respectively.

We calculate Tx = T1-T0 and Ty = T2-T0. We have worked on the automation of the measurements and were able to repeat the tests more than 23500 times. As we cannot present all the results in this document, we decided to calculate the average and standard deviation for IMA Light and Cloud IMA. These are presented next:

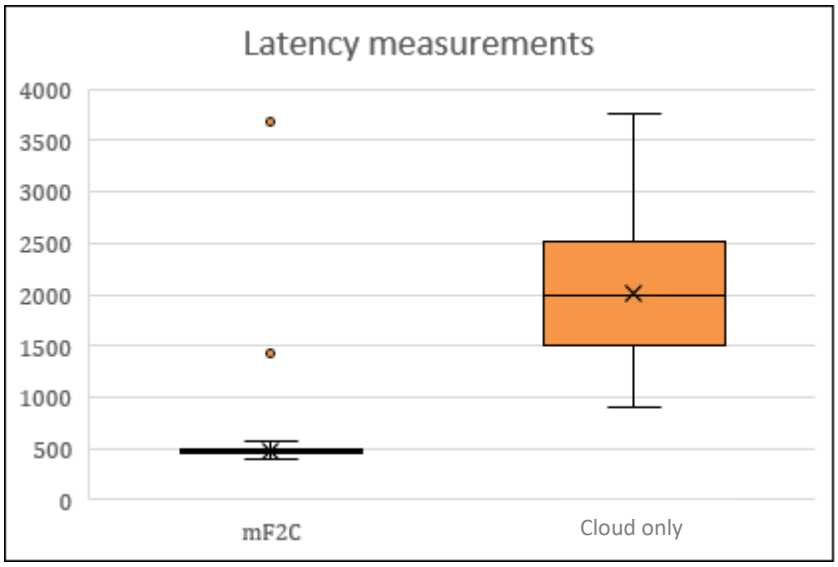

Figure 13 Latency measurements statistics for the 23.500 measurements taken

Application	Average	Standard deviation
Cloud IMA (Ty) - Cloud	2004 ms	577
Lightweight IMA (Tx) - Fog	479 ms	38
Improvement	76%	

Table 4. latency KPI measurement data analysis

As we can see, running the Lightweight IMA (Fog) will provide 76% improvement to the Cloud IMA, of the latency to start the alarm. It is important to mention that the transmission time between the Tiltmeter and the Gateway was not taken into account for these experiments. This is because it varies randomly and it is highly dependent on the sampling rate that is configured on the datalogger, which for these types of solutions that require real time reaction, should be 1 or 2 seconds. Moreover, these latencies will not vary with or without mF2C.

The architecture is validated through three others commercially relevant KPI. As explained in the first iteration, ensuring a high reliability is critical for infrastructure monitoring and to be competitive in this market. Here, thanks to the dense environment of agents that mF2C will provide, the IMA light can be run on different devices as a backup of the commercial solution, which would run only in the cloud. Various instances of the IMA light can be run at the same time, providing a much better reliability of the solution, close to 100%.

Assuming that there will be a high number of mF2C capable devices, having at least one IMA Light instance running on the Fog as a backup of the one running on the Cloud, we can say that the Monitoring Service will be up and running close to 100% of the time, we will consider 99.9%.

The quality of service is tightly related to the reliability since, if mF2C was not being used and the solution was intended to be only kept to the cost of the sensors, gateway and cloud provider, then we should consider that the Internet connections to the Cloud is not ensured with a 100% availability. For instance, Vodafone, which is one of the most important Internet service providers in Spain, provides compensation of the monthly bill if the service is down more than 48hs during the month (http://www.vodafone.es/c/statics/pdf-calidad-del-servicio-conocenos/). This means that the QoS they provide on availability is 93.33%. For this reason, and to allow for potential system failures of the system, Worldsensing would normally commercially offer a 90% QoS. If we consider that the reliability can be improved by reducing the Cloud dependency and by making use of the intrinsic redundancy offered by mF2C, Worldsensing can increase the availability of its commercial offer to 99% with the use of the mF2C environment to run at least one backup instance of the Monitoring Software. This represents a 10% improvement.

Finally, also linked to the intrinsic redundancy offered by the framework, the CAPEX and OPEX are reduced because the use of IMA light allows us to use gateways with a lower capacity. The AAEON AIOT-ILRA01 gateway, that costs 439 euros, has been replaced by other, less capable gateways. Because of its technical specifications and exact requirements of the use case, we have used the Tektelic KONA Micro IoT Gateway that costs 375 euros. This represents CAPEX savings of 17%. Also, by translating some of the IMA functionalities to the IMA light, we can reduce the virtual machine installed on the Cloud, which would also translate into OPEX savings, although this action has not been taken yet since we have focussed on reliability and complementarity of the solutions.

3.6. Business Prospective and conclusions

For this reason, the mF2C architecture is a great improvement for Worldsensing's solution as it allows intrinsic redundancy to the system. The current solution is cloud-based only and relies on a constant connection to the cloud. If it becomes unavailable through a network failure, because of interferences,

or in the case of a network saturation, the solution proposed might not be trustworthy. The competition is scarce for this reason. Being able to provide a QoS of 99% or more would provide Worldsensing with an unfair advantage to capture most of the existing market.

mF2C allows to increase the reliability to almost 100% and the Quality of Service by 10%, while decreasing delays by 76%. This provides a crucial commercial advantage which makes the solution commercially relevant. Finally, the mF2C provides an inherent redundancy to the system. In order to guarantee a correct service, any installation would require both physical and digital redundancy, in order to reduce issues. The intrinsic redundancy provided by the mF2C architecture allows for a CAPEX reduction of 17%, through the reduction of dependency on both the hardware (hence maintenance and service) and the Cloud, reducing the Cloud fees.

Moreover, the solution's components have important exploitation possibilities, described in more detail in D6.7 and summarized in the following figure:

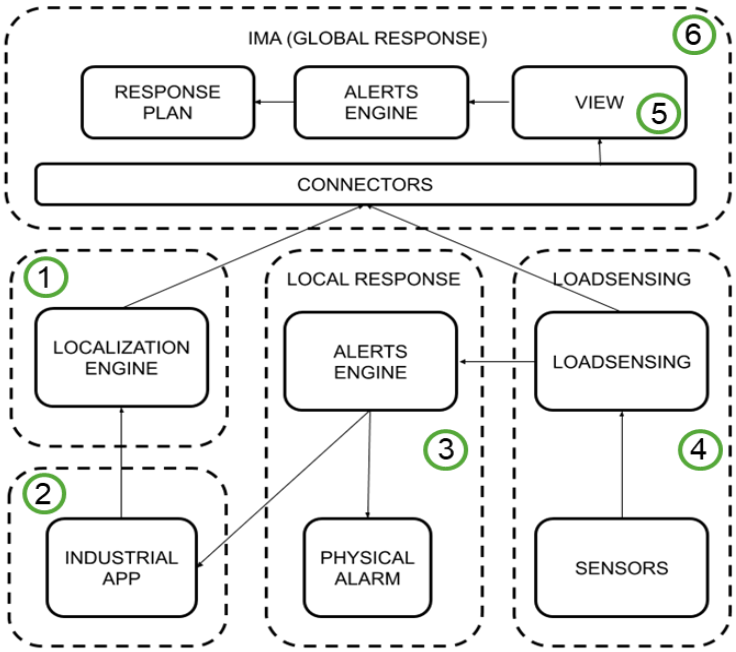

Figure 14 Exploitation possibilities by component

The first component (1) is the localization app, integrated with outdoor location in an industrial Proof-of-Concept partnership with a commercial partner. Component (2) is an industrial app developed based on an industrial development performed for a commercial client. Component (3) is the local response Proof-of-Concept, developed and now ready for on-premise business opportunities. Component (4) is the Loadsensing commercial product and improvements such as real-time measurements have been added in the frame of the project which are now integrated in our commercial offer. Also, the visualisation tool, integrated in one of the current commercial solutions, is presented as component (5). Finally, the global response system originated an industrial proof-of-Concept for emergency situation management to be used for technology evangelisation purposes.

4. Use Case #2: Smart Boat Service (SBS)

4.1. Complete Architecture Use Case

The four scenarios for the Smart Boat use case are

1. continuous boat monitoring,
2. anomaly detection,
3. online docking and mooring reservation and
4. data plan sharing.

We have demonstrated functionalities for 1, 2 and 4 in IT1. In IT2, we have enhanced existing features and covered scenario 3. Several networking scenarios for UC2 were described in Deliverable D5.3 Section 5.1 [2]and while they are still present, we have put less emphasis on them.

A cloud deployment was introduced in the role of the marina server, which allows anonymous docking and berth reservation underpinned by cutting-edge cryptographic solutions provided by the emmy library. The hardware was updated with an Intel Compute Stick which is an ultra-small form factor (USFF) PC, dramatically more capable than a Raspberry Pi, which allows us to run the full mF2C agent on it and, because of its integrated HDMI connector, presents new hardware deployment opportunities.

Throughout the development we have made improvements on our understanding of a user's perspective and needs when using the application. To improve the scenarios described above and to better comply with the modular aspect of the application's offering and the mF2C architecture, we have redesigned the list of scenarios by:

- merging Scenario 1 and Scenario 4 into Scenario 1 and
- introducing a new Scenario 4.

This allows us to better map to user expectations. The new scenarios are now:

1. Continuous boat monitoring and data sharing
2. Anomaly detection
3. Online docking and mooring reservation
4. LED light management

4.1.1. Current Hardware and Software

The list of hardware and software for IT1 was described in D5.3 [2], so Table 6 only lists differences for IT2.

A major change was the addition of new hardware as described in the previous section, along with an OpenStack virtual machine where the new marina backend module is deployed. Our general-purpose machines now have new components deployed on then: an LED controller to enable high-level management of the boat's lighting system, components that enable secure cryptographic communication via the emmy libraries, a messaging server supporting both chat and SOS distress message broadcasts and lastly a data backup and restore server for user convenience.

For a significant improvement in user and developer experience, we created extensive documentation served by our documentation server in both HTML and PDF format.

The SmartBoat software uses a modern CI pipeline through XLAB's internal GitLab instance. CI runners continuously build releases of the SmartBoat software with each change, enabling quicker turnaround

times and rapid responses to changing requirements. All development is handled through version control and utility CI/CD tools conforming with the latest software development standards.
There are two different kinds of deployment: one for the boat, which existed in IT1, and a new one for the marina, which is new. The SmartBoat marina application (and server side components) are separate from the main deployment on boats, as they serve the always-on purposes of the marina.

<table>
<tr><th colspan="2">Edge IoT devices</th></tr>
<tr><td colspan="2">Sentinel Boat Monitor (unchanged)</td></tr>
<tr><td colspan="2">Sentinel Hub (unchanged)</td></tr>
<tr><td colspan="2">Fog-capable devices</td></tr>
<tr><td colspan="2">Raspberry Pi 3</td></tr>
<tr><td>Hardware</td><td>Software</td></tr>
<tr><td>Unchanged.</td><td>Better LoRa communication support via a developer-friendly interface.
Much more flexible LED control.</td></tr>
<tr><td colspan="2">LoRa Module for RaspberryPI (unchanged)</td></tr>
<tr><td colspan="2">Sentinel router (unchanged)</td></tr>
<tr><td colspan="2">General-purpose machines</td></tr>
<tr><td>Hardware</td><td>Software</td></tr>
<tr><td>Intel NUC (unchanged)
Intel Compute Stick
● 4 Intel Atom CPU cores @ 1.44 GHz
● 2 GB RAM
● 32 GB eMMC
● WiFi, Bluetooth
Laptop (unchanged)</td><td>LED controller
emmy supporting components
messaging and SOS server
data backup and restore server</td></tr>
<tr><td colspan="2">Cloud-capable devices</td></tr>
<tr><td colspan="2">Cloud Virtual Machine</td></tr>
<tr><td>(Virtual) Hardware</td><td>Software</td></tr>
<tr><td>OpenStack virtual machine
● 8 GB RAM
● 4 CPU cores
● 40 GB root disk</td><td>Global marina cloud deployment.
Documentation server.</td></tr>
</table>

Table 6. Current hardware for Smart Boat use case

4.1.2.Software responsibilities

The data layer for the SmartBoat application consists of several databases and services. The PostgreSQL relational database holds all platform data apart from the sensor readings, which are for performance reasons stored in InfluxDB, a time series database. A key-value store, Redis, is used as a communication mechanism with the emmy cryptographic libraries. Finally, a MQTT messaging server is used as a support mechanism for LoRa communication when an appropriate transport protocol is not available.

Operations of the emmy security library are handled by three components: the registration server, the validation server and the proof initiator. The first functions as the credential issuer, the second as the validator and the last as a utility for clients to allow selective proofs of their credentials. Splitting components in this way allows the deployment to be flexible in terms of where the hardware is located, as only the necessary components can live on specifically designated machines.

The main part of the SmartBoat software stack is served by 12 modules. The sensor data reader module connects to the Sentinel Boat Monitor sensors via a proprietary protocol and forwards data to a reader, usually hosted on a different machine. Also interfacing with hardware are the LoRa communication server and LED controller, the latter of which is handled by a LED logic module, allowing for higher flexibility in LED lighting. These modules are depicted in Figure 15 and Figure 16, with a simplified and detailed component interaction diagram, respectively.

A data sharing server takes care of sharing sensor data with other boats, depending on the type of connection---whether LoRa or an IP stack---and similarly to that, the messaging component for communicating chat or distress messages. The alerting component is responsible for data analytics and generating alerts based on sensor data and the backup-restore component handles data exports.

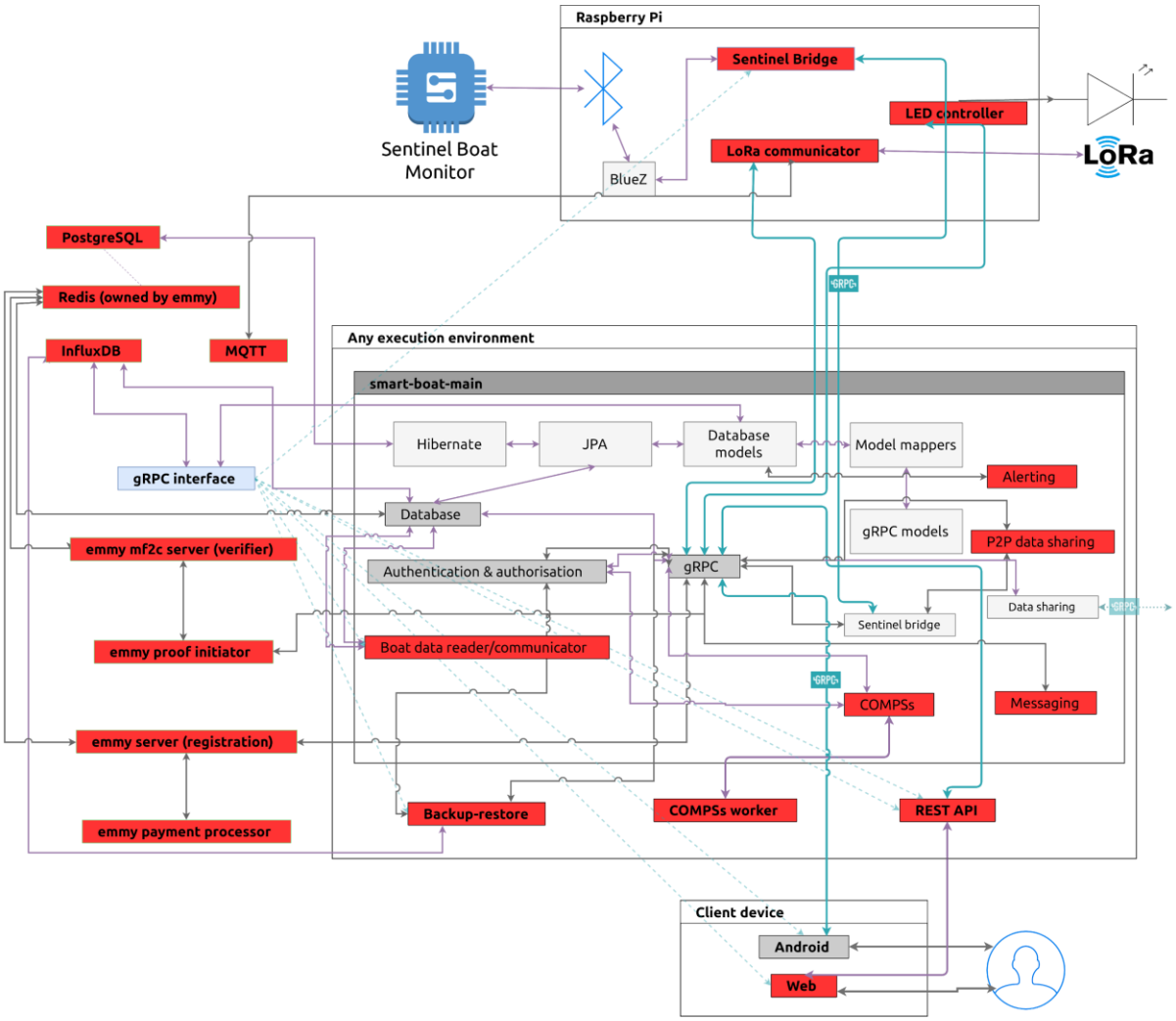

Figure 15 A simplified diagram of SmartBoat architectural and implementation components and their interactions.

Outward integration into mF2C is done by communicating through COMPSs, where a controller spawns worker component to perform computationally intensive work. These tasks control the

computationally-intensive tasks of calculating historical trends and other data used for display to the user and that can't be computed for every request by the controlling backend service itself because of time and latency constraints. COMPSs-powered tasks enable SmartBoat to distribute computation across multiple nodes, taking advantage of data locality to make processing more efficient, possibly even without contacting the cloud.

Other supporting components are the web and API servers used for browser-based access.

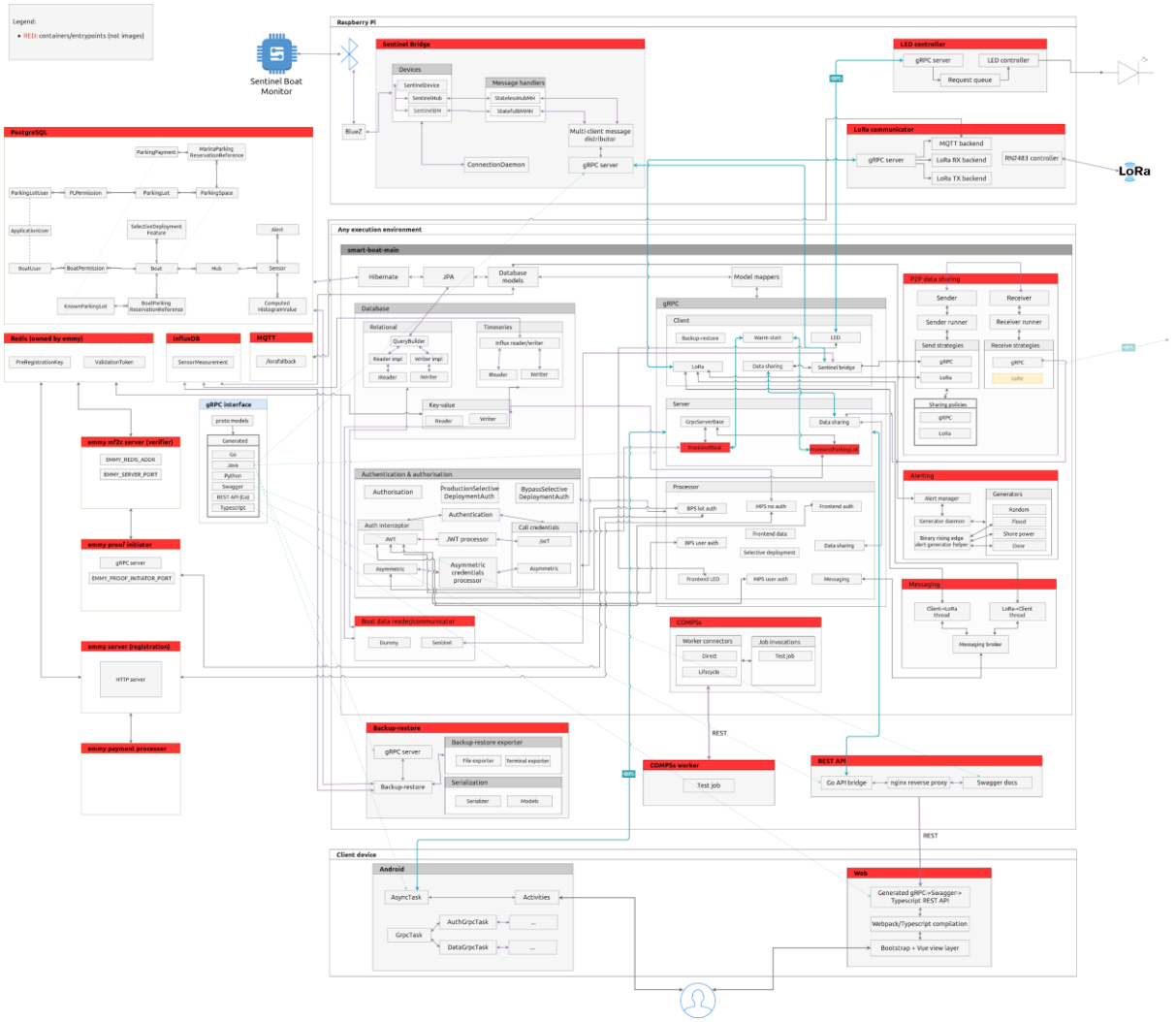

Figure 16 A detailed diagram of SmartBoat architectural and implementation components and their interactions.

4.1.3.Tasks to be executed by the mF2C agent

There are multiple tasks to be executed by the mF2C framework for the SmartBoat use case in IT2. The first, already present from IT1, is the deployment and usage of COMPSs. This is used to avoid transmitting all data points from the fog to mobile devices, improving application responsiveness and user experience. The COMPSs jobs are computationally intensive tasks spanning multiple nodes; this invokes the DER (Distributed Execution Runtime), which schedules bulk data processing tasks on the most appropriate nodes, which then work together to create instantly-accessible end-user reports about the condition of their boat.

A very important role for the mF2C agent is deploying SmartBoat application modules on demand. We have identified four modules of the application that can be presented to the user as optional addons:

distress signalling, chat, docking reservation and lighting control. These are enabled through the SmartBoat mobile application as seen in Figure 17.

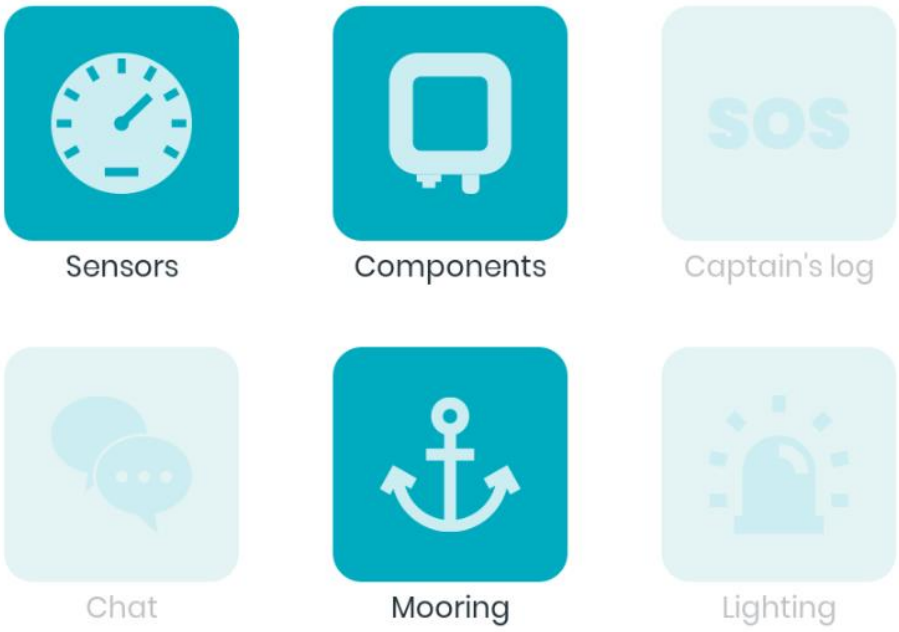

Figure 17 SmartBoat optional component toolkit, with the mooring reservation component enabled.

The user first deploys the base SmartBoat application through mF2C, which enables basic functionalities. Then, through the SmartBoat Android application, they can purchase and activate further functionalities. The action triggers mF2C to spawn new containers, thereby reducing developer workload as deployment is provided completely through mF2C.

Figure 18 Deployment of the SmartBoat server-side software stack through mF2C.

4.2. Use Case Storyline

Use case is targeted to cover the features that vessel market demands or by features that can present the possibilities which mF2C and digitalisation of the boat can bring to the market. To have a vivid sense of the target users and their interaction with the Smart Boat software, this section presents four storylines in a film script manner.

Story 01: *Continuous boat monitoring*

Thunder can be heard amid the raindrops loudly hitting the heliotropes outside. Bjørn, a recent owner of a SmartBoat system, is sitting in his dimly-lit lounge, browsing for holiday destinations he hasn't yet been to. No easy feat, considering all his expeditions in the last year. Greece? Too popular. Italy does not interest him, nor do the numerous scattered Spanish islands. Croatia has been a staple for many years, but a man can only spend so much time in small pearlescent inlets looking at bare islands before growing tired, wishing for more adventure in deeper waters.

Ding. An urgent-sounding chime emanates from his laptop's speakers. "*Extreme weather alert: Kotor, Montenegro*", from his weather application. *Damn*. That's where Bjørn's yacht is docked. Pulling out an uncharacteristically battered phone from his pocket, he frantically tries to unlock it. *"This is the time those newer toys would come in handy"* he berates himself for not being able to use a fingerprint. Finally managing to type in the PIN his daughter, Emmy, insisted on setting, he launches the SmartBoat application through a conveniently-placed shortcut she had created just days before.

Story 01: *Continuous boat monitoring*

The thundering outside grows louder, as if preempting his own tensing stomach.

Tap-tap. He logs in.
Tap. He selects his boat, noticing a new orange alert beside it.
Tap. He checks the sensors.

Scrolling down the list, past the battery voltage, motion alert sensors and the various weather sensors, his finger stops at the bilge water level. *"Whew"* he exhales, his shoulders relaxing, relieved that his boat seems to be okay. The bilge water level is within acceptable bounds and there is no alert associated with it. *"But what was the alert about?"* he ponders. Now with calm and collected movements, he discovers that his boat lost its LTE signal in the past hour. The SmartBoat system assuring him the network outage was intermittent, he returns his attention to the laptop, continuing to plan his next voyage.

Story 02: Remote control

A week has passed since the storm and Bjørn is speeding down the motorway in his freshly-detailed German all-electric SUV. Another one of his daughter's suggestions. Despite countless hours spent arguing about the purity and flexibility of internal combustion, he conceded and bought the two-and-a-half-ton machine he is now astounded by.

Recalling this, he arrives at his boat's marina. Greeting the owner, he takes his time walking in the morning sun, a breeze cooling his back. *"Let's try this out."* he thinks to himself, excited about trying out his SmartBoat. His gangway has been wired to the system, as well as the door locks.

Tap. The bridge starts extending. *"Huh. Neat."* he steps on the yacht instead of jumping the gap.
Tap. The lock on the cabin door clicks open. *"So this is what it's like."* he remembers unlocking the doors for the cleaning crew from his couch the day before.

Stowing away his snorkeling gear and readying his signature sunglasses, he unties the boat from the pier and sits in the captain's chair. The twin engines spring to life, ready to start the journey.

Story 03: LoRa distress call

A break on the open seas, with nothing but the sound of waves hitting the broadside of the boat. This is what Bjørn dreamed of when he bought the boat. He is far away enough from the shore that there is no mobile network coverage, so none of his business partners can call him for their self-proclaimed very important business meetings.

Story 03: LoRa distress call

Wheeeep. An alarm sounds from the phone in the quasi-glovebox at the captain's seat. *"This can't be a good sound,"* he hurriedly goes to grab his phone, thinking *"Is there something wrong with the boat?"* Upon turning on the screen, he sees an orange-coloured SOS notification from the SmartBoat application, saying that a nearby sailing boat has grounded.

Remembering how terrified he was nearly grounding his own yacht on last year's trip to Ibiza, Bjørn starts up the boat. He now always everything necessary for rescuing another from the situation. The SmartBoat application shows him the exact location of the other boat, so he is off immediately, notifying the passengers through the chat functionality that help is on the way.

"But how did I get an SOS when there is no coverage out here?" he begins pondering. *"Oh."* A thought springs to mind. His daughter helped him pick out the modules for the SmartBoat system. She kept ranting about the technological advancements and network range expansion of something called LoRa. *"This must be it."* he concludes.

After seeing the boat in need through his self-stabilising binoculars, he soon arrives by its side. With a "Well met, good friend!", greetings are exchanged and the sailing boat unleashed from the strong grip of the muddy ground. Bjørn got some new contacts and a few good bottles of wine for his assistance. Feeling satisfied after seeing the other boat scoot off, he returns to the bow.

Story 04: Online docking and anchoring reservation

The sun is beginning to set and Bjørn receives a message from Viktor, his childhood friend and frequent fishing buddy.

"Want to grab a few beers tonight?"
"Sure. Where are you?"
"Hvar, the city. Are you close?"
"Yeah. I can be there at 9."
"Should I make a reservation in the marina for you?"
"Wait, let me check something."

Bjørn opens up the SmartBoat app and enables the berth reservation module. In a moment, it is deployed to his boat and he can browse the list of marinas. He notices the Hvar marina in the list and makes a reservation for the night right away.

"It's done." he tells Viktor. "What's done?" "I've made a reservation myself." Bjørn explains, "I'm using SmartBoat and it's right there in the app." "Wow. Quick." sends Viktor, impressed.

The two agree on a time and Bjørn sets off, headed to Hvar.

Story 05: Remote Light Management

Cars on a nearby road faintly illuminate boats in the marina, interspersed with occasional flashes of daylight from an occasional driver with their high beam still turned on. Bjørn is having ice cream at the local promenade. Pistachio was always his favourite and he never understood what *blue sky* should taste like. Feeling adventurous, this time he went for the wasabi flavour. *"Wasn't wasabi supposed to be hot?"* he is surprised by the sweetness.

Finally, a message arrives from Viktor. Fashionably late, as always. "Hey, I'm at the marina, where's your boat?" he says, lost in the forest of masts.

"Here, I'll turn on the lights for you so you can find it." Bjørn replies, switching to the SmartBoat application and flicking on all exterior lights on the boat remotely. "I'll be there in a minute. Just have to finish my ice cream."

A yacht is illuminated in the distance. Viktor notices it: "Great, see you there!", leisurely beginning to walk in its direction, six-pack in hand.

4.3. Data Flow Diagrams

In addition to the data flow diagrams already present in D5.3, Section 5.3 [2], more functionalities and integrations into the mF2C framework have been exclusively developed for IT-2. These UC2 workflows are described in detail in this section.

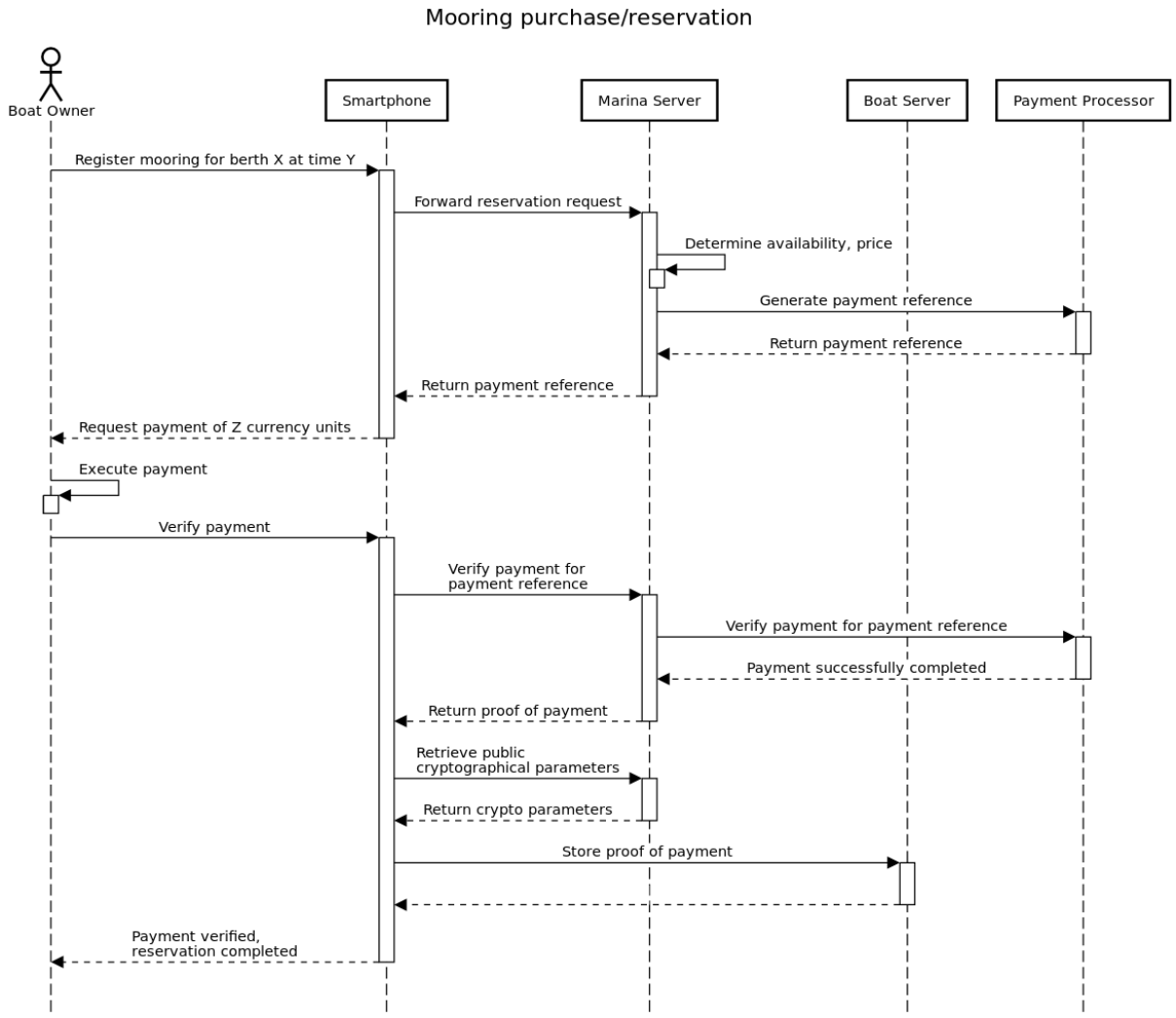

Figure 19 Mooring purchase/reservation sequence diagram

In Figure 19, the process of making a reservation for a berth is presented. The interaction begins with the boat owner requesting a reservation for a boat at a specific time. The smartphone processes this request by forwarding it to the server in the appropriate marina, which, in turn, determines its availability and price. If the berth is available at the time, the process proceeds. A payment reference is generated through an external payment processor, which is then returned to the marina server and, through the smartphone, to the user, which is requested to perform the payment.

Asynchronously, the user then pays the requested amount and later verifies payment with the marina. This is done by initiating the payment verification process through the smartphone for a reservation request. The reservation request stores a payment reference, which it forwards to the marina server for verification. The external payment processor is again contacted for verification, and, if the payment has been completed, the process continues by the SmartBoat marina server issuing a proof of payment and transmitting it to the SmartBoat application on the phone.

Cryptographical parameters of the marina are then retrieved by the smartphone to secure further validation communication. Finally, the boat is contacted to store the proof of payment on it in order to facilitate verification without requiring the physical presence of the boat owner. The user is then informed, through the SmartBoat application on their smartphone, that the process has finished successfully.

In this process, several operations regarding cryptographical security occur behind the scenes. In particular, the payment reference and proof of payment are entities of the emmy authentication and authorization scheme. They provide “parking slip” to the boat, the mere existence and possession of which is enough to exhibit permission to inhabit a berth, due to its cutting-edge security scheme.

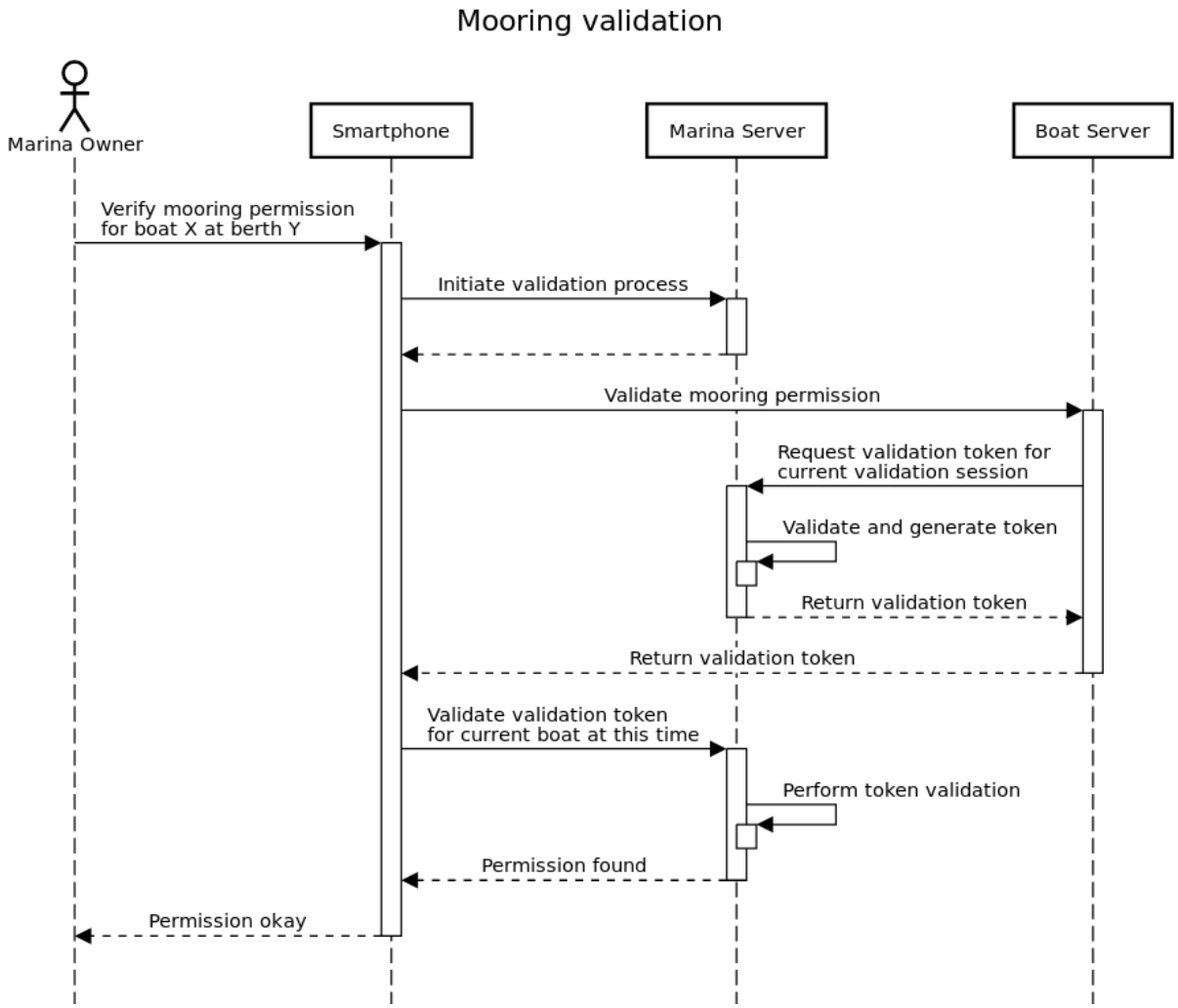

Figure 20 Mooring validation sequence diagram

Figure 20 shows a related process, after a mooring reservation from Figure 19 has been executed. This is the validation process, which is performed by the marina owner or one of their employees through the SmartBoat Marina smartphone application.

For the boat owner, the process requires no interaction or even physical presence, as all interaction and verification is done directly through the SmartBoat server on the boat. For the one validating, the marina employee, the process is exceedingly simple, as it only involves connecting to a boat and performing a one-click verification process.

The validator initiates the process through the smartphone application, which, behind the scenes, starts a validation session with the marina SmartBoat server. The validator’s SmartBoat smartphone application then connects to the boat server and asks it to verify itself.

The SmartBoat server on the boat requests a validation token for the validation session from the server in the marina, which validates cryptographical parameters and generates an emmy-based

validation token, which it then returns to the boat server, which, in turn, then forwards it to the mobile device of the validator.

The validator's smartphone then transparently connects to the marina server to verify information from the perspective of the marina, eliminating possible man-in-the-middle attacks performed by the SmartBoat boat server intercepting the validation token. The marina server is asked to validate the token for the specified boat at the current time. If mooring permission is found, the process succeeds by informing the validator that the boat has permission to reside at the berth.

Behind the scenes, again, several cryptographic operations are in play. To summarise them, the validator and their devices only receive the information that the boat has mooring permission and nothing else, which is, again, a cutting-edge solution to the privacy concerns a boat owner might have.

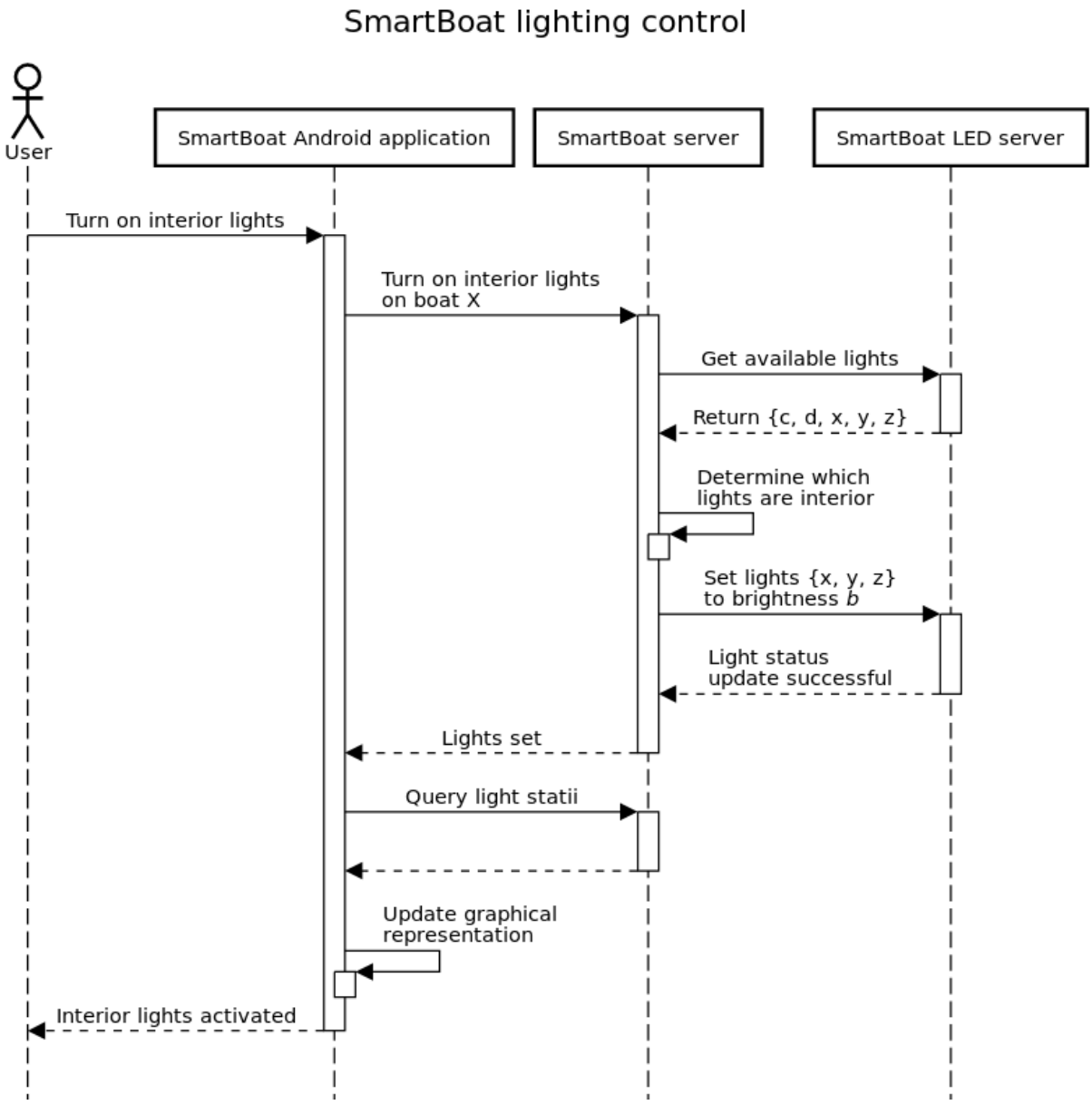

Figure 21 SmartBoat lighting control sequence diagram

A new module that was developed in the scope of IT-2 is the LED lighting control module. Its workflow presented in Figure 21, it allows a user to control their boat's interior and exterior lights remotely

through the SmartBoat mobile application, regardless of whether they are currently on the boat or in a remote location, only wanting to ensure that the lights are switched off.

The process of turning on the interior lights is, for the user, one toggle of a switch. The SmartBoat smartphone application requests to turn on interior lights to the SmartBoat server, which first retrieves the set of available lights from a backing LED control component. From an internal mapping, it then determines which of those lights are classified as interior lights and then issues another request to the backing LED controller to set that subset of lights to a specific brightness.

Returning to the smartphone application, it then queries the SmartBoat server for the updated state of boat lights and then updates the graphical representation on the phone to match the state of physical lights, making the information available even to a remote user.

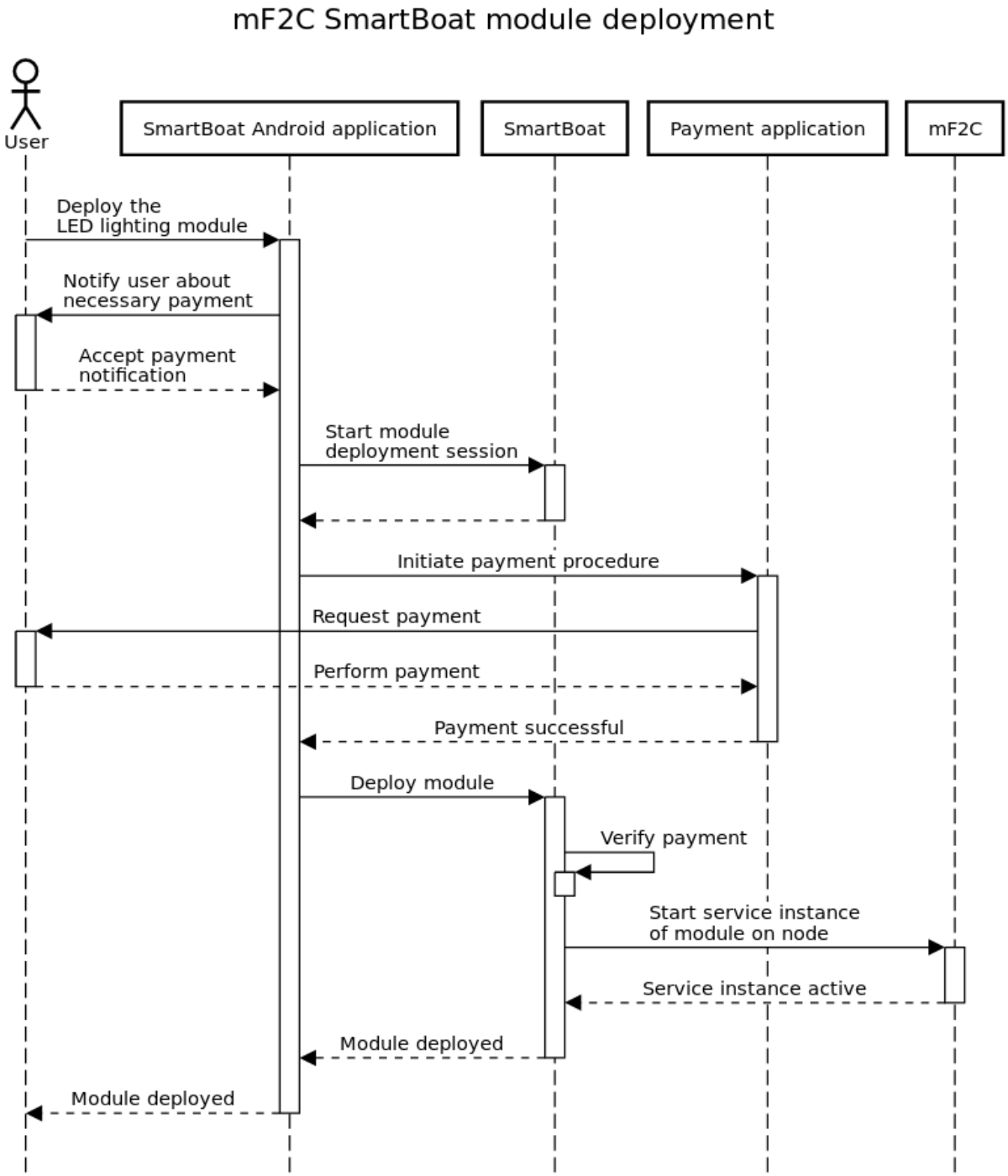

Figure 22 mF2C SmartBoat module deployment sequence diagram

A major benefit that the mF2C framework provides to the user is the capability to bootstrap and deploy the SmartBoat application itself and, additionally, deploy extension SmartBoat server side modules. Figure 22 details the new module purchase and deployment process.

When a user wishes to enable a new SmartBoat module on their SmartBoat server, they request the enablement through the SmartBoat smartphone application interface. The application informs the user that a payment is necessary, which the user then accepts.

The smartphone application starts a new module deployment session with the SmartBoat server and initiates the payment procedure with a payment application. This payment application is decoupled form the SmartBoat platform for flexibility. It requests payment from the user and, after the payment

is performed on the smartphone synchronously, informs the SmartBoat smartphone application of the fact.

Because the payment has been completed, the SmartBoat smartphone application can begin module deployment with the backing SmartBoat server and begin a loading animation to communicate the process to the user. The latter must first verify payment itself to prevent spoofing and, on success, contacts mF2C to start a new service instance on the node the boat is bound to.

After the service instance is deployed and active, the process finishes by informing the user that the new module is now available and ready for use.

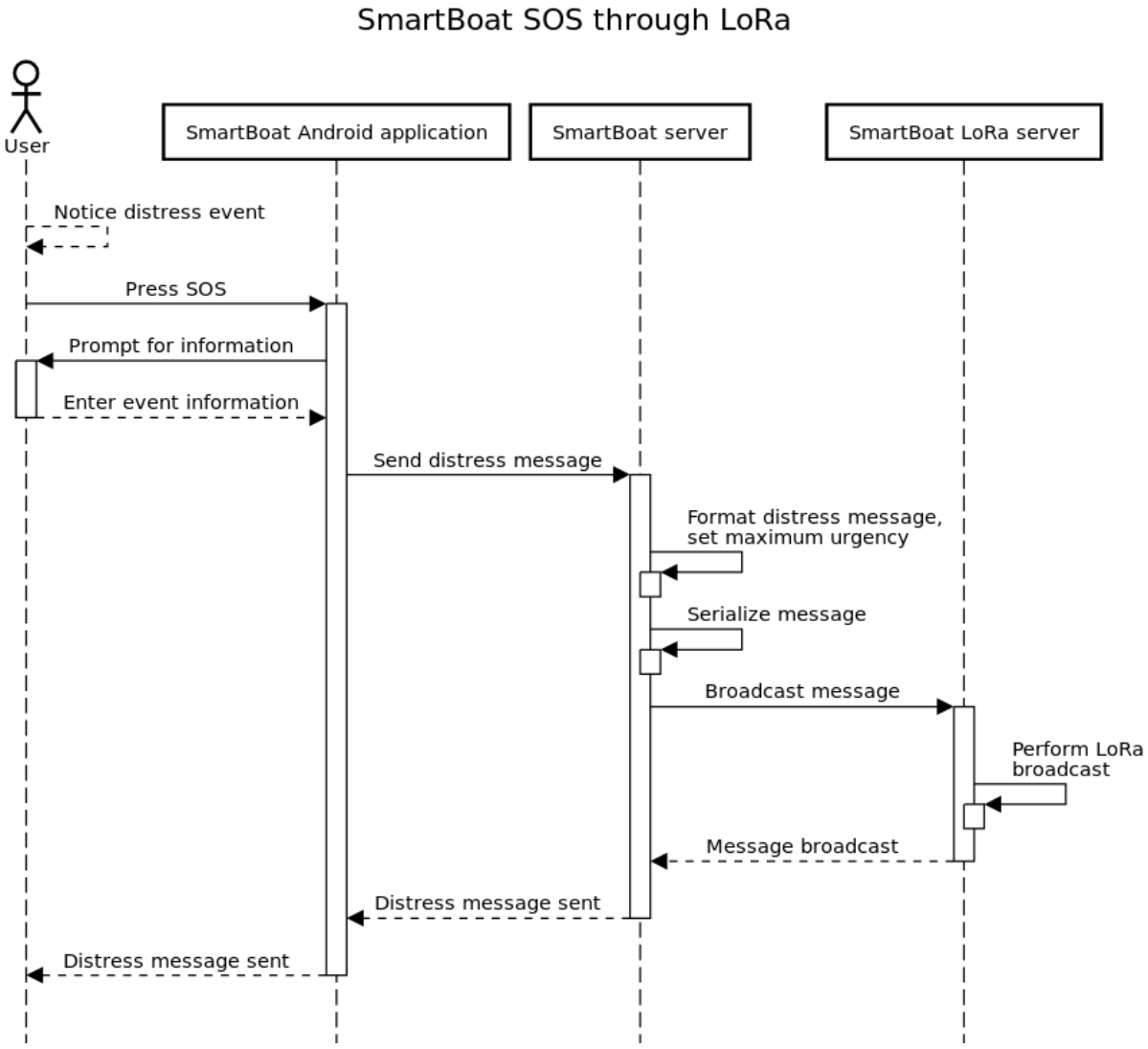

Figure 23 SmartBoat SOS through LoRa sequence diagram

Due to the utilisation of LoRa communication hardware in UC2, SmartBoat, an alternative to GMDSS, which is the ubiquitous system for maritime communication, has been developed, exclusively for distress calls. This allows even users not trained and certified to operate a GMDSS receiver to broadcast distress messages to boats in radio proximity, even tens of kilometres away. This process is presented in Figure 23.

When a user notices a distress event, they can press the SOS button in the SmartBoat smartphone application to initiate the distress message process. The application prompts the user for the minimum

necessary details, filling in machine-obtainable information by itself. It then contacts the SmartBoat server on the boat to send the distress message.
The SmartBoat server then formats the distress message, setting the maximum urgency, and finally serializing it to a wire format. It then contacts a backing LoRa server to perform a broadcast operation immediately.
After the message is broadcast, the user is informed that the message has been sent successfully.

4.4. Experimental Set Up

To test SmartBoat, UC2 of the mF2C project, we performed numerous experiments in various setups to verify and validate whether the system is fully operational and, furthermore, integrated with the mF2C framework. These tests were performed both in a lab environment and on-location, on a yacht. This was in order to have both the necessary flexibility in deployment scenarios and the ability to validate the technology on actual equipment, facing real-world challenges. Field testing was used when using positioning data and communication interfaces was paramount, while other tests, testing functionalities also easily demonstrated as a portable demo, were performed in a lab.

Figure 24 A portable SmartBoat device package with battery power for demo and testing purposes

The first line of tests are unit and integration tests performed through our GitLab Continuous Integration pipeline system. In total, 1818 test suites were initiated between all components described in Figure 24 that verified our compliance with functional requirements and guaranteed our targeted level of quality. A detailed chart is shown in Figure 25 The prevalence in tests in the Android and documentation repositories denotes our dedication to a complete and comprehensive user experience, as well as our ability to deploy everything through a smartphone.

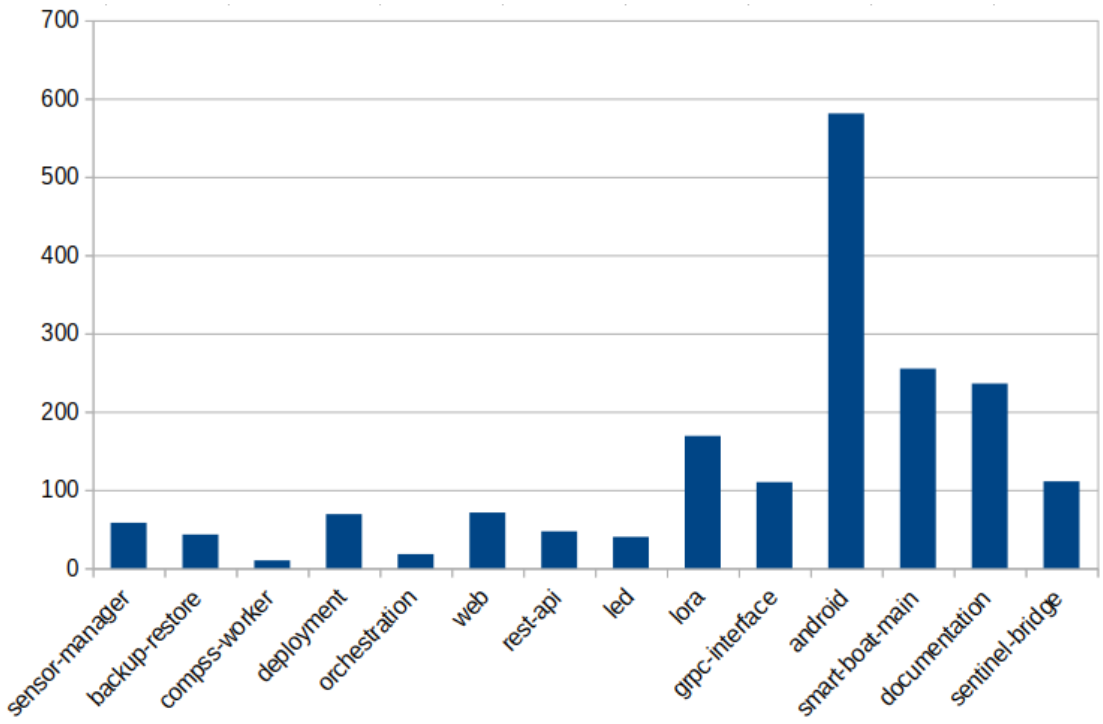

Figure 25 One of the SmartBoat testing tracks in the Croatian Adriatic Sea

The in-lab testing consisted of up to five sets of devices (Smart Boat Monitors, Raspberry Pis, etc), three sets of LoRa transceivers and NUC units, PC sticks and OpenStack VMs. A few sets of devices were powered by USB power banks that allowed freedom of movement to devices to the outside of the building and expanded the territory of in-lab testing. This heterogeneous setting allowed us to mimic a realistic Fog2Cloud environment for development and testing. Figure 24 presents one of the settings deployed in XLAB's offices.

Figure 26 shows a track the XLAB SmartBoat testing team took from Sukošan, Croatia on a working field trip where SmartBoat components were validated and a promotional video shot for exploitation and dissemination purposes.

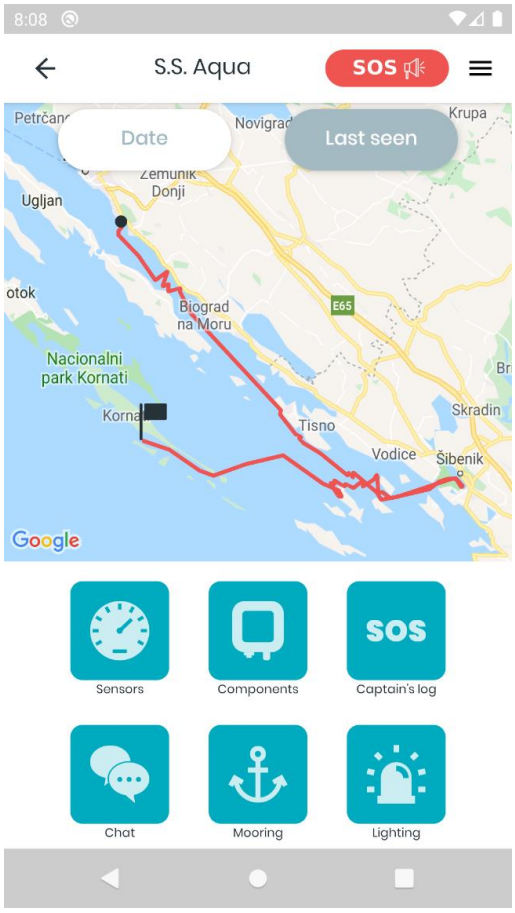

Figure 26 One of the SmartBoat testing tracks in the Croatian Adriatic Sea

This section describes the most common testing configuration, a variation of which will also be presented in the final demo.

The list of devices and equipment used for a set-up consisting of a single boat is:

- Sentinel Router; used to provide local connectivity between devices and clients,
- Sentinel Boat Monitor; used as a main proprietary maritime sensor hub,
- Sentinel Hub; used to extend sensing capabilities to remote areas,
- Intel NUC; used to deploy an mF2C agent and the main SmartBoat components,
- Raspberry Pi; used to connect to sensors and control actuators and luminaires and also to deploy an mF2C microagent,
- the RN2483 LoRa module, attached to the Raspberry Pi through a hat,
- a 16750 mAh power bank; used to provide battery backup to the Sentinel Router and Raspberry Pi for increased stability and resilience,
- a smartphone; used as a client accessing the system and
- a virtual machine in The Cloud; used as a central root of the SmartBoat system.

The Sentinel Router is a central communication point for most devices. The Intel NUC and Raspberry Pi are connected to it via a wired connection for stability, but we have also successfully tested, validated and experimented with a wireless connection for even more flexibility. The Sentinel Router and the Raspberry Pi are also provided with battery backup power through a significantly durable power bank.

Other Sentinel devices, the Boat Monitor and the Hub, which are based on Bluetooth, are connected logically to the Raspberry Pi running the SmartBoat sensor module via Bluetooth connections. The smartphone connects to the WiFi network provided by the Sentinel Router, thereby providing local

connection to the local agent on the boat and remote connection to the cloud through the uplink provided by the same device.

Another aspect of testing was testing LoRa and its reliability and range. The testing showed the effects of different obstacles on the propagation and disruption of signal and were performed in a suburban environment with a mix of green and paved surfaces. Scenarios can be divided into two main groups: clear line-of-sight and obstructed view. The hardware used in the tests are Raspberry Pis and their LoRa expansion modules, as well as battery power for portability. Sentinel Boat Monitors were used as a source of positioning data.

Raspberry Pis with connected Boat Monitors were set in the following way: one stationary, on a windowsill in the main XLAB Research office and the other one mobile, moving between predetermined points.

Both devices log message sequential numbers and start and end times of transmissions/receptions. In addition to this, the transmitter and receiver also log device specific parameters: the number of retransmissions and signal-to-noise ratio. The transmitter logs every packet it sends, each one containing a sequential number. When the receiving device gets a packet, it stores data using the sequential number contained inside of it. By comparing sequential numbers from logs of both devices we could easily see number of lost and successfully received messages. DSata is then analysed and plots are generated by a dedicated analyzer script.

4.5. Results and KPI measurements

To measure the main benefit of utilising a fog-based instead of a custom-built cloud-focused architecture, we deployed a laboratory environment with the SmartBoat application on an Intel NUC production-grade machine serving as both the fog and/or cloud to achieve consistency between tests and to eliminate possible differences between the processing power of physical and virtual machines. Sensor data was simulated for consistency across multiple runs and to eliminate other anomalies irrelevant to the tests at hand. A mobile phone was used to access the SmartBoat system. The test was executed in two phases:

A. testing access through the fog
B. testing access through the cloud.

In phase A, the mobile phone was connected to a WiFi network local to the network the Intel NUC machine, running the SmartBoat software on the mF2C agent. This accurately simulates the scenario whereby a user connects to their own boat without a connection to the cloud.

In phase B, the mobile phone was not connected to the WiFi network but instead used a 4G connection. The SmartBoat software running on the mF2C agent was accessible through an external production-grade gateway, simulating the scenario where a user has a mobile connection at sea and is accessing historical data not available on a boat.

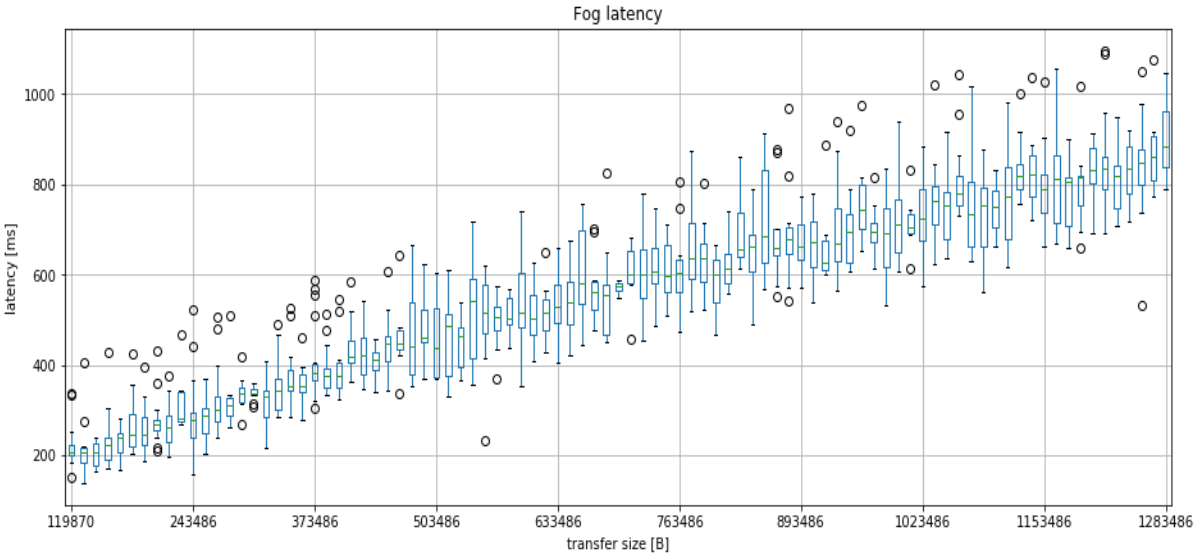

Figure 27 Fog request latency with respect to response size

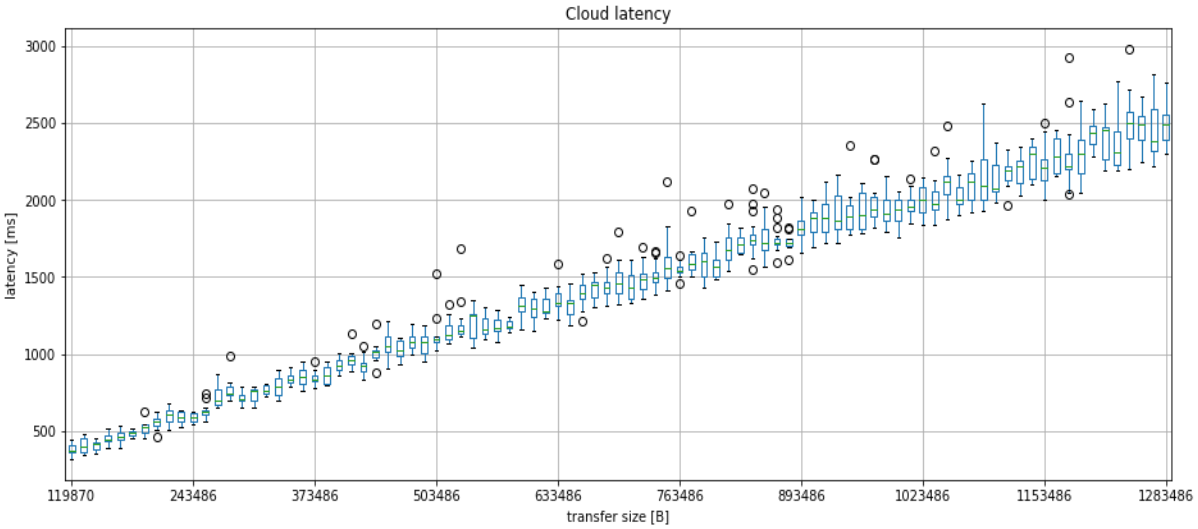

Figure 28 Cloud request latency with respect to response size

Figures 27 and 28 show detailed data of the complete request latency, as the user would experience, with respect to the request size. This corresponds to the amount of data transferred when e.g. transferring sensor data to the device for local display.

There is a very notable and consistent 2.5x improvement when accessing the same data, the only difference being the method of access: local network access for the fog and mobile network access through the edge. This difference is more notable with higher transfer sizes, as clearly shown in Figure 29.

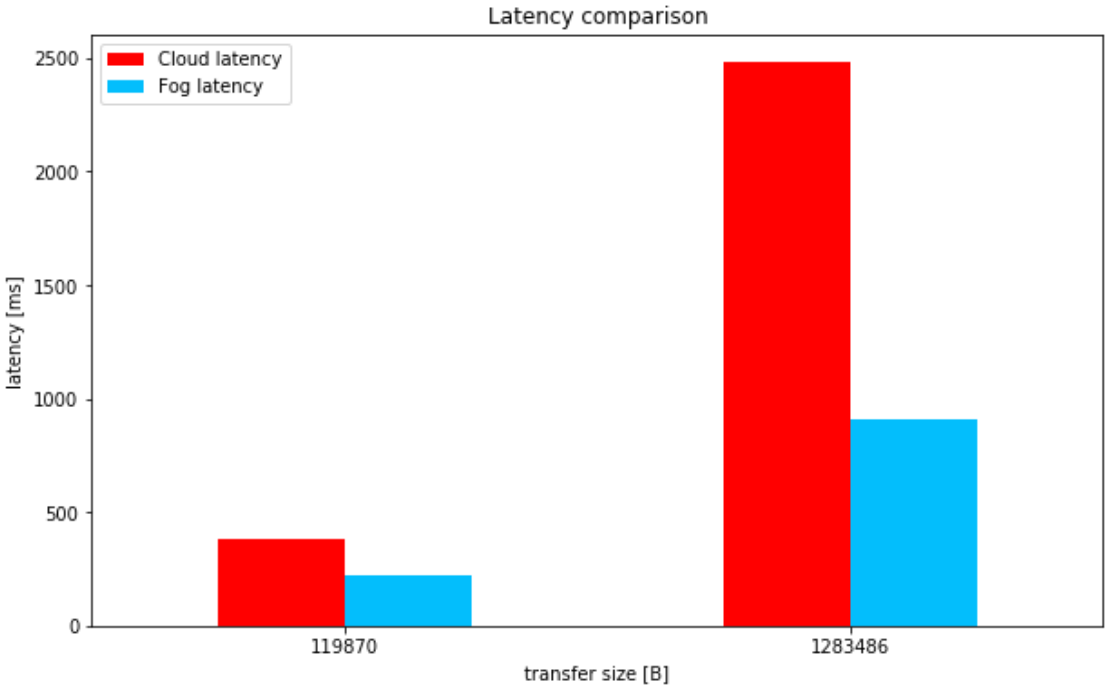

Figure 29 Differences between latencies in small and large payload scenarios

These results can be surmised into the fog-to-cloud paradigm, where requests are intelligently rerouted to the most appropriate service point. If data exists in the fog, a measurable 2.5x improvement in response time was achieved, thanks to the fog deployment capabilities offered by the mF2C system. When also taking into account processing time, the difference can be lesser because machines in the cloud may be more powerful. However, considering non-intensive computations for workloads other than, e.g. machine learning, fog devices mostly show no difference to their cloud counterparts when taking into account latency, which constitutes the majority portion of the request latency.

When developing the SmartBoat application, the development team paid special attention to the work required to integrate and use the mF2C system. The deployment feature that the mF2C framework provides was considered paramount to the usage and integration and replaced efforts to create a homegrown distributed development platform. It is estimated that 15-25 % of time was saved by using the existing deployment feature compared to developing a solution in-house.

Results of the LoRa range and performance testing were measured on a track near the XLAB global headquarters. Two testing routes were used: one nearby and another somewhat far away, looking at the transmitter and receiver, both shown in Figures 30 and 31.

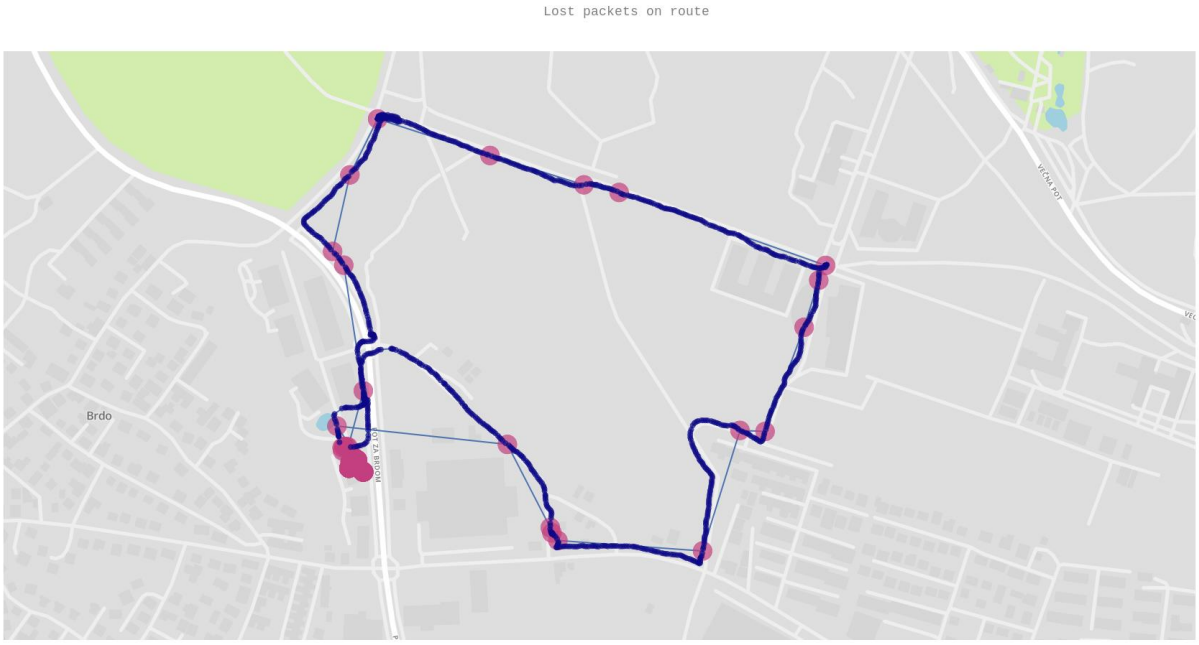

Figure 30 The nearby LoRa testing route

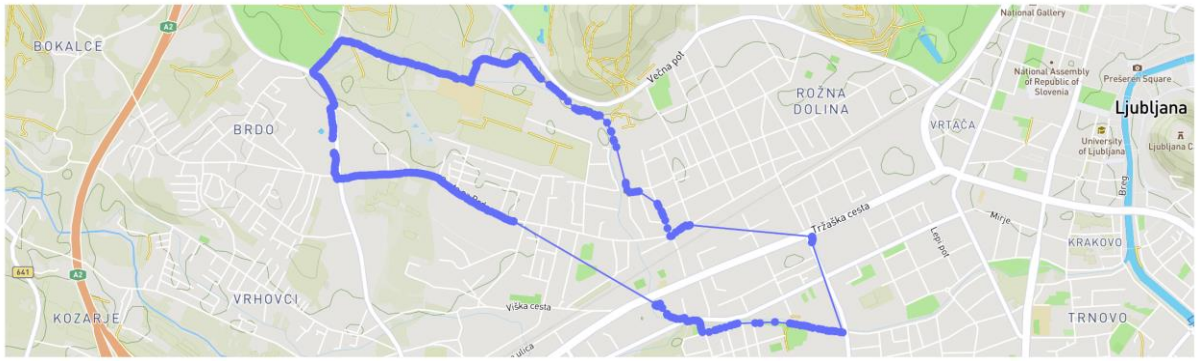

Figure 31 The further LoRa testing route

Since testing on distances under 800m produced much better results than initially predicted and had not given us any idea about the limits of LoRa, we focused on the second, longer-range test. This route contains distances around 2km, which is often mentioned as limit for reliable LoRa transmissions in urban areas.

When looking at the results of this test we observe much higher packet loss; over 30%. This is suggesting that we were much closer to the limits of LoRa. Signal-to-noise ratio had increased significantly. We can easily see much higher distortion of the data at greater distances. Here we also have to point out much higher density of urban structures compared to previous testing. This could be an important factor, because it limits signal propagation ability drastically.

We can observe the majority of the packets on distances above 1.6km were lost. This shows the importance of space to propagate signal even better. All the southernmost readings had much better success rate - this is due to the structure of the streets testing was concluded on. These were very open, without any high buildings in direct line-of-sight to the receiver device in XLAB offices. The easternmost part of the test with minor success was, in contrast to that, concluded in very densely built streets - this resulted in majority of the packets being lost.

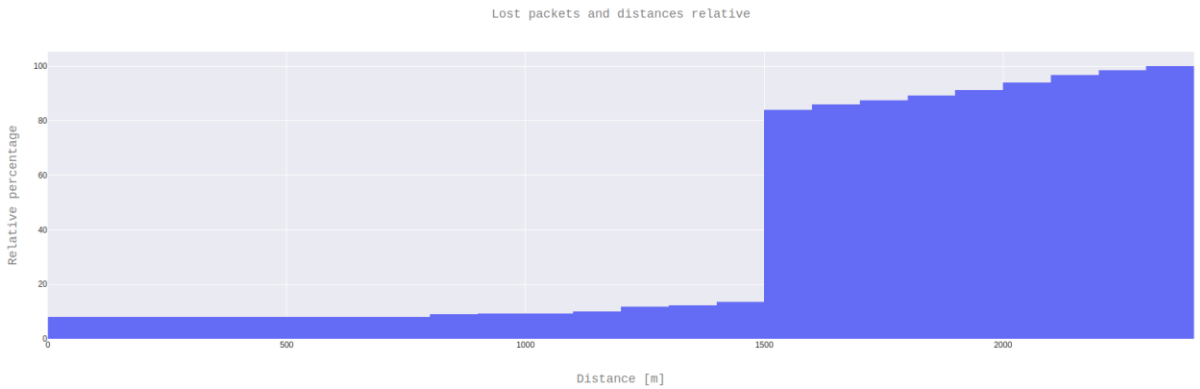

Figure 32 Lost LoRa packets with respect to transmission distance

Breaking down the packet loss with consideration of the distance, we can see the great majority of the packets being lost at 1500-1600 metres. In conclusion, this test showed relatively reliable LoRa transmissions on distances under 1600m and the importance of space to propagate signal. If the needed space if provided---e.g. at sea, we could observe a reliable connection at significantly larger distances.

4.6. Business Prospective and conclusions

Through conversations with strategic business partners, we have determined that SmartBoat includes several cutting-edge functionalities interesting for the market, but requires more polish and product, instead of research, focus. Significantly, the LoRa communication feature and the feature for mooring reservation via the emmy library were pointed out to be very interesting, but present several challenges related to the maturity of the market and the necessary adoption ratio and market penetration for features to be usable. However, the technology can be used for other scenarios, and having this in mind the demonstration with a prototype on a real use-case scenario has an important impact on the potential users and investors.

For further business development, SmartBoat will endeavour to adapt its planned business model to suit current market conditions. Working with progressive marinas and other early adopters of cutting-edge technology, as well as existing strategic partners in the field will undoubtedly prove paramount to achieving a high level of success for both exploitation and future development.

The mF2C framework has shown to provide useful functionalities in the creation of a fog area, connecting devices and providing dynamic connectivity to participating devices, as well as a deployment mechanism suited for the use in deploying the SmartBoat system.

5. Use Case #3: Smart Fog-Hub Service (SFHS)

5.1. Complete Architecture Use Case

The following diagram represents the final architecture of the Smart Fog Hub Service Use Case, for IT-2.

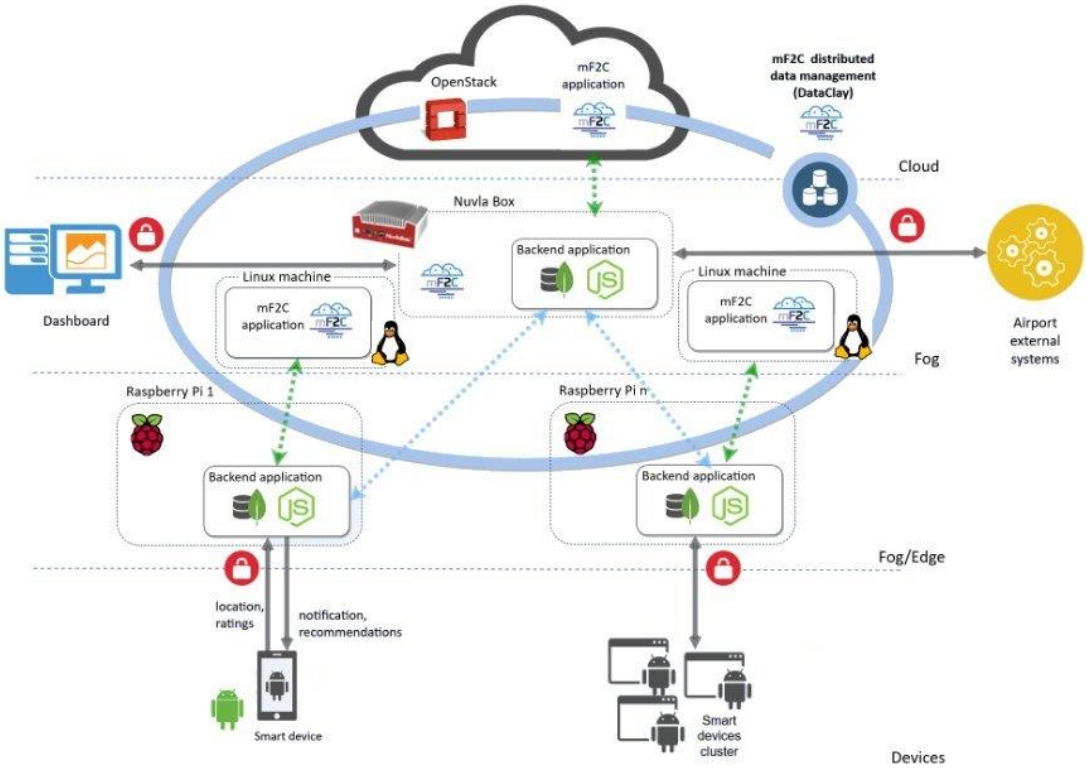

Figure 33 UC3 final system architecture

This platform aims to allow the end user a friendly and interactive experience during the stay and moving in the airport field, using an Android app specifically designed and developed for this purpose. The main functionalities are:

1) Proximity computation of POI and notification delivery to the user. The notification can include a brief message, for example a promotion or discount for a shop, or other useful information. The proximity range is defined for each single POI, thus allowing end user notifications even for relative long distances in the coverage area. The possibility to configure this range could fulfil specific business-driven requirements for service providers. For example, it could be used by a shop that wants to reach the largest possible audience for its advertisement, even in a given time slot.
2) Subscription for notifications on the traveller flight. The traveller can choose in the app his/her flight and be notified for status changes, like departure time, gate assignment or change, delay, etc. The information about the flights are taken from the external information systems of the airport.
3) Subscription for notifications on specific topics of interest to the user. This feature could be particularly useful for notifying the user of information or events of interest are outside the airport, in places that the user wishes to reach or visit. For example, the user can be notified about the status of city transports, folklore events, movies programming at the cinema etc.

4) Rating tool for the POI. Through the app, the end user can see the information for a given POI and vote his/her feedback (a simple score from 1 to 5 stars). Currently it is not possible for the end user to post a comment, but this feature can be developed in the future enabling a proper user identification.
5) Topics Advisor fed by historical end user ratings. This feature, based on Machine Learning algorithms, provides a list of suggested POIs that could be of interest to the user.
6) Administrative dashboard for system and user behaviour monitoring. It includes more tools and minor improvements. This software, developed for demo purposes, can be further improved taking into consideration business-oriented suggestions provided by the service providers.

In IT-1 we have already demonstrated functionalities for 1 and partially for 2, 3 and 6, while 4 and 5 are new functionalities. All software is deployed as docker containers to improve development, operations and maintenance, achieving a high level of reliability and stability.
The UI of the Android app has been totally redesigned and improved, offering a better and smooth end user experience.

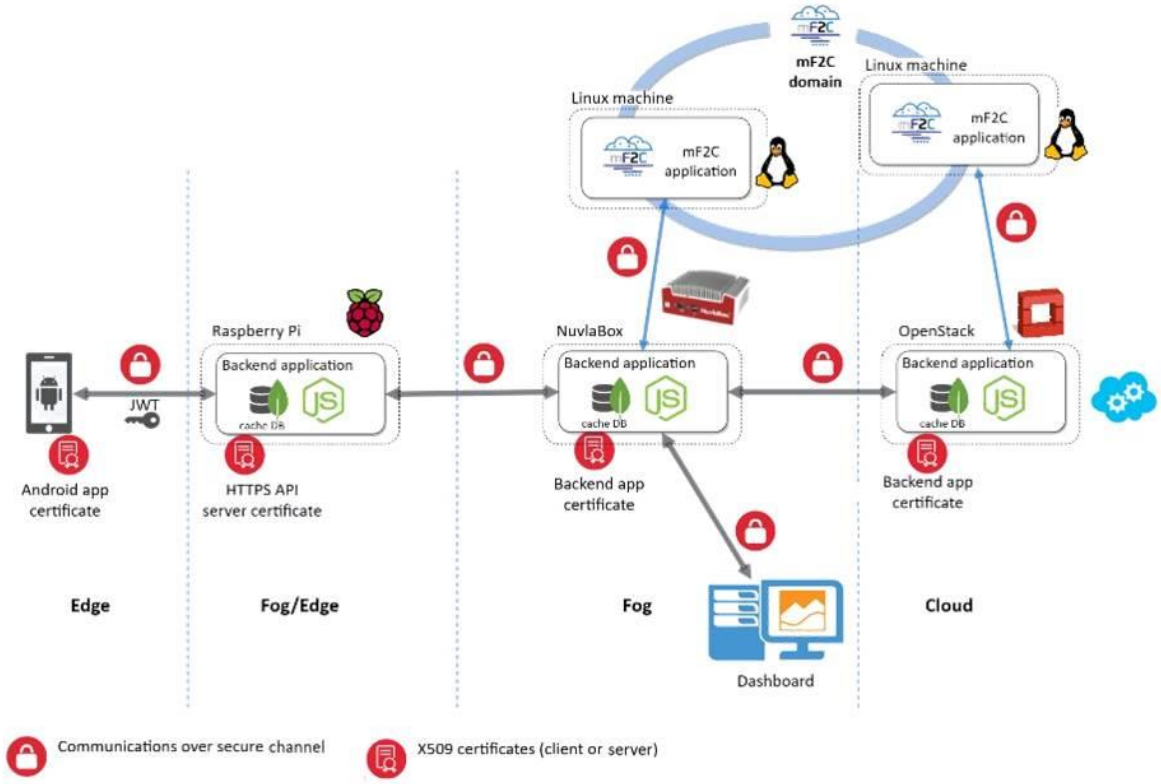

Figure 34 UC3 security architecture

Particular care has been taken to the security aspects of the whole platform, both in the architecture and in the developed software, thus a custom PKI is used to fully support them. All communications between different software components use certificates signed by a custom root CA, so communications are tamper-proof. In particular, the communication between Android app and the API server uses both client and server certificates for mutual recognizing. Moreover, the Android app uses a JWT (JSON web token) to identify and recognize the device session and to access the API server functionalities.

5.1.1.Current Hardware and base Software

The hardware and software for IT-2 aims to fully support the functionalities described above, offering suitable performance, stability and reliability.

The devices that compose the platform architecture, depicted in Table 4, are:

- A set of RaspberryPi, which act as wi-fi access point for the end user smartphones with the specific Android app installed. A single RaspberryPi hosts the https REST API server that provides all the functionalities needed by the Android app; moreover, it keeps trace in a local database the end user device sessions together with other needed data. In this way, each RaspberryPi handles its own cluster of end user devices. All end user data collected are synchronized with the backend software in order to make them available to the upper layer on the software stack. The software handles automatically the handover in case that the end user device switches between different platform access points.
- A SixSq Nuvlabox mini, which hosts the backend software which collects in a local database all data coming from the lower software stack, which basically are end user session data. This database is the main point of view of all user data collected in real time in the field and is accessed by the administration dashboard; moreover, it stores and handles all information needed for proper end user notification, e.g. flight status changes, advertisement for shops, topics related messages, etc. This set of information is properly delivered to the lower layer of the software in order to make them available to the end users in their devices. Furthermore, this device hosts another software component, which is in charge to query the airport API that provides real time flights status data, and store them in the local database.
- A set of miniPC x86 devices, which host the mF2C agent and services in the fog layer.
- A cloud VM in an OpenStack instance, which hosts mF2C cloud agent and service, and another software component in charge to store the anonymized historical end user data useful for long term statistics, in a local database.

The following table lists all devices and installed software:

Edge IoT devices	
Smartphone used by end-users	
Hardware	Software
ASUS ZENFONE 4 MAX CPU: Qualcomm® Snapdragon™ 430 Mobile Platform with 64-bit Octa-core Processor GPU: Qualcomm® Adreno™ 505 (SD430) Memory: LPDDR3 3GB Internal storage: eMCP 32GB Display: 5.5-inch IPS Main Rear Camera 13 Mpix, Front Camera 8 Mpix Wireless Technology: WLAN 802.11 b/g/n; Bluetooth 4.1; Wi-Fi direct Sensor : Front fingerprint sensor, Accelerator, E-Compass, Gyroscope, Proximity sensor, Ambient light Navigation : GPS, AGPS, GLO, BDS	Operating system: Android Nougat (v7) Custom app for user engagement
RaspberryPi3 Model B 1.2	
Hardware	Software
Broadcom BCM2837 SoC, with quad-core ARM Cortex-A53 1200 MHz processor VideoCore IV dual-core 400 MHz GPU	Operating System: Raspbian Jessie Lite Docker, version 18.03.0-ce Container A: MongoDB v3.2.18 32bit

1 GB SDRAM - shared by the GPU and CPU MicroSD card slot for boot and storage 4 x USB 2.0 ports (via on-board 5 port hub) RJ45 10/100 MBit/s Ethernet port HDMI and Composite video, CSI (camera) and DSI (display) connectors	Container B: API server for end-user session management, based on Node.js v9.10.1 and several npm modules: Express, Mongoose, Socket.io, etc.
L2 Fog-capable devices	
HP Laptop	
Hardware	Software
ibm thinkpad t420i model 4236G90 intel i3-2350m cpu 2.30GHz 4G ram + 4G swap space 160G HD	Ubuntu 18.04 - CentOS 7.4 mF2C agent, mF2C service application
L1 Fog-capable devices	
Nuvlabox Mini	
Hardware	Software
Intel Celeron processor 8 GB RAM 128 GB SSD 60 W external power supply WiFi 2 USB 2.0; 2 USB 3.0 VGA / HDMI 2 x RJ-45 network interface	Centos 7.1 CloudLayer: OpenNebula Virtualization: Kvm AppStore: SlipStream CentOS Linux release 7.4.1708 (Core) Docker 18.03.1-ce - Docker Compose 1.21.2 Container A: MongoDB v3.6.0 Container B: Backend server based on Node.js v8.9.2, and several npm modules: Express, Mongoose, Socket.io, etc.
L0 Cloud-capable devices	
Cloud Virtual Machine	
(Virtual) Hardware	Software
VM with 2 vCpu, 16G RAM 100G storage	Centos 7.4 MF2C Agent Application for data collection

Table 5. Current hardware and base software in UC3

5.1.2.Software Responsibilities

The full stack of software components in the use case consists of eight modules. It is slightly different from IT-1 according to new features and optimization. The following list describes the components and the functionalities.

- API server (with SSL), hosted in the RaspberryPi.
  1. Provides API endpoints for the Android app installed on user devices
  2. Forwards device/user position data to the mF2C service, which is used for real-time proximity computation
  3. Stores user session data in the local database and forward them to the upper layer backend module (the fog backend module), in order to synchronize data between

databases. This synchronization is performed by specific tasks, which runs periodically at configured interval time.
4. Receives notifications messages from the upper layer backend module and make them available to end users' devices by using proper API endpoints. These notifications messages can be related to POI or specific topics or can be an update of the status of the traveller's flight.

- Fog Backend module, hosted in Nuvlabox mini
  1. Receives device position and other user session data from the lower layer API server through synchronization tasks and stores them in the local database. Thus, this database contains all the sessions data provided by all the user devices in the field, each one connected to its own access point.
  2. Stores the updates for the flight status in the local database, supplied by another specific module, and forwards them to the lower layer API server through specific synchronization tasks, which runs periodically at configured interval time.
  3. Stores notification messages related to POI and topics in the local database and forwards them to the lower layer API server through specific synchronization tasks, which runs periodically at configured interval time.
  4. Stores statistics data like user movement tracking in the local database, making them available to the Administration Dashboard module. These data are forwarded to the upper Cloud backend module for long-term storage and analysis, through specific synchronization tasks.
- Flights status update module, hosted in Nuvlabox mini
  1. Query the airport API for real time flight status
  2. Updates the status of the flights stored in the local database. The database is the same used by the fog backend module.
- Administration dashboard, hosted in Nuvlabox mini
  1. Get data to analyze and display from the local database. The database is the same used by the fog backend module
  2. Provides a web GUI for system administrators and field operators
- Cloud backend module, hosted in the cloud VM instance
  1. Provides long-term storage for users' sessions and other useful historical data ready for statistics. All stored data come from the lower fog layer and are properly anonymized for privacy compliance.
- Topics Advisor, hosted in the cloud VM instance as mF2C service
  1. Provides the engine for Collaborative Filtering algorithm used for end user recommendations for POI. The data used for computation are taken from the local database. This engine runs periodically in batch mode, the timing can be configured as needed by the system administrators according to the volume of data produced in the field.
  2. Stores the result in the local database, available for the lower layers and ready to be queried on-demand by the end user Android app.
- mF2C agent and services, hosted in miniPC and cloud VM instance
  1. Provide the deploy of mF2C related software components
  2. Provide the software stack for specific functionalities detailed above.
- Android app, installed on end user devices.
  1. Provides the UI and all functionalities for the end user, connecting with the API server.

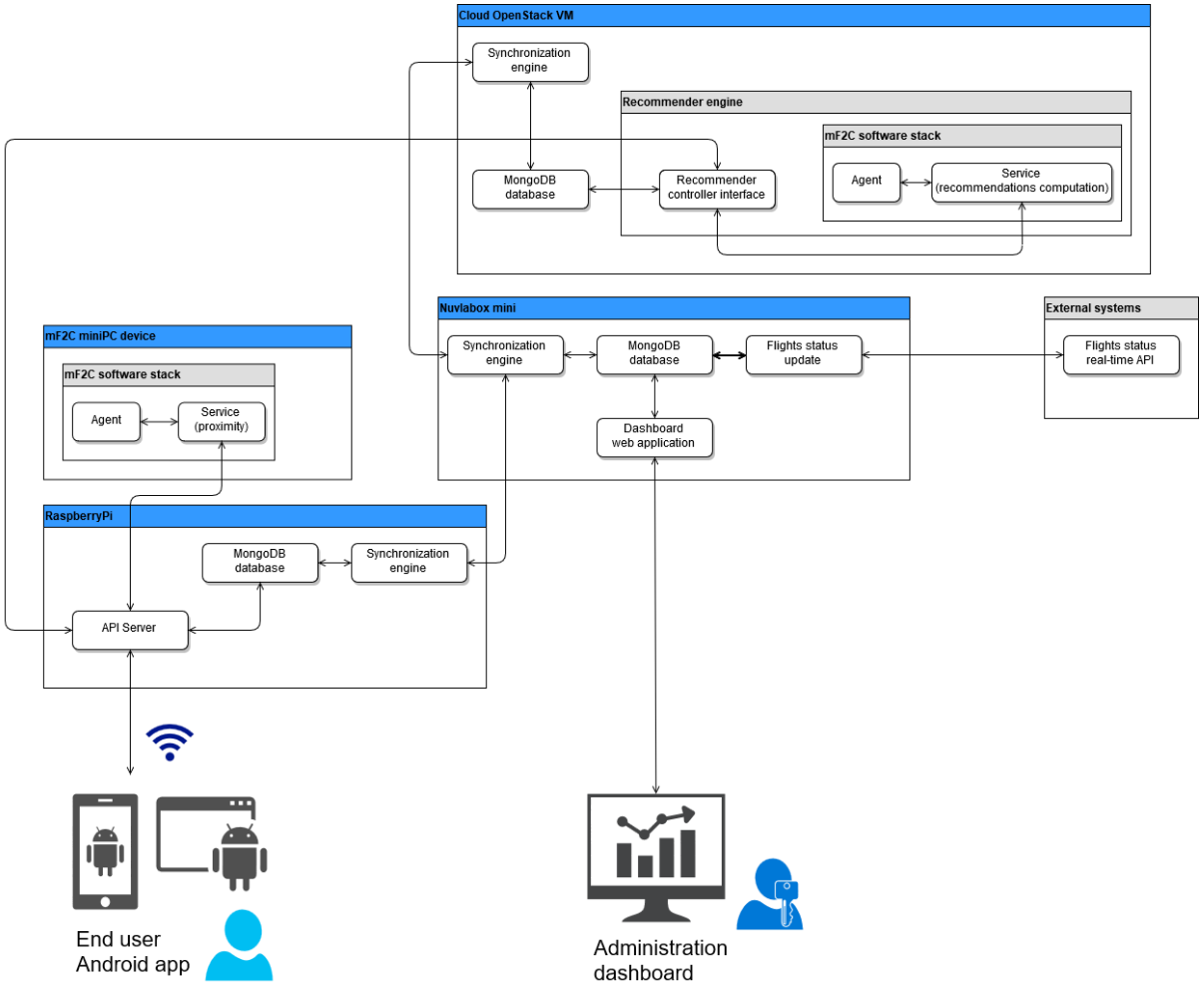

Figure 35 Privacy Architectural hardware and software components

As shown in figure 35, the data layer in the platform consists of several local databases instances, which are synchronized both ways, between the upper (backends hosted in Nuvlabox and Cloud) and lower (edge/API server in the RaspberryPi). This approach, already developed in IT-1, assures the delivery of the data when and where needed, and emphasizes the data proximity concept. All databases are instances of MongoDB.

As described in other deliverables, the mF2C agent uses internally dataClay for its data communications and sharing.

5.1.3.Tasks to be executed by the mF2C Agent

In this use case, there are two main tasks deployed as mF2C services, and thus managed by the agent, as described below:

- Proximity computation, given the end user position in the field. Each position update is generated by the end user movement in the field, sent to the API server and stored in the local database. The position is then compared with the list of POI in the field and the computation result is the list of nearby POIs. Nearby means, as mentioned before, that the user is located inside the “coverage area” of the specific POI, and this can be configured individually for each POI according with business strategies. For example, the service provider can offer a pricing option for larger coverage areas, giving the POI’s owner the chance to promote it to a larger audience.
- Recommendation of Points of Interest (POI). This task uses Machine Learning algorithm (Collaborative Filtering) to propose a custom list for each user, containing the POI that similar

users like most. The concept of similarity is based on previous user preferences and feedback for the POI, thus if a user has not yet given any feedback, the system propose the plain list of the most preferred POI by all the users, considering the whole dataset of ratings.

The adoption of the mF2C system gives two main benefits. First, it is possible to handle both up and down scaling, according to the number of simultaneous users to manage. When the number of users grows, the system administrators can easily put in place additional devices in the fog layer, install the mF2C agent and deploy services in them, in order to support efficiently the larger load in processing data. At the opposite, if the number of users decreases, the system administrators can decide to dismiss some resources, simply by stopping services and/or get out devices.

In this use case scenario, the number of users can be easily gotten by airport statistics but also by analysing user tracking long-term data stored in the cloud layer of the platform.

Another clear benefit of the mF2C system is that it is able to manage more than one fog areas in more complex scenarios like smart cities (e.g. a fog area in the airport, another in the train station, another in specific tourist places of interest in the city, etc.). Doing this it is possible to develop and offer a complex set of services to end users who subscribes to the system. These services could be free or paid, even managed by different service providers, but finally handled by mF2C services. Moreover, being mF2C agent able to proper manage resources and services in the whole set of available devices, they can be used effectively and optimized for service execution.

This could and should be done taking into account the privacy of the user data, which is always a strict requirement, by leveraging built-in mF2C security features.

5.2. Use Case Storyline

The user experience starts when the traveller completes the security check and reach the departure pier with all gates. He/she follow the instructions and download and install the airport app in his/her Android smartphone.

Once the app has been started the first mandatory step is related to reading and accepting the privacy terms, in this case the shown information explains that no personal or personal identifiable information will be used nor stored by the app.

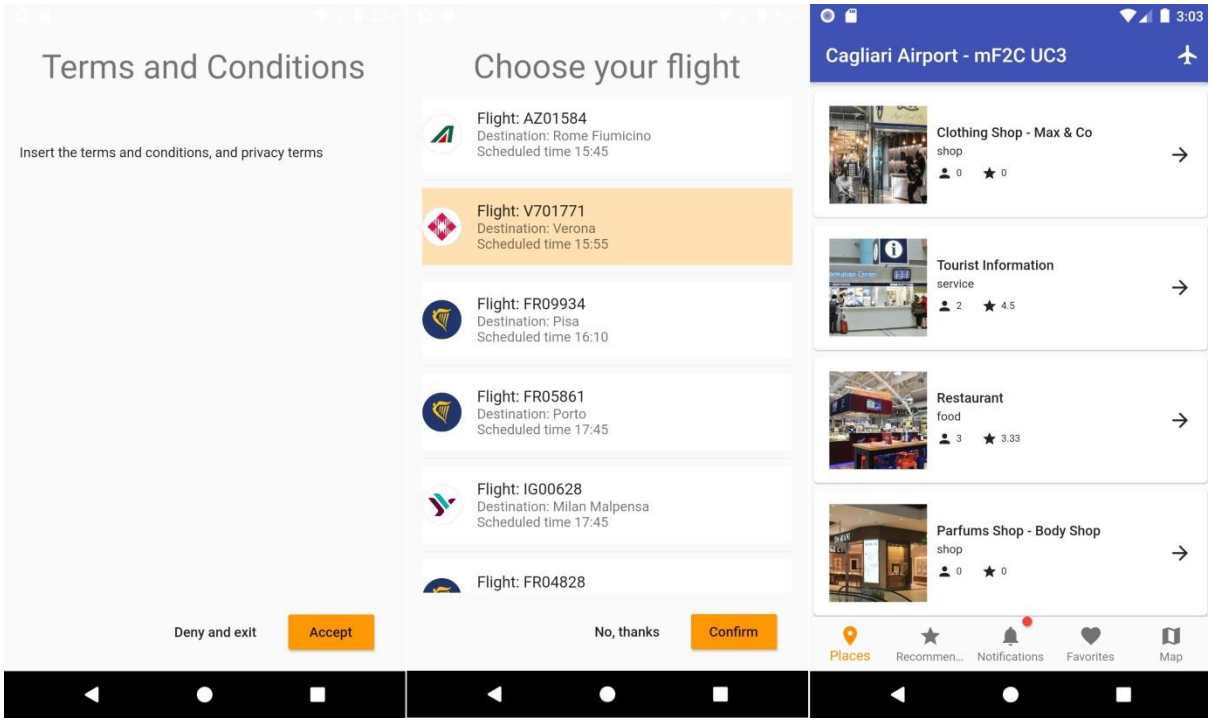

Figure 36 Privacy Terms and first configuration on the app

After that, the user can configure the app according to the following:

- his/her own flight, to get informed on important events related to the flight (assign gate, change gate, delays, first call, last call); In case he/she can change later the selected flight.
- topics of interest, in order to be informed of available Points of Interest (PoIs) in the area.

Then the user moves freely in the field, being tracked and notified about POI's proximity. By the app in every moment the user can get all available information on the selected flight, flight status, and so on.

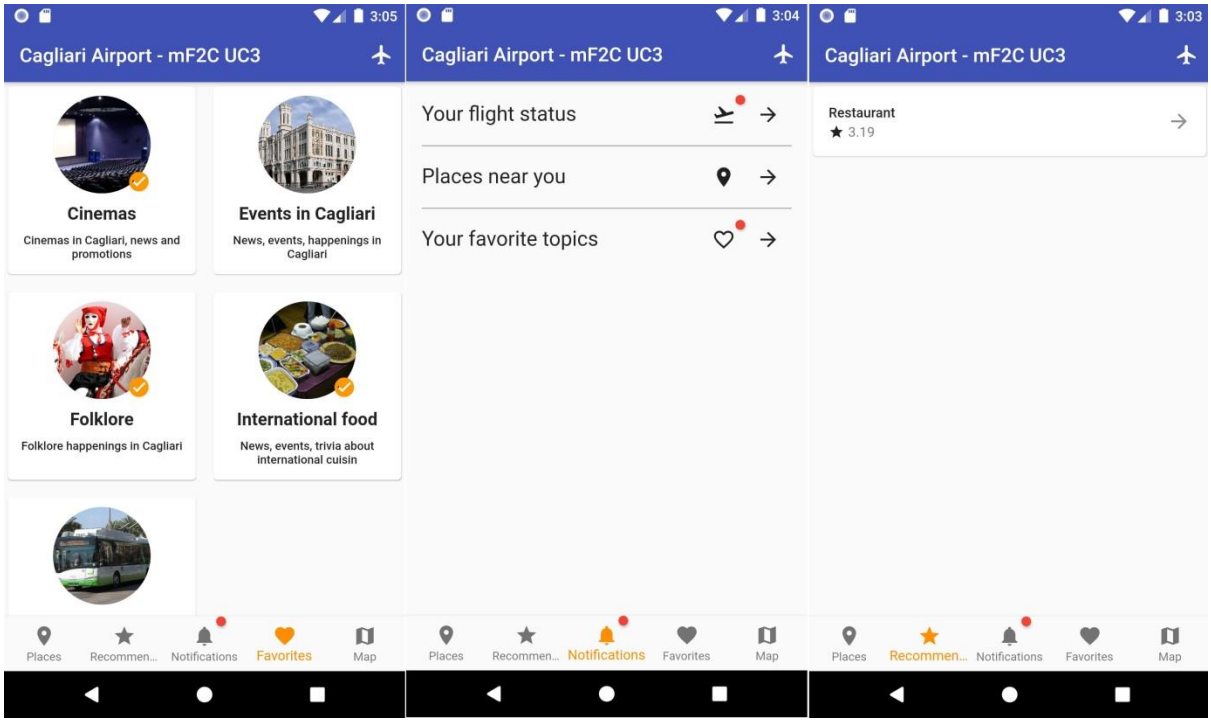

Figure 37 main information available in tabs

Moreover, the app has the following tabs:

- Places
- Recommendations
- Notifications
- Favourites
- Map

Selecting Places, the user can proactively search interesting PoIs, with possibility to get more information on offered services. In this way he/she can realize the whole amount of services offered and make his own choices. But he/she can move around getting driven by PoIs discovery.

Selecting Recommendations, the user can get recommendations on interesting PoIs according to the selected topics and check the rate of the listed PoIs. The option of rating PoIs feeds the topics advisor system, based on Collaborative Filtering algorithm, that propose some PoIs to the user according to the similarities in behaviour with other users.

The Notifications tab shows all relevant alerts on the flight status, nearby PoIs and favorite topics, so here the user is advised to check this periodically. A red bullet announces the reception of new notifications.

The Favorites tab shows the most rated PoIs, so here the user can be driven by the rating of other users. Here he can find daily promotions offered by the various shops.

Finally, the Map tab shows the position of the user in the field as he/she moves, with different symbols to represent the user and available PoIs.

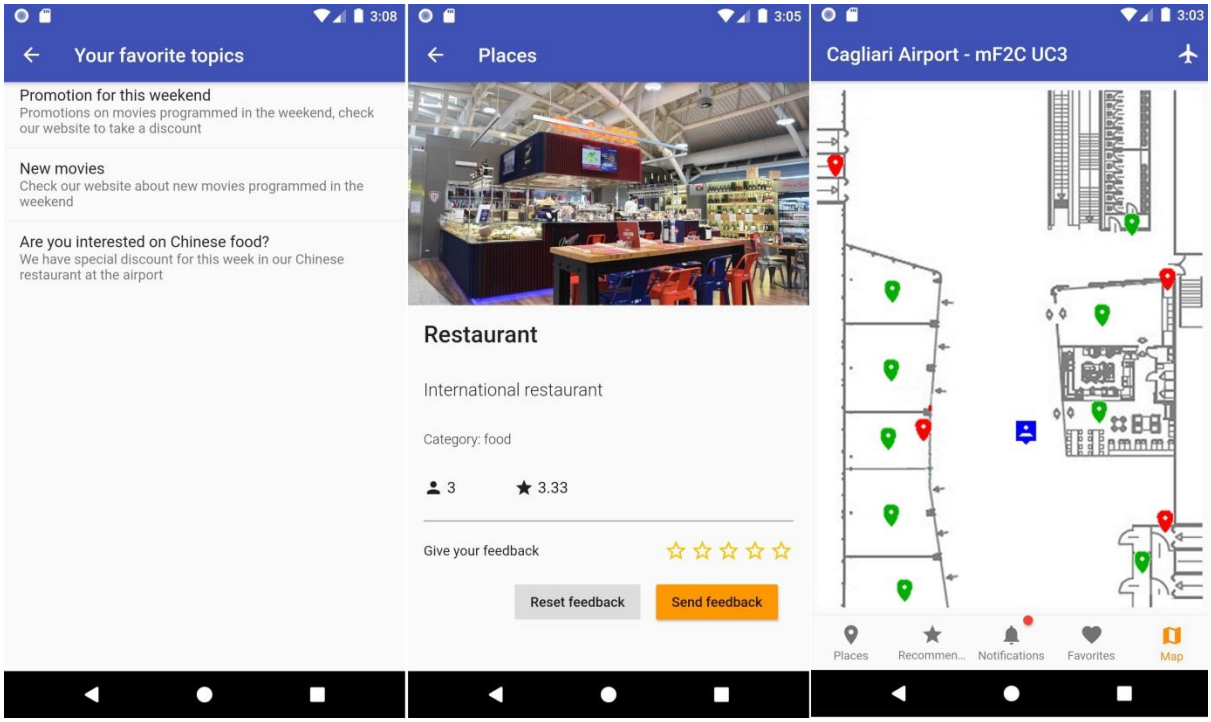

Figure 38 more details on PoIs and the map

There is another usage, dedicated to the airport administrators. They can connect to the Admin Portal, get access to a unified view of the field, having the ability to manage the PoIs information, and related promotions, but also the real time information on active users, with possibility to have dedicated reporting. These reports can show the behaviour of travellers, offering way of co-hort analysis, splitting the users according to low-cost or business flights, or destination or other fields. All these reports can spot bottlenecks in the airport infrastructure and suggest ways of improving it.

5.3. Data Flow Diagrams

In addition to the functionalities already present in IT-1 [2], some new functionalities and integrations have been developed specifically for IT-2. The workflows are detailed in the following paragraphs.

5.3.1. App functionalities

As stated before, the UI of the Android app has been redesigned and improved for a better user experience. With respect to IT-1, there are two new functionalities.

- Rating for a specific POI. The user can select a specific POI and assign a rating from 1 to 5 stars. The rating, properly anonymized, is then delivered in the system until the Cloud backend, where is stored and made available to the topics advisor engine for further computation.
- List of recommended POI. The user can see on-demand the list of recommended POI, which is based on similar user's ratings and is tailored for him/her. This list is produced by the recommended engine.

The workflow for these functionalities is detailed in paragraph 5.3.4.

As already stated, the app needs to connect to the API Server to provide all the functionalities.

5.3.2. User tracking with proximity notification

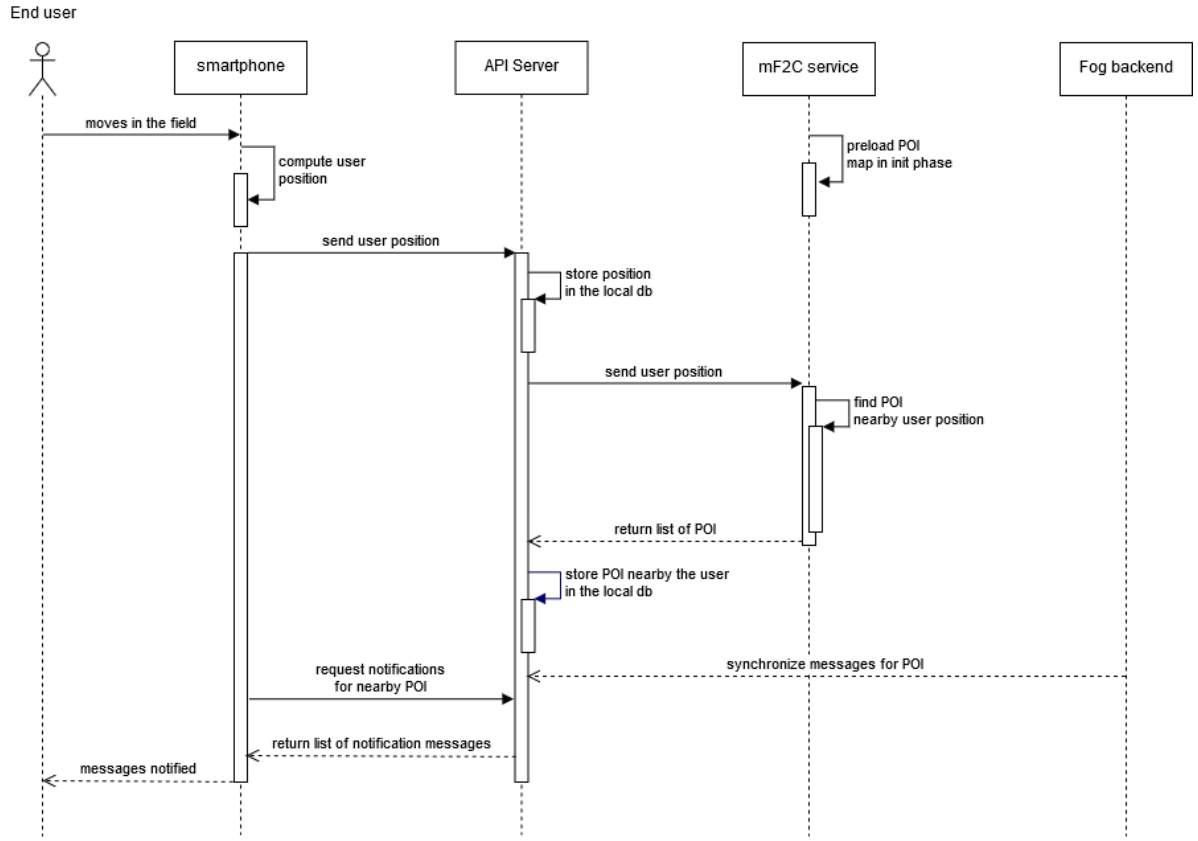

Figure 39 Sequence diagram of user tracking with proximity notification

The figure 39 shows the process for the user that moves in the field while using the Android app. In this scenario, the position data is computed in the user device, which is the smartphone in the diagram, sent to the API Server and stored. This data is then used to compute nearby POI, involving the specific mF2C service. If there are messages related to the found POI, they are finally sent to the user device for proper notification and displaying.

5.3.3.Flight updates and favorite topics notification

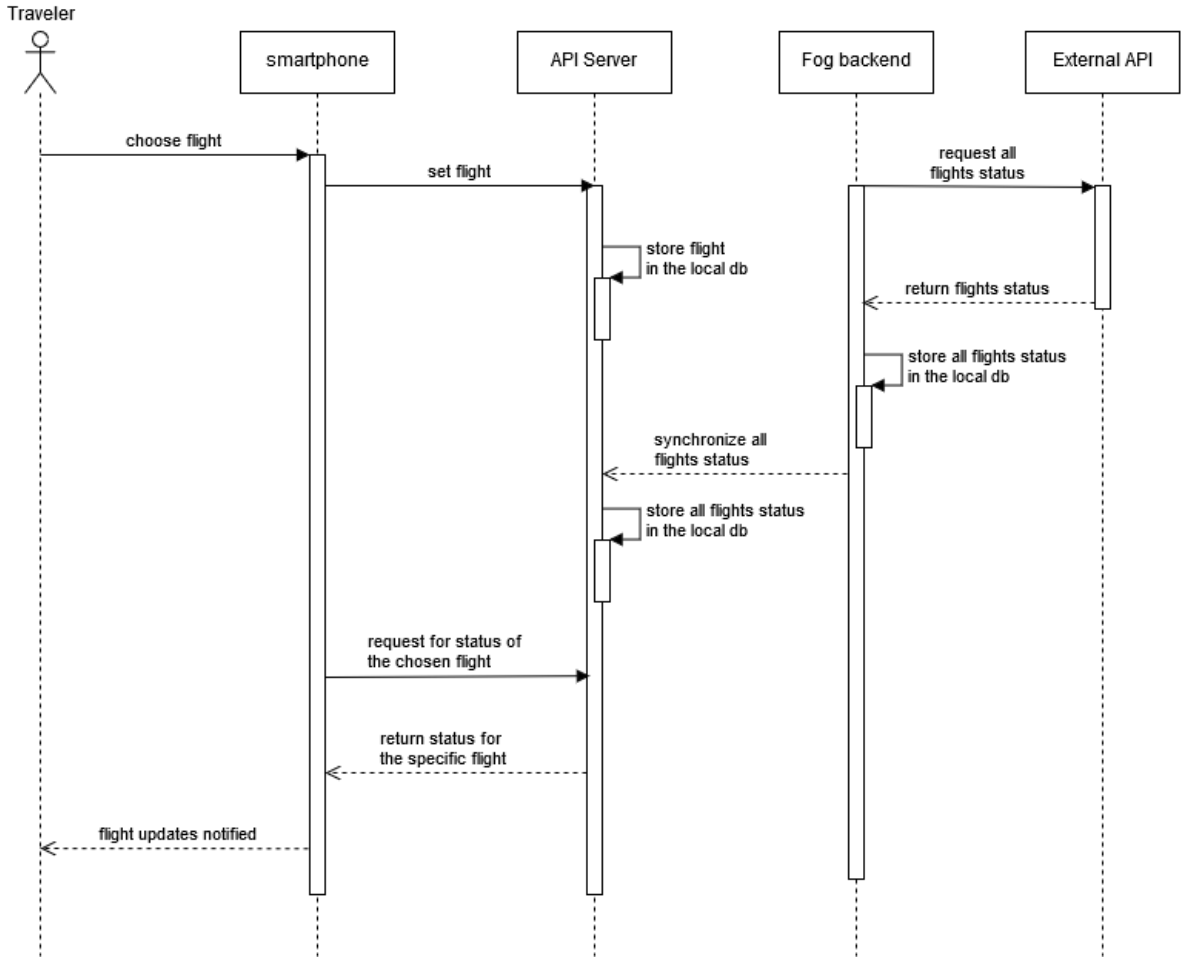

Figure 40 Sequence diagram of flight update

The figure 40 shows the process for the flight update. In this scenario, the end user, typically a traveller, choose his/her flight from the airport timetable. The list is provided by querying an external real-time API hosted in the airport systems and filtered in order to provide only departure flights that a traveller can take. So, for example, departed flights are filtered out from the list. The real-time updates for all the flights are stored in the Fog backend and synchronized in the API Server, where are made available for proper notification to the user. Periodically, in a short time interval, the Android app requests information about the status of the flight chosen by the user. If there are updates, the app promptly notifies the user. Considered updates are, for example, changes regarding the gate, departure time, status like delayed, last call, boarding, etc.

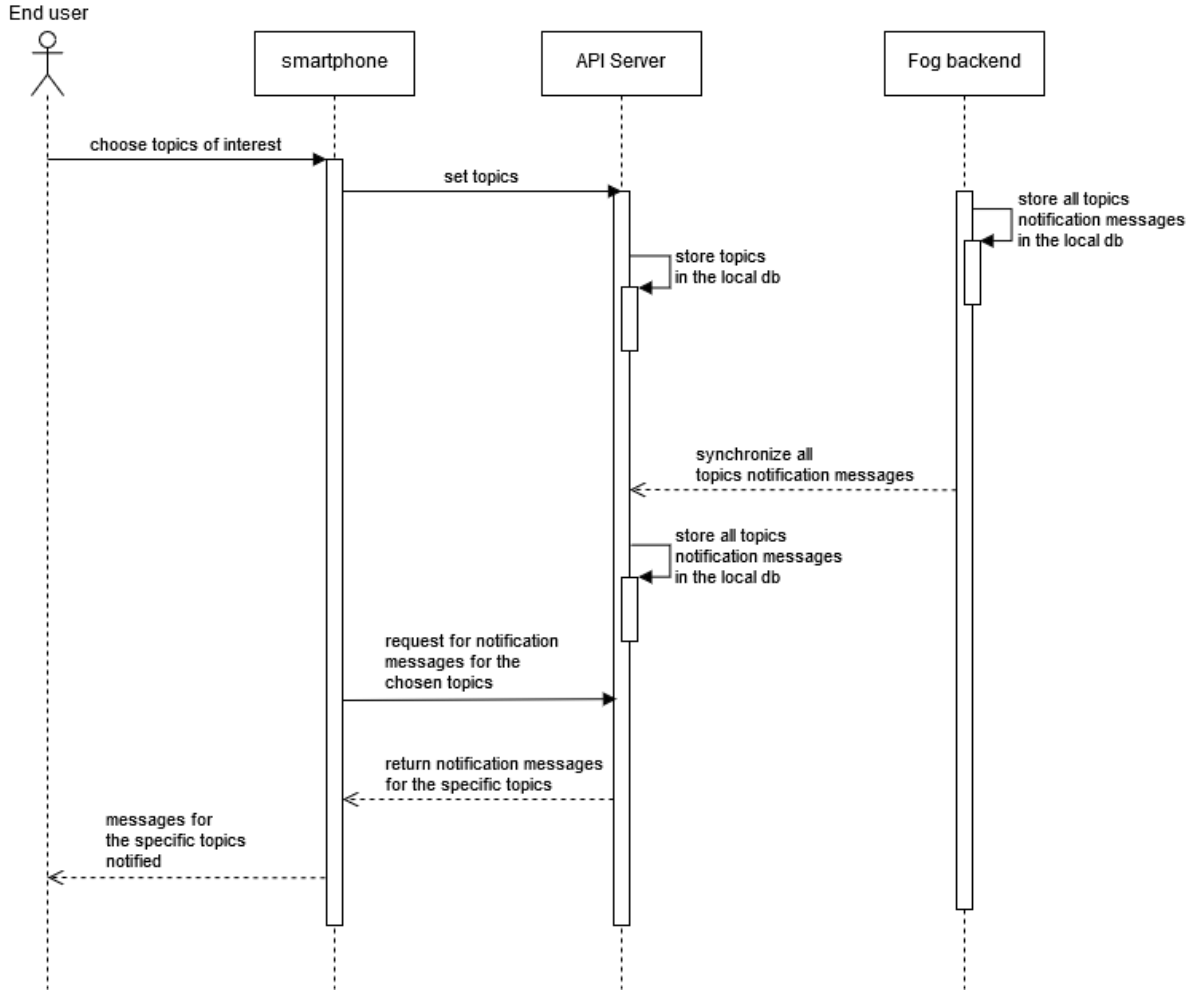

Figure 41 Sequence diagram for favorite topics notification

The figure 41 shows the workflow related to the user's favourite topics notification. The user can select one or more topics of his/her interest from a list, which is configured in a specific catalogue in the API Server database. The topics selected by the user are sent to the API Server and stored in its local database. Holding this data, the system is able to deliver to the user proper notification messages.
The Fog backend has in its local database the complete dataset of the information about topics. For example, this data set can be loaded in with batch jobs or even supplied time by time by mean of the administrator dashboard. The dataset is finally synchronized to the API Server, so when the app periodically requests for updates on the topics selected by the user, the API Server returns them, and the app display the notification messages.

5.3.4.Recommendations

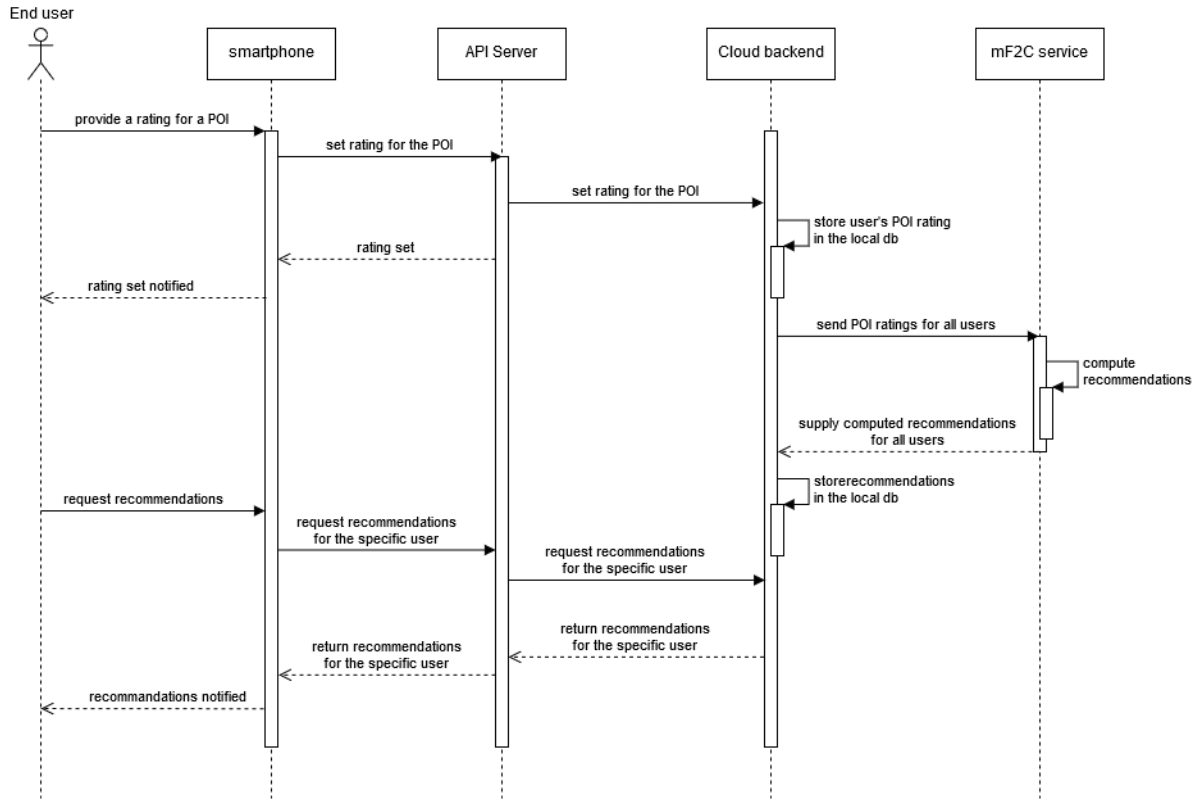

Figure 42 Sequence diagram of topics advisory process

The figure 42 shows the process for the recommendation's functionality. In this process, each user can provide a rating for specific POI. All user ratings for all POI are then sent and stored in the Cloud backend database. This dataset is used by the so-called topics advisor engine to compute recommendations for individual users, using a Machine Learning algorithm. The topics advisor engine is an mF2C service that periodically performs this sort of computation. The output of computation is then sent to the Cloud backend database for storage. When the end user demands the list of recommended POI, the request is forwarded to the Cloud backend database that returns the list for that individual user. Finally, this list is notified to the user in the Android app.

5.3.5.Admin Portal

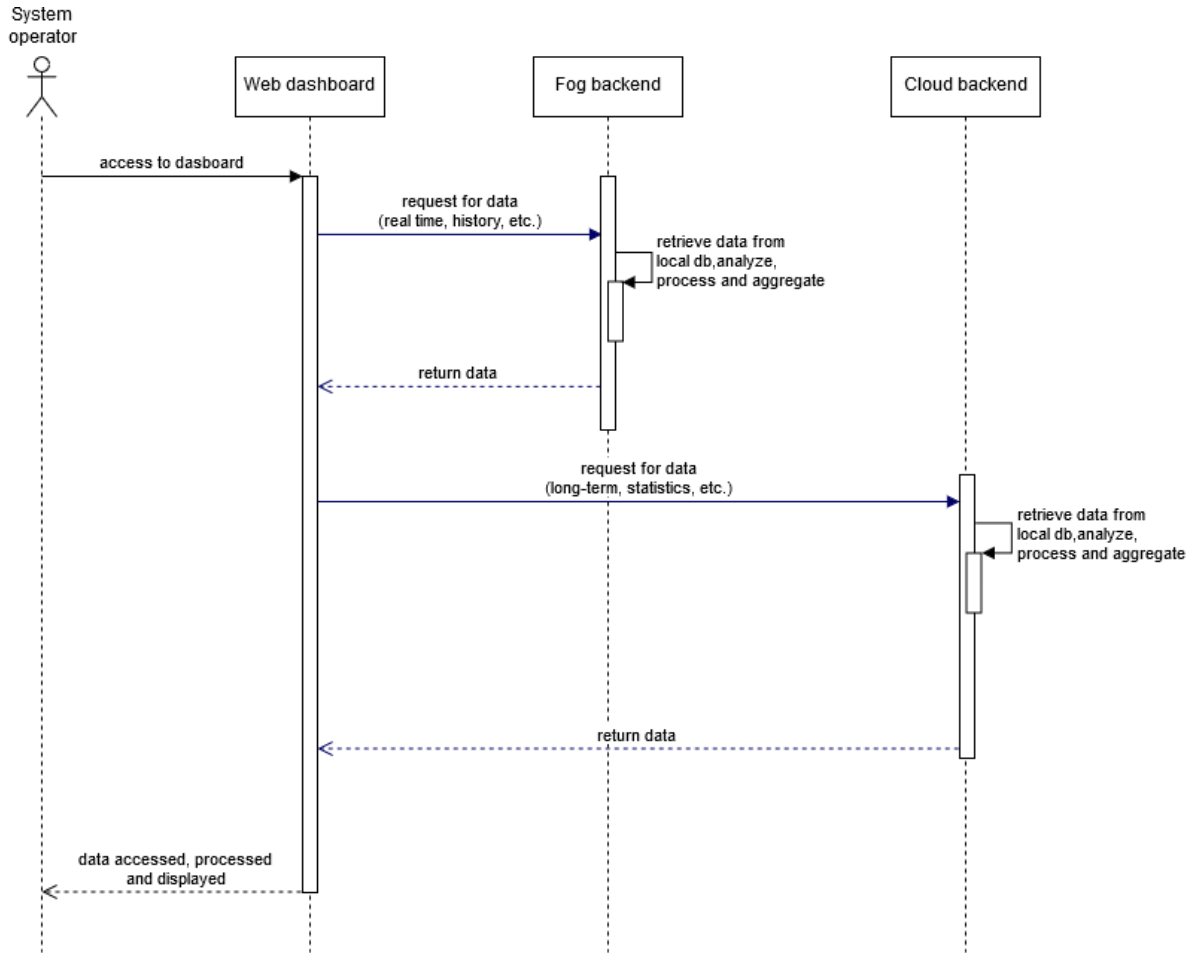

Figure 43 Sequence diagram for dashboard web application

The figure 43 shows the data flow for the Admin Portal (Web Dashboard) component. Basically, it is the same as IT-1, just with some more reports. The system operator accesses to the Admin Portal with a browser and uses all provided functionalities. The data are taken from either the Fog backend or the Cloud backend, depending on the functionality used.

5.4. Experimental Set Up

To step ahead in IT-2 and test proficiently the Smart Fog Hub, UC3 of the mF2C project, we improved first our internal testbed with more resources at fog level, enabling us to test the load balancing under different conditions, using more fog nodes and the cloud to get better results. Several runs have been done to validate the capability of the mF2C agent to distribute the services on all available computing resources, guaranteeing optimal response times.

In this scenario this is the list of devices used for the extensive tests:

- Some Smartphone (ASUS, Samsung) have been used to run the airport app,
- 4 RaspberryPI3 have been used to track people in the field and provide the available services,
- 3 miniPC with 8GB of RAM have been used as fog nodes, that have access to the public airport API (running in voli.sogaer.it server) to get real-time flight information
- Link to the OpenStack instance for cloud support.

With this setup we succeeded in testing the platform under different loading, even simulating more end-users moving in the field, and the different fog and cloud nodes have been attached and detached to check how the system determine the available resources and optimize the scheduling of tasks.

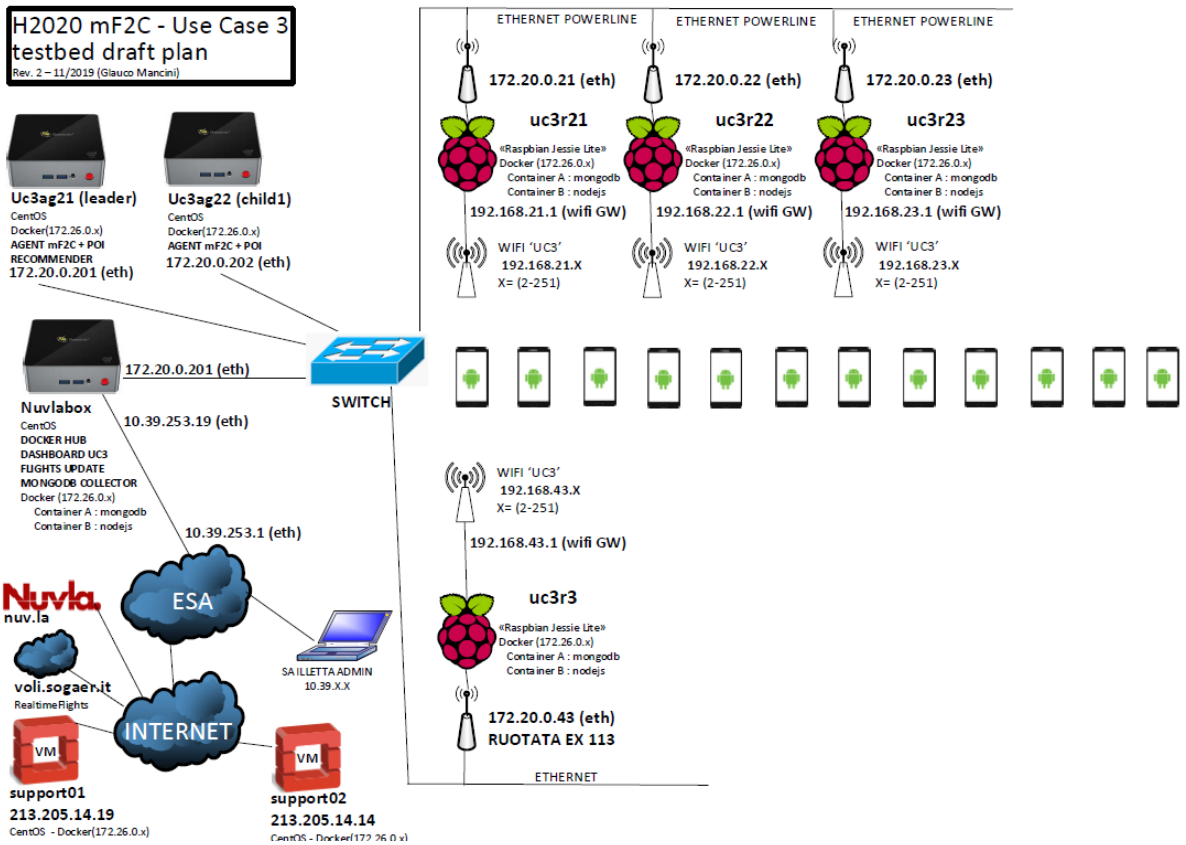

Figure 44 IT-2 internal testbed for Use Case 3

Then the final setting in the airport has been defined. Some meetings with technical specialists of the airport have been held and we decided that the departure pier would be the best option for various reasons:

- Higher concentration of passengers that stay longer in this area
- More shops and PoIs in general
- Easier way to cabling and connecting all devices

In order to guarantee a full coverage of this area 8 points have been selected to install the RaspberryPIs, as in the following picture:

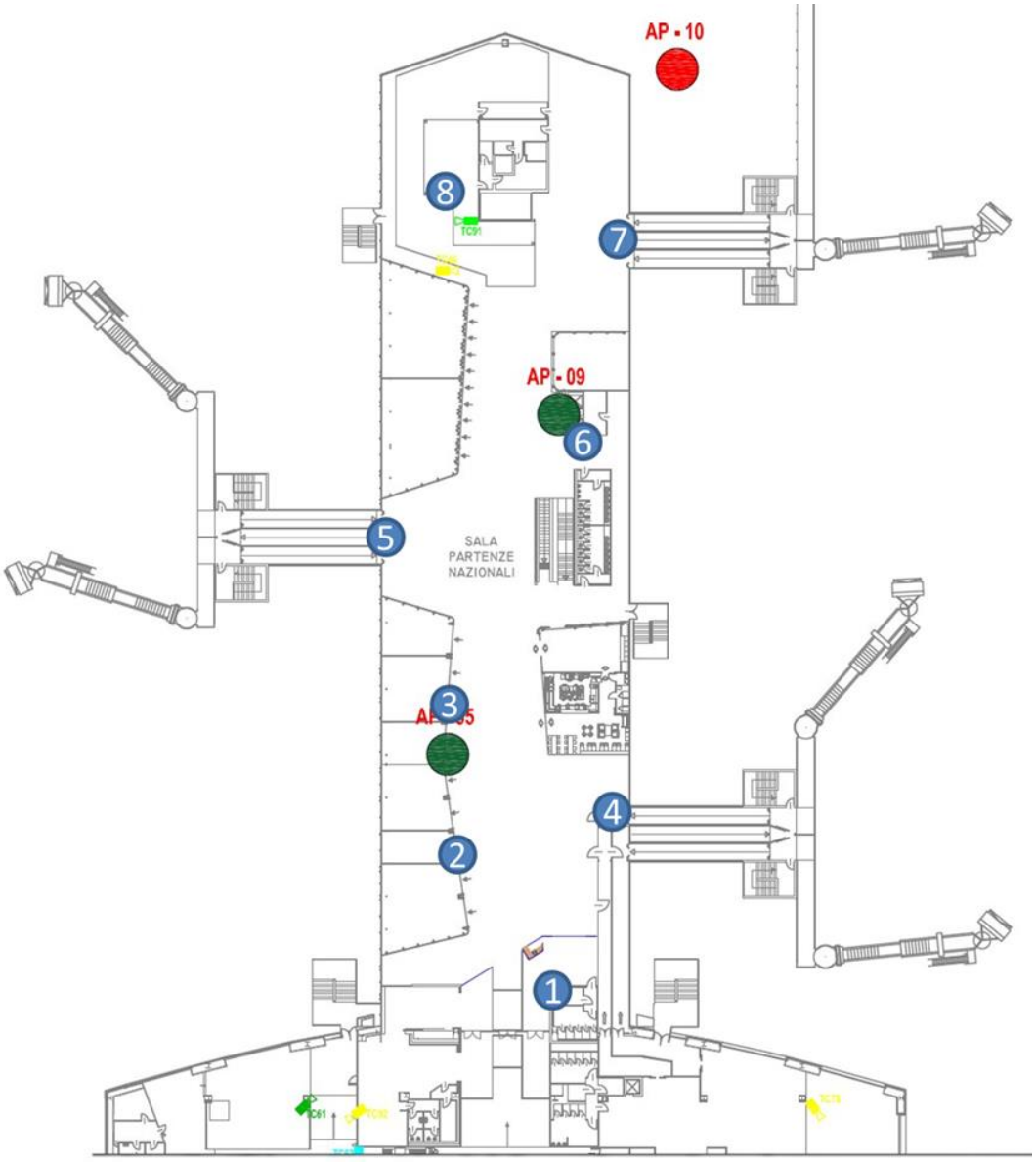

Figure 45 UC3 deployment in the airport

The Blue spots represents the points where we placed the RaspberryPIs, while the green and red spots present existing airport WIFI AP that can be used as additional points to improve the precision of people tracking.

The fog nodes have been positioned in a separate datacentre room, with the router that is fibre connected with the Tiscali Datacentre, which guarantee connection with the Openstack instance.

All computing nodes work in a dedicated VLAN, so separated from the airport equipment, thus fulfilling a security strict requirement imposed by the airport.

The list of devices used for the final scenario and demonstration is the following:

- More Smartphones to run the airport app, and in the testbed several runs have been performed with high number of simulated users,
- 8 RaspberryPI3, as described in Figure 46, to guarantee full coverage of the departure area, to track people in the field and provide the available services,
- 3 miniPC with 8GB of RAM have been used as fog (working) nodes,

- 1 Nuvlabox mini equipped with 8 GB RAM, that acts as fog leader, provides real-time computing and storage resources to the edge elements, and have access to the public airport API (on voli.sogaer.it server) to get real-time flight information
- CISCO 3400 providing fibre connection to the Tiscali datacentre
- OpenStack instance for cloud support and long-term storage for all data produced that needs to be analysed, even offline, in order to provide feedbacks on system behaviour and allow for possibly optimizations.

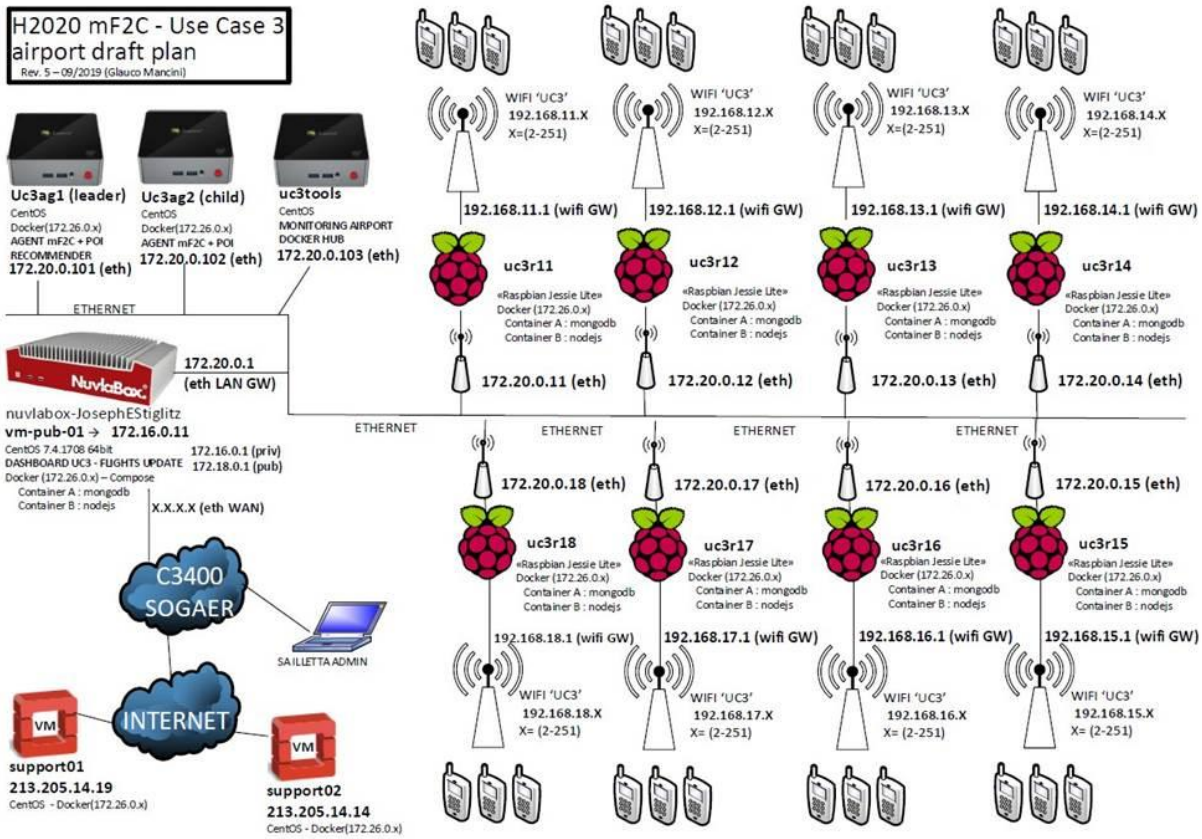

Figure 46 UC3 final system diagram

More tests have been performed in the final testbed, in order to validate the use case in the final settings, facing a real-world scenario and related challenges.

This setting could be extended, according with airport requests, adding more RaspberryPIs to cover other areas of the airport, and more fog nodes could be added as well to scale-up the system as the number of people to be tracked grows, leveraging the automatic load distribution and off-loading provided by the mF2C agent through COMPSs.

5.5. Results and KPI measurements

Since the focus of the use case is on a fully edge centric solution the defined KPIs resemble the key elements of the Fog/Edge computing paradigm:

Latency and response Time –

The continuous interaction with the user at the edge and the real time response demanded by proximity marketing ask for fast and optimized response time, leveraging the mF2C orchestration, load balancing and distribution, with a proficient use of resources (processing, storage, communication). This is stressed by the huge amount of data (movements, choices, rates, preferences,

etc.) that is generated by all end-users. This force the optimal distribution and automatic scalability capabilities of the mF2C framework.
For all these reasons a real time response time for every user under every circumstance is mandatory.

Bandwidth and QoS –

Applications are based on continuous data communication, so tight engagement with the end user, require efficient use of available bandwidth and resilience capabilities.
This is fulfilled by the hierarchical communication structure and redundant architecture elements of the mF2C, that take advantage of the redundant links at the edge and enables the correct functioning of the system even in case of failure of some components.

Data locality and Regulatory Compliance -

Given the high performance of current (and future) edge devices, distributed processing between nearby nodes is more efficient than referring to far-away centralized servers in the cloud. The same edge devices are powerful enough to use intelligent software and run autonomous decision-making applications. Since all data, including personal and sensitive data, is generated and located in the edge, fast calculation is possible locally without the need to move data to the cloud, where fog-to-cloud topology enable filtering and optimization on resources & communications while fulfilling the real-time requirement, not possible with direct cloud topology.
The control of trust and management of sensitive information need to be started from the edge, from people that use these devices.
With these assumption privacy and security by design are implemented in the mF2C framework, thus guaranteeing the confidentiality, integrity and availability of personal data, fulfilling GDPR rules.
Applications can use these native capabilities and policies enforcement to have satisfactory privacy management.

Performance Tests

To validate the performance of the mF2C within the Use case 3, some following measure has been defined:

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

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

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

In terms of metrics the response time for every request is collected, in third scenario the number of requests processed at fog level are summed separately, thus determining the percentage of requests processed at fog level against the total.
In the "**Smartphone to Fog2Cloud"** test the first run has been run with only one fog available node (PCbox), while in the second run an additional PCbox has been included.

The “Smartphone to Fog2Cloud” approach performed the best, and performance improved adding more fog nodes. With 10000 samples, we measured an improvement of about 25% compared to the “Smartphone to Cloud” approach. The results have been fully described in D5.2.

5.6. Business Prospective and conclusions

Engineering has validated the initial idea of the adoption of the mF2C to support proximity marketing applications in indoor spaces seems a very good and beneficial decision, and the mF2C capability to manage different fogs gives a relevant advantage in designing and shaping complex smart cities solutions.

For the final deployment of the system, an engagement with the Cagliari Airport Management company (Sogaer) started at the beginning of the current year, they expressed their interest in the proposal, both asking to extend the coverage of the SFHS Use Case to the full airport area, and asking for sophisticate communication means to inform all passengers of the app inviting them to use it. They are interested in developing the app to approach a full commercial maturity of the solution. This could give them chance to collect information on traveller’s behaviour, service usage, spotting bottlenecks in current infrastructure, so giving advice on the best way of improving the airport service offering and maximizing their revenues.

At the same time Engineering Line Business directors expressed their interest to have a commercial product to be offered and deployed to other airports. Also, Tiscali expressed their interest in the use case, they offered to support the improvement of the usability of the app, they foresee a good opportunity to leverage the engagement of the airport services to sell more products and services.

The entertainment app offered to end users could be enhanced with a freemium proposition, so more revenues in case of premium users, or could include advertise banners.

Engineering plans also to use mF2C as a standardized platform, to be integrated in their commercial and research platforms, to enable simple provisioning of services in fog cases, particularly in smart cities scenarios. In this kind of scenarios, it is also possible to enable payment methods in the platform, so that the end user can take advantage of promotions or special discounts in the region, feeding the microeconomics. For example, it could be possible to enable payments with credit card or through a personal wallet in the app, or even manage cryptocurrencies or tokens in a blockchain, involving all stakeholders, like commercial and administrative partners.

For this reason, Engineering leverage its extensive relationships with Municipal, Regional and Nation Public Administrations to offer new innovative smart cities services, for example to offer advice and guidance to cruise travellers that stay just one day in the city.

6. Next Steps: Demonstration and final review

6.1. Proposed demonstration strategy

In a similar strategy to that carried out during the IT-1 demonstration, the mF2C project envisions an IT-2 demonstration strategy centred in two main steps, a standalone demonstration, and the use cases validation. In the IT-1 demonstration, we first ran a scenario aimed at validating the registration process as well as potential policies for recovery from a leader failure. This decision was since the agent was not completely deployed and only a minor set of functionalities could be tested and validated in the use cases. In IT-2, the agent, and the microagent, are both completely deployed, and the proposed demonstration strategy will not test again the registration steps but will focus on the agent installation and deployment. For the sake of illustration, in IT-2 we propose a standalone demonstration, covering aspects related to agent and microagent installation, showing how the download process is to be done, including the GUI used for downloading and obtaining the "How To", where the whole the installation process is described. Thus, the standalone demonstration will show the installation of both the agent and microagent, including two different devices for the microagent, the normal agent and the leader agent to show diversity, setting a specific topology. Then, two services will be executed to show how a microagent is selected and how the different components in the topology work.

Once the standalone demonstration is done, the second step is the mF2C validation in the three proposed use cases. To that end, the different mF2C components are deployed on each individual use case and several KPIs are assessed to show the benefits of an mF2C deployment.

6.2. Standalone demo

As explained before, some of the mF2C functionalities are demonstrated in IT-2 independently of the execution of an mF2C service in mF2C use cases. These functionalities are detailed in the upcoming sections. This demo is complementary to the project use cases. It is intended to demonstrate the use of mF2C to configure an mF2C processing topology out of an available set of devices at the Edge. In order to demonstrate this, we have used two Edge devices available as commercial offering from two of mF2C partners: SixSQ's Nuvla Box (https://sixsq.com/products-and-services/nuvlabox/overview) and Atos BullSequana Edge (https://atos.net/en/products/bullsequanaedge).

6.2.1. Topology

There are many options to build a topology and we propose the one in Figure 47 47 for this standalone demo.

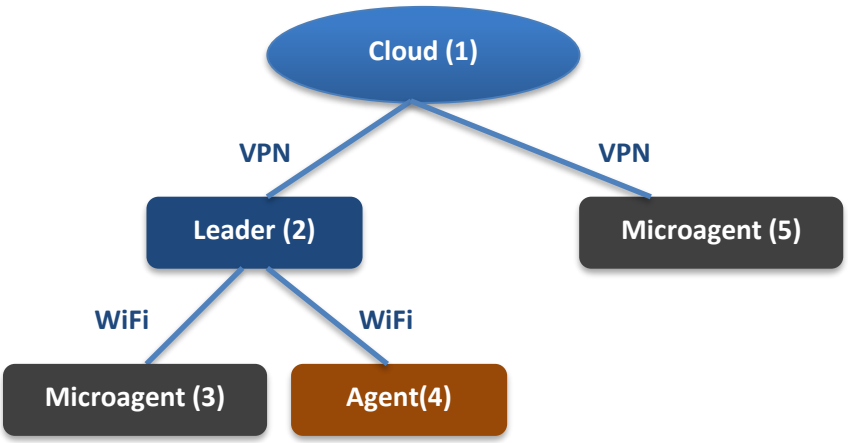

Figure 47 Topologies for standalone demo

We may also have considered Microagent (5) to stay connected to the leader but we think that having this sort of two areas from cloud is much more appealing.
The following devices are considered to be deployed in the topology above:

- ATOS Edge Server
- NuvlaBox
- Laptop *2 RP
- Cloud at ENG

According to this list of eligible devices and assuming that different devices may play on different topology components, we propose to stay with Configuration 2 from the list below:

Topology component	Config 1	Config 2	Config 3
Cloud (1)	Cloud at ENG	Cloud at ENG	Cloud at ENG
Leader (2)	ATOS Edge Server	Laptop #1	Laptop #1
Microagent (3)	RP	RP	RP
Agent (4)	Laptop #1	ATOS Edge Server	Laptop #2
Microagent (5)	NuvlaBox	NuvlaBox	NuvlaBox

Table 6 Standalone demonstration possible configurations

So, the topology to be set is the following one:

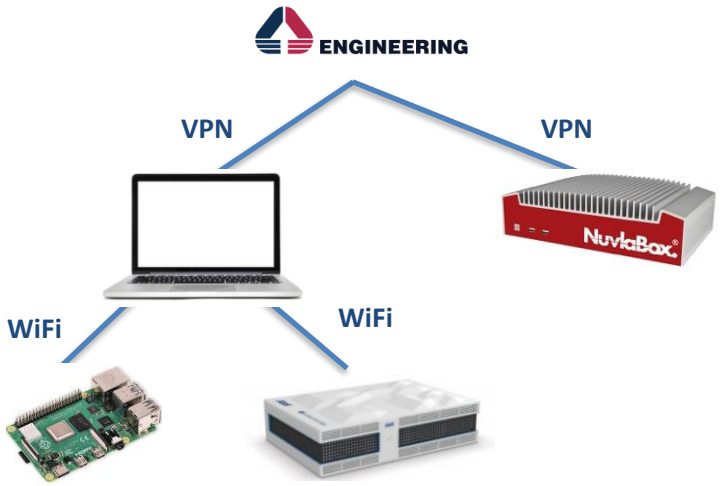

Figure 48 Final topology for standalone demo

Certainly, some other configurations may come up, for example playing with Nuvla and the RP, but the list above is just shown as a starting point.

6.2.2.Features to be tested

We assume:
The leader is already installed at the laptop
Microagent installation at RP

In this demo we will basically test:
a) User registration
b) Download the agent
c.1) Agent installation at the Atos Edge Server
c.2) The microagent is already installed at the NuvlaBox
c.3) Create the SLA template for the web camera service
d) Create a service launched at cloud to be deployed at RP (having a web camera) with the SLA previously created)
e) Create the COMPSs Hello World (query received at the Leader) to run on the Atos Edge Server and move to the lifecycle (through the dashboard) to show where the service is executed.

Bullets c.3), d) and f) will be deployed through the dashboard.

7. Conclusions

This deliverable presents the final deployment and results of the use cases used during the mF2C project to validate the technology developed in relevant commercial environments. The second section introduces the rationale used for the Proof of Concept implementation, as well as listing the map of technical and scientific challenges addressed within the project and demonstration strategy.

The following section is dedicated to the stand-alone demonstration that has been developed to illustrate some of the functionalities of the architecture itself. Although the use cases are used to validate the mF2C ecosystem, it is sometimes not easy to make some of the functionalities visible without interfering with the use case itself. For demonstration purposes, it was decided that some of the most transparent functionalities, those that happen behind the scenes but are not easily observed within the use case, would be shown in a stand-alone demonstration to be presented together with the use cases. Details of the demonstration and functionalities can be found in section three.

The three following sections are dedicated to the three use cases. The Emergency Situation Management in Smart City describes the improvements performed on top of the first iteration of the project, the hardware, the storyline, the data flow diagrams, and the set-up used. The results are also presented with numerical KPI to provide an objective evaluation of the results. The delay is reduced by 76%, the CAPEX reduced by 17%, the reliability increased to almost 100% and the QoS increased by 10%. Finally, the conclusions and business opportunities are presented.

The Smart boat use case section follows the same structure and reports a measurable two and a half times improvement in response time, 15-25 % of time saved by using the existing mF2C deployment feature compared to developing a solution in-house, and a communication distance improvement of about 20%.

The Smart Fog-Hub service reports a decrease response time of 15%, a QoS and resiliency improvement due to the intrinsic redundancy of the mF2C architecture and the possibility to process personal data at the edge using security/privacy by design mF2C features, thus fulfilling the GDPR constraints. Also, the use case has sparked the interest of the Cagliari airport and the region, allowing the installation of the use case in a real airport environment, opening commercialisation opportunities through this collaboration, both for Engineering's use case and for mF2C itself.

We can conclude that the mF2C ecosystem has been successfully validated in all three use cases, providing an objective evaluation of the performance of the framework, and opening potential exploitation paths.

References

[1] mF2C Consortium, “D5.1 mF2C reference architecture (integration IT-1),” 2018.

[2] mF2C Consortium, “D5.3 mF2C solution demonstration and field trials results (validation IT-1),” 2018.

[3] mF2C Consortium, “D2.7 mF2C Architecture (IT-2),” 2019.

[4] mF2C Consortium, “D3.4 Design of the mF2C Agent Controller Block (IT-2),” 2019.

[5] mF2C Consortium, “D4.4 Design of the mF2C Platform Manager block components and microagents (IT-2),” 2019.

[6] mF2C Consortium, “D3.3 Design of the mF2C Controller Block (IT-1),” 2017.

[7] mF2C Consortium, “D4.3 Design of the mF2C Platform Manager block components and microagents (IT-1),” 2017.