Smart Place Manager:
provisioning, deployment and monitoring of cloud-based application servers

Pedro Miguel Reis Queiroga Branco Valério

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Miguel Filipe Leitão Pardal
Prof. António Manuel Raminhos Cordeiro Grilo

Examination Committee
Chairperson: Prof. Horácio Cláudio De Campos Neto
Supervisor: Prof. Miguel Filipe Leitão Pardal
Member of the Committee: Prof. Renato Jorge Caleira Nunes

June 2018
Acknowledgments

Firstly, I would like to thank my advisors Prof. Miguel Pardal e Prof. António Grilo that made possible my assignment for this dissertation and also because they were tireless while helping me in the development of it. I also would like to thank the members of the committee that evaluated this work and provided feedback.

Secondly, I would like to thank my mother for all the support and specially for her constant belief and obstination. I also want to dedicate this thesis to her, to my grandparents and specially to my father, I know that wherever he is, he'll be very proud of me.

Lastly i want to thank all my friends and colleagues that supported me throughout my academic path.
Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code Conduct and Good Practices of the Universidade de Lisboa.
Abstract

The Internet of Things aims to connect billions ($10^9$) of smart physical objects to the Internet, which can bring a promising future of smart places or even smart cities. These objects, equipped with sensors and actuators, are expected to generate large amounts of raw data that can be captured automatically and then sent to the cloud for further processing. These data can later be used for knowledge discovery and to improve business processes.

The objective of this work was to build a mobile app - a Smart Place Manager - on the iOS mobile platform, to provision and configure servers in the Cloud platform and to support the IoT application’s operation. The mobile application also monitors the processing, memory and storage usage among other metrics, taking action when necessary to the effective management of the Smart place. The response time is also monitored, which is often decisive in IoT applications, requiring low or predictable latency.

The demonstration scenario is an automated building access control that uses a RFID event processing software to track accesses to the facilities. The experimental results show that the developed tool can effectively and efficiently provision and monitor a smart place environment using a mobile application.

Keywords: Internet of Things, Smart Places, Cloud computing, Fog computing, Cloud Automated Provisioning and Monitoring, Application Deployment, Mobile Application, RFID
Resumo

A Internet das Coisas tem como objetivo conectar biliões ($10^{10}$) de objetos físicos inteligentes à Internet, o que pode trazer um futuro promissor de lugares inteligentes ou mesmo de cidades inteligentes. Espera-se que esses objetos, com sensores e atuadores, gerem grandes quantidades de dados brutos que possam ser capturados automaticamente e, em seguida, enviados para a nuvem para processamento adicional. Esses dados podem ser usados posteriormente para a descoberta de conhecimento e para melhorar os processos de negócios.

O objetivo deste trabalho foi criar um aplicativo móvel - um Smart Place Manager - numa plataforma móvel como o iOS, para provisionar e configurar servidores na plataforma Cloud e para suportar a operação da aplicação IoT. Este aplicativo móvel também monitoriza o processamento, memória e o uso de armazenamento entre outras métricas, agindo quando necessário à gestão eficaz da Smart Place. O tempo de resposta também é monitorizado, o que geralmente é decisivo em aplicações IoT, que exigem latência baixa ou previsível.

O cenário de demonstração é um controlo de acesso a um edifício automatizado que usa um software de processamento de eventos RFID para rastrear os acessos às instalações. Os resultados experimentais mostram que a ferramenta desenvolvida é eficaz e eficiente para fornecer e monitorizar um local inteligente usando uma aplicação móvel.

Keywords: Internet das Coisas, Smart Place, Computação em Nuvem, Smart Place Manager, Computação em Nuvem, Computação em Nevoeiro, Provisionamento e Monitorização automático na Nuvem, Deployment Aplicacional, Aplicação Móvel, RFID
# Contents

<table>
<thead>
<tr>
<th>List of Tables</th>
<th>viii</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>xi</td>
</tr>
<tr>
<td>List of Source Codes</td>
<td>xv</td>
</tr>
<tr>
<td>Acronyms</td>
<td>xvii</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Application Domains ..... 1

1.1.1 Example Scenario ..... 3

1.2 Objectives ..... 4

1.3 Dissertation Outline ..... 5

## 2 Background

2.1 Concepts and tools ..... 7

2.1.1 Fog and Edge Computing ..... 7

2.1.2 Cloud Computing and Mobile Cloud Computing ..... 9

2.1.3 Provisioning, deployment and monitoring APIs: the AWS cloud example ..... 12

2.2 Related Work ..... 14

2.2.1 EPC Framework ..... 15

2.2.2 GS1 EPCglobal Architecture ..... 15

2.2.2.1 Fosstrak Platform ..... 16

2.2.2.2 Chef ..... 17

2.2.2.3 Docker ..... 19

## 3 Solution Design

3.1 Business Perspective ..... 21

3.2 System Perspective ..... 23

3.3 Technology Choices ..... 24

3.4 Designing of Smart Place Manager API ..... 24

3.4.1 Cloud automation and Cloud orchestration ..... 24

3.4.2 SPM Backend server ..... 26

3.5 Designing of Smart Place Manager App ..... 27

3.5.1 MVC Design on Swift ..... 27

## 4 Solution Implementation

4.1 Implementation of Smart Place Manager API ..... 31

4.1.1 Chef ..... 32
List of Tables

5.1 Requirements Achievement - R01 Create Smart Place .............................................. 58
5.2 Requirements Achievement - R02 Actions and Details .............................................. 58
5.3 Requirements Achievement - R03 Monitor Smart Place .............................................. 58
5.4 Abstract Power consumption of Smart Place Manager Application ............................... 66
5.5 Smart Place Manager Request' Latencies table ............................................................ 66

B.1 SPM API create instance request ............................................................... 85
B.2 SPM API Start Instance ................................................................................. 86
B.3 SPM API Stop Instance ............................................................................... 87
B.4 SPM API create instance request ............................................................... 88
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Smart Cities Applications</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>RFID Example Domain</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>Cloud, Fog and Edge computing combined</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Cloud Computing Layers</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>MCC Architecture</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Offloading process overview</td>
<td>12</td>
</tr>
<tr>
<td>2.5</td>
<td>EPCGlobal Architecture Framework</td>
<td>15</td>
</tr>
<tr>
<td>2.6</td>
<td>Fosstrak architecture</td>
<td>16</td>
</tr>
<tr>
<td>2.7</td>
<td>Chef architecture</td>
<td>18</td>
</tr>
<tr>
<td>2.8</td>
<td>Docker architecture</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>Smart Place Manager Logical view of overall architecture</td>
<td>23</td>
</tr>
<tr>
<td>3.2</td>
<td>Provisioning Sequence Diagram</td>
<td>25</td>
</tr>
<tr>
<td>3.3</td>
<td>Technological view of Automatic Provisioning and Monitoring Workflow</td>
<td>26</td>
</tr>
<tr>
<td>3.4</td>
<td>MVC Architecture</td>
<td>28</td>
</tr>
<tr>
<td>4.1</td>
<td>Deployment Diagram for Smart Place Manager Architecture</td>
<td>37</td>
</tr>
<tr>
<td>4.2</td>
<td>URLSession architecture</td>
<td>39</td>
</tr>
<tr>
<td>4.3</td>
<td>Main Screen SPM APP</td>
<td>40</td>
</tr>
<tr>
<td>4.4</td>
<td>Create Screen SPM APP</td>
<td>41</td>
</tr>
<tr>
<td>4.5</td>
<td>Monitoring Screen Spm App</td>
<td>41</td>
</tr>
<tr>
<td>4.6</td>
<td>About Screen Spm App</td>
<td>42</td>
</tr>
<tr>
<td>4.7</td>
<td>About Screen Spm App</td>
<td>42</td>
</tr>
<tr>
<td>4.8</td>
<td>List Screen Spm App</td>
<td>43</td>
</tr>
<tr>
<td>4.9</td>
<td>Detail Screen Spm App</td>
<td>43</td>
</tr>
<tr>
<td>4.10</td>
<td>About Screen Spm App</td>
<td>44</td>
</tr>
<tr>
<td>5.1</td>
<td>SPM Requirement 01</td>
<td>46</td>
</tr>
<tr>
<td>5.2</td>
<td>Main public survey Resume</td>
<td>49</td>
</tr>
<tr>
<td>5.3</td>
<td>App Usability Survey</td>
<td>50</td>
</tr>
<tr>
<td>5.4</td>
<td>User Experience Survey</td>
<td>51</td>
</tr>
<tr>
<td>5.5</td>
<td>The iPhone is on with power supplied by the power monitor</td>
<td>52</td>
</tr>
<tr>
<td>5.6</td>
<td>Instruments options</td>
<td>52</td>
</tr>
<tr>
<td>5.7</td>
<td>Cpu usage impact on energy using CPU activity log on Instruments</td>
<td>53</td>
</tr>
<tr>
<td>5.8</td>
<td>Energy impact gauge on Xcode</td>
<td>54</td>
</tr>
<tr>
<td>5.11</td>
<td>Speedtest result on survey day</td>
<td>57</td>
</tr>
<tr>
<td>5.12</td>
<td>Application Usability Evaluation - Steps to complete final goal</td>
<td>59</td>
</tr>
<tr>
<td>5.13</td>
<td>Application Usability Evaluation - Landscape Orientation</td>
<td>59</td>
</tr>
</tbody>
</table>
5.14 Application Usability Evaluation - Navigation ................................................. 60
5.15 Application Usability Evaluation - Content .................................................. 60
5.16 Application Usability Evaluation - Look and feel ......................................... 61
5.17 Application Usability Evaluation - Scrolling .................................................. 61
5.18 Application Usability Evaluation - Application Response Time .................... 62
5.19 Application Usability Evaluation - Ease of Use ........................................... 62
5.20 Application Usability Evaluation - Experience fun ...................................... 62
5.21 Application Usability Evaluation - Final Goal .............................................. 63
5.22 Energy impact on instruments measures ...................................................... 63
5.23 Energy impact on instruments while standby ................................................ 64
5.24 Energy impact on instruments while creating .............................................. 64
5.25 Energy impact on instruments while listing .................................................. 65
5.26 Energy impact on instruments while monitoring (Starting instance and Modifying) .......................................................... 65

A.1 SPM R1 - Create Smart Place Requirements summary .................................... 73
A.2 SPM R2 - Actions and Details Requirements summary .................................. 74
A.3 SPM R3 - Monitor Smart Places Requirements summary ............................... 75
A.4 SPM Requirement R1.1 - Choose Smart Place Type ..................................... 76
A.5 SPM Requirement R1.2 - Choose Smart Place Name ..................................... 76
A.6 SPM Requirement R1.3 - Choose Smart Place Cloud Provider ....................... 76
A.7 SPM Requirement R1.4 - Choose Smart Place Region ................................... 77
A.8 SPM Requirement R1.5 - Choose Smart Place CPU Type ............................... 77
A.9 SPM Requirement R1.6 - Choose Smart Place Storage Size ........................... 77
A.10 SPM Requirement R2.1 - Create Smart Place ............................................ 78
A.11 SPM Requirement R2.2 - Terminate Smart Place ........................................ 78
A.12 SPM Requirement R2.3 - List Smart Places ............................................... 79
A.13 SPM Requirement R2.4 - Start and Stop Smart Place ................................... 79
A.14 SPM Requirement R2.5 - Read Smart Place rmation .................................... 80
A.15 SPM Requirement R3.1 - Read Memory Utilization for a Smart Place .............. 80
A.16 SPM Requirement R3.2 - Read Storage Utilization for a Smart Place .............. 81
A.17 SPM Requirement R3.3 - Choose time interval for CPU and Network Metrics .. 81
A.18 SPM Requirement R3.4 - Read Network In and Network Out for and Smart Place .......................................................... 82
A.19 SPM Requirement R3.5 - Read CPU Utilization Average and Maximum values for a Smart Place .......................................................... 82
A.20 SPM Requirement R3.6 - Modify volume size .............................................. 83
A.21 SPM Requirement R3.7 - Modify instance type ............................................ 83
A.22 SPM Requirement R3.8 - Read Volume Read and Write Bytes ...................... 84
A.23 SPM Requirement R3.9 - Read Latencies for a SP machine registered on a Load Balancer .......................................................... 84
List of Source Codes

1. Maven command to Clean and Install a SPM API package. ................................................. 31
2. Cloud provisioning role for Smart Place Manager and Fosstrak. ....................................... 32
3. Fossstrak EPCIS Recipe. ........................................................................................................ 33
4. Fossstrak DB Recipe. ............................................................................................................. 33
5. Fossstrak ALE Server Recipe. ............................................................................................... 34
6. Fossstrak Capture Application Recipe. .................................................................................. 34
7. Wildfly command to start SPM API backend server in debug mode. .................................... 36
8. Cocoa Pods command to install pods into the SPM Xcode project. ...................................... 38
Acronyms

ACW  Amazon CloudWatch.  13, 14, 24, 26, 31
ALE  Application Level Events.  16, 34
AMI  Amazon Machine Image.  13
API  Application Programming Interface.  4, 5, 12, 14, 21, 23, 26, 31, 32, 35, 56, 68, 69
APP  Application.  5, 21, 24, 26, 27, 31, 36, 63, 69
AWS  Amazon Web Services.  9, 12–14, 17, 23–26, 32, 36, 40, 66
C4T  Cloud4Things.  14, 32, 33, 35
CC  Cloud Computing.  7, 9, 11, 13, 14
CPU  Computer Processing Unit.  13, 24, 37, 65
EC  Edge Computing.  1, 7, 8
EC2  Elastic Compute Cloud.  12–14, 24–26, 31, 32, 35, 36, 40
ELB  Elastic Load Balancers.  13
EPC  Electronic Product Code.  14, 15, 17
EPCIS  Electronic Product Code Information System.  16, 17, 33
ERP  Enterprise Resource Planning.  15
FC  Fog Computing.  1, 7, 8, 14
HAL  Hardware Abstraction Layer.  17
IaaS  Infrastructure as a Service.  9, 14
IoT  Internet of Things.  1, 3, 5, 7, 8, 14, 67, 69
LLRP  Low Level Reader Protocol.  16, 17
LXC  Linux Container.  20
MAU  Mobile App Usability.  46
MBaaS  Mobile Backend as a Service.  1, 7, 32
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCC</td>
<td>Mobile Cloud Computing</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-View-Controller</td>
</tr>
<tr>
<td>OS</td>
<td>operating system</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>RAM</td>
<td>Random-access memory</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio-Frequency IDentification</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SP</td>
<td>Smart Places</td>
</tr>
<tr>
<td>SPM</td>
<td>Smart Place Manager</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>UI</td>
<td>UI</td>
</tr>
<tr>
<td>UX</td>
<td>UX</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The **Internet of Things (IoT)**[1] is a system of interrelated computing devices, mechanical and digital machines, objects or people that are provided with unique identifiers. An IoT system has the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction. Nowadays the software architectures are using the Cloud platform layers to integrate IoT. One way to build IoT applications is to structure them as a **Smart Places (SP)**[2]: a physical location instrumented with sensors and actuators that have middleware that allows it to be programmable. The lifecycle of a SP starts when the smart objects that make part of it starts to send events (i.e. Radio-Frequency Identification (RFID) tags reader) which are captured by the readers and sensors that are placed over the SP. Then this sensor information is going to be processed outside the SP itself, through a internet connection from the physical location to the **Cloud**. The result of the data interpreted on the Cloud is retrieved back to the SP where the action over it is going to be performed. In the case of the present work the action is interpreted as the opening of a **access door on a Smart Building**.[3] The mobile applications that integrate IoT or manage the SP have their support servers on the Cloud - **Mobile Backend as a Service (MBaaS)**[5]- and so the common mobile phones limitations such as the battery life, processor and the memory capacity are no longer an issue. The **Smart Place Manager (SPM)** mobile application is playing a role as an interface to interact with the SP and to run the business logic needed behind.

The SPM mobile application and backend SPM API developed in this work play a role as a system that manage the SP systems resources that are running in the Cloud, in order to allow SP to be managed by people that do not have system administration background. This solution allows to create and manage SP in the Cloud, in a simplified and useful way. It can monitor the computing, network, storage and memory metrics within the cloud machines that host these new spaces, reducing the need for machines maintenance.

The solution also allows common management from several mobile devices such as tablets or smart phones which are fed from a common database and in this way adding portability to the integration with other SP systems within **Fog Computing (FC)** and **Edge Computing (EC)** nodes concept thus allowing to choose which Cloud Provider will host the system and the network infrastructure.

1.1 Application Domains

The SP is a space where humans live which is increasingly equipped with sensors and actuators. The most common examples are: heating, ventilation, air conditioning, lighting or access control. These locations can be controlled remotely giving the ability to connect and orchestrate all networked devices
in one place\cite{6}. The SP enable new services to be developed and existing services to be improved for citizens, local businesses and anyone associated with the area; whether through services such as smart lighting, environmental monitoring, waste management, smart traffic, smart housing (domotics), smart parking or simply through access to free public Wi-Fi.

A smart building is a structure that uses automated processes to automatically control the buildings’ operations including heating, ventilation, air conditioning, illumination, thermal comfort, etc. A smart building can be seen as the agglomeration of several SP that uses sensors, actuators and processors in different rooms to collect data and manage it according to a business functions and services. This infrastructure helps owners, operators and facility managers improve asset reliability and performance, which reduces energy use, optimizes how space is used and minimizes the environmental impact of buildings.

Beyond this, as cities and towns become ever larger and more complex there is an increasing demand on traditional services such as transport, energy, healthcare, utilities and the environment like it is shown on picture 1.1.

Figure 1.1: Smart Cities Applications\cite{7}
Smart Cities use IoT devices such as connected sensors, lights, and meters to collect and analyze data. The cities then use this data to improve infrastructure, public utilities and services. The most significant application areas for IoT in Smart Cities are:

**Smart Infrastructure and Buildings**  A smart building can have different hardware, software, sensors, and smart applications for different automated operations, including data network, voice over IP, video distribution, video surveillance, access control, power management and lighting control. For example, the smart grid can be used by smart buildings.

**Smart Transportation**  Smart transportation, also known as an ITS, includes various types of communication and navigation systems in vehicles, between vehicles (e.g., car to car), and between vehicles and fixed locations (e.g., car to infrastructure). Specific examples of smart transportation technology include sensors in vehicles for collision avoidance and anti skidding to increase the safety of the vehicle. A radio frequency identification (RFID)-based toll collection is an example of smart transport technology. With RFID toll collection, drivers need not stop at a physical toll booth, which typically takes time, blocks traffic flow, and requires manpower for toll collection.

**Smart Energy**  The smart energy system consists of the intelligent integration of decentralized sustainable energy sources, efficient distribution, and optimized power consumption. Smart energy includes three independent building blocks that must be stitched together and effectively communicate with each other to form a unified smart energy system. These blocks are Low-carbon Generation, Efficient Distribution and Optimized Consumption.

**Smart Health Care**  The various components of smart health care include emerging on-body sensors, smart hospitals, and smart emergency response. In smart hospitals, various mechanisms for their operation are used including cloud computing, smartphone apps, and advanced data analysis techniques.

### 1.1.1 Example Scenario

Our smart place example domain is digital unlocking access to rooms or floors in a building, that is a Access Control system. The accesses are read through a Smart City, by RFID technology. These systems let a user have access to a facility or room inside a building and also registers that event. They must have a good performance and they use algorithms for reliability, accuracy and speed. As an alternative to biometrics devices for Access Control there is the possibility of using proximity label reading as RFID readers. The reader usually is connected to a third party system that is accepting (and storing) RFID related events and uses these events to trigger actions such as unlocking the door. In this small type of solution, the latencies of the requests in the network must be fast so that the authentication and access opening command order is fast. Thus this factor takes away the hypothesis of this technology becoming inconvenient due to the wait time between placing the smart card near the reader and the access command response for opening.

There are a few things needed to have an operational RFID system in place: a reader, that is connected to (or integrated with) the access door/floor; an antenna, that sends out a radio signal and a tag (or transponder) that returns the signal with data added which in this case, is the smart card token, which is shown on figure 1.2.
Let us have a closer look at access control systems. These systems let a user have access to a facility or room inside a building and also registers that event. They must have a good performance and they use algorithms for reliability, accuracy and speed. As an alternative to biometrics devices for Access Control there is the possibility of using proximity label reading as RFID readers. The reader usually is connected to a third party system that is accepting (and storing) RFID related events and uses these events to trigger actions such as unlocking the door. Thus these systems usually consist of:

- RFID access control cards that are read by
- Access control RFID card readers next to the door, that are connected to
- Access control panels (a physical controller), hardware that is able to open door locks and that is connected to
- An access control management system (software) that manages building access credentials and authorizations

These systems store access control rights for people (or vehicles) and also link those people to something that identifies them. Usually a number that is stored on an access control card. When an access control card (the RFID tag) is shown to the access control reader next to the door (the RFID reader with its antenna), that specific number is sent to the access control panel (a physical controller). The control panel connects with the access control management software (at a server or in the cloud) to check who is connected to that number and if he/she has access to the door that is approached. When the person is authorized an event is stored at the server (for the event log book) and the access control panel is asked to open the door (by telling the physical lock to unlock). In the present work we will use a simulated software solution to generate RFID events corresponding to tag readings and also to their processing and storage modules.

### 1.2 Objectives

The goal of this work is to allow control automated provisioning and installation process for deploying a SP through a mobile application with backend. The application is the frontend and it uses an Application Programming Interface (API) to access the backend servers. The scope focuses on providing an abstraction layer on all the technologies and tools needed to install and manage an IoT application inside a SP through a mobile application. This mobile application sends remote provisioning and deploying orders but also do the server configuration in the Cloud provider, as well as the management of SP throughout its use. Thus it is going to ensure a good user experience but the mobile application is also going to be analyzed for concluding about the energy consumption, in order to minimize its processing and network energy drains on the device’s battery.
It is intended that both the SPM application and the backend SPM API also allow the monitoring of SP server cloud utilization with processing, storage and networking metrics - but also the response time is going be measured between all the three solution parts, which is often decisive in IoT applications running in the cloud which that require low or predictable latency.

So in the present work our main goals are:

1. Provision and manage a RFID-based smart place application and its virtual cloud resources through an iOS Application (APP) - Smart Place Manager Application.

2. To provide the cloud automation and orchestration processes through a SPM API - for the smart place application (environment) that control access to a room through RFID authentication.

1.3 Dissertation Outline

The remainder of this document is organized as follows:

• **Chapter 2. Background** summarizes the relevant work in the field and introduces some key concepts that support our work such as a description of the Cloud, Fog, Edge and Mobile Cloud Computing paradigms, the platform, the mobile application design architecture, the Cloud Architecture and the devops tools used.

• **Chapter 3. Solution Design** covers the high-level structural description of the architecture of the system composed of a mobile API and a backend API.

• **Chapter 4. Solution Implementation** presents the detailed design of the individual components in the system approach adopted for the automated provisioning, monitoring strategies, virtualization technologies, the deployment approaches and the mobile solution integrated.

• **Chapter 5. Evaluation** describes the experiments made to meet the defined objectives and presents an analysis of the obtained results.

• **Chapter 6. Conclusion** summarizes the presented work, presents the main conclusions, some important research points for future work and the main contributions that resulted from this dissertation.
Chapter 2

Background

In this section the concepts and technologies associated with understanding the problem to be solved are presented. It begins by explaining the concepts of Fog Computing (FC), Edge Computing (EC), Cloud Computing (CC) and Mobile Cloud Computing (MCC). This last is a new way of enabling the execution of rich mobile applications on a panoply of mobile devices, with a rich user experience. The recent trends of the FC and EC concepts are emerging more and more in several domains, but also the MCC and MBaaS as there is always the need for the end-user device.

2.1 Concepts and tools

The CC is a new way of providing computing resources and services. It refers to an on-demand infrastructure that allows users to access computing resources anytime from anywhere. In this section we are going to describe its general concepts.

2.1.1 Fog and Edge Computing

The idea of a SP is to have a physical place, which offers a proximity-based service due to existence of tags placed on objects that are relevant. In this way, a SP allows the interaction of a user with a given environment through sensors that provide precious information about this environment, allowing the development of applications that receive and analyze this collected data.

However, sensing all possible data captured by a smart object and then sending to the cloud is less efficient and can also lead to resource wastage (e.g. network, storage). The FC and EC computing paradigms have been proposed to address this weakness by pushing data processing and analytics, to the network edges. The EC is a method of optimizing CC systems by performing processing near the source of the data. The node-oriented view of Internet consists of data centers and clouds at the core. Thus, surrounding this core are smaller web servers and content distribution networks as the next layer, which is in turn followed by the "edge" consisting of individual human-controlled devices such as desktop PCs, tablets, smart phones and nano data center.

On the other hand Fog Computing is a layer between the edge and the cloud which extends the cloud closer to the nodes that produce and act on IoT data. The industrial gateways, routers and other devices with the necessary processing power, storage capabilities and network connection can be fog nodes. With the analysis and pre-processing data on these nodes it is possible for FC to minimize latency and to produce bandwidth usage. This scenario with the cloud, fog and edge is illustrated on figure 2.1 and it is showing the three paradigms work together in order to build an IoT infrastructure for SP that need to feed latency-sensitive applications.
As shown in the figure 2.1, the cloud layer aggregates data summaries from multiple fog nodes and performs deeper analysis on larger data set and send application rules to fog nodes. The fog layer is a transient storage for immediate data and it has the real-time analytics and control based on the application rules provided by cloud layer. Lastly the edge layer captures user interactions, it sends feeds to the fog and it performs intelligent actions over the smart place, based on the real-time control signals from the fog nodes.

The FC and EC real opportunity lies in configuring the nodes and optimizing their performance. The primary difference between an IoT device communicating with a node versus the cloud is that bi-directional communication with a node can take milliseconds while conversing in the same manner with the cloud can take minutes. Hence these are two different solutions for a single problem to optimize cloud performance. Hosting analytics, performance processing, and heterogeneous applications closer to physical centers and control systems, can help enable edge intelligence. They will help us move portions from cloud-based applications closer to devices which use them. It is not easy to figure out what software tasks to remove from the cloud, but the growth of bandwidth-consuming devices may force to take a different approach.

Nevertheless, the edge devices have limited computational capabilities. Due to inherited strengths and weaknesses, neither Cloud computing nor Fog computing paradigm addresses these challenges alone. Therefore, the three paradigms need to work together in order to build a sustainable IoT infrastructure for a SP. In the present work the Fosstrak software that simulates the RFID with simulation and capture (FC) modules, and also filtering and aggregation (EC) modules are running all in one cloud node for ease of implementation and also because the purpose of the work is to build a mobile and backend solution, this way it will not focus on the solution of the Smart Place itself, but rather on the system that creates and monitors it. Thus, it is easy to understand that the distinction between
Fog and Edge paradigms will not be considered.

The usage of smartphones has moreover increased rapidly in various domains, including enterprise, management of information systems, gaming, e-learning, entertainment, gaming, and health care\[15\]. Although the predictions that mobile devices will be dominating the future computing devices, mobile devices along with their mobile applications are still restricted by some limitations such as the battery life, processor potential, and the memory capacity of the storage devices\[15\]. Nowadays, modern mobile devices have sufficient resources such as fast processors, large memory, and sharp screens. However, it is still not enough to help with computing intensive tasks such as natural language processing, image recognition, and decision-making. Mobile devices provide less computational power comparing to server computers or regular desktops and computation-intensive tasks put heavy loads on battery power.

2.1.2 Cloud Computing and Mobile Cloud Computing

The concept $\text{CC}$ describes the hosting and delivery of information and on-demand computing resources on the Internet using a remote network of servers. The alternative is storing, managing or processing data on local servers. Many technology applications that gained popularity as installed software are now preferred as cloud applications due to the ease of use and reduced maintenance. The main is to allow IT departments to focus on their businesses and projects instead of just taking care of their data centers and keeping them working. It is a new concept that aims to provide computational resources as services in a quick manner, on demand, and paying as per usage. Thus the most popular cloud computing benefits are:

- **Scalability** It can dynamically increase or decrease the processing, networking and storage resources needed based on the environment and application requirements.
- **Data Security** Information is stored on a remote server; if a personal computer or internal network fails, no information is lost.
- **Accessibility** Remotely storing a catalog of products or services means that potential customers can access the ecommerce site from virtually anywhere in the world.
- **Pay-Per-Use** Computing resources are measured at a granular level, end users only pay for the resources they require.

The main goal of $\text{CC}$ is to allow IT departments to focus on their businesses and projects instead of just taking care of their data centers and keeping them working. It is a new concept that aims to provide computational resources as services in a quick manner, on demand, and paying as per usage. The CC paradigm is presented in three cloud delivery models: $\text{IaaS}$, $\text{PaaS}$, and $\text{SaaS}$:

- **IaaS** refers to the provisioning of virtual and physical machines on-demand - e.g. storage, processing and network.
- **PaaS** refers to providing platform layer resources and software development frameworks.
- **SaaS** refers to providing applications over the Internet.

As examples of public $\text{CC}$ we can list Microsoft Azure\[1\], Amazon Web Services (AWS)\[2\] and the Google Cloud Platform\[3\]. Both are open and flexible cloud platforms that provide several services to develop, deploy and run web applications and services in cloud data centers. They are considered as

\[1\]https://azure.microsoft.com/
\[2\]https://aws.amazon.com/
\[3\]https://cloud.google.com/
examples of public computing tools, providing users with the three cloud architecture layers like it is shown in Figure 2.2. These services allow the user to use virtualized resources in cloud data centers. Computational clouds implement a variety of service models in order to use them in different computing visions.

![Cloud Computing Layers: IaaS, PaaS and SaaS](image)

Figure 2.2: Cloud Computing Layers: IaaS, PaaS and SaaS.

The cloud options come in three forms: public clouds, private clouds, and hybrids clouds. Depending on the type of data you’re working with, you’ll want to compare public, private, and hybrid clouds in terms of the different levels of security and management required.[16]

A public cloud is one in which the services and infrastructure are provided off-site over the Internet. These clouds offer the greatest level of efficiency in shared resources; however, they are also more vulnerable than private clouds. A public cloud is going to be chosen for this project and it seems an obvious choice when:

- The standardized workload for applications is used by lots of people, such as e-mail.
- There is the need to test and develop application code for collaboration projects, and software development frameworks.
- There are SaaS applications from a vendor who has a well-implemented security strategy.
- There is the need for incremental capacity (the ability to add computer capacity for peak times). The web tier of an application that is mission-critical or latency-intolerant (Custom Auto Scaling groups and automated scripting deployments)
- Any new application that demand is uncertain for, especially for microsites or other interactive properties for marketing and ad campaigns.
- It is going to be done an ad-hoc software development project using a PaaS offering cloud for testing environments.
- Long-term storage of any kind, especially if you are currently hosting physical media that fails often or needs to be replaced.

On the other hand, a private cloud is one in which the services and infrastructure are maintained on a private network. These clouds offer the greatest level of security and control, but they require the company to still purchase and maintain all the software and infrastructure, which reduces the cost savings. A private cloud is the obvious choice when:

On the other hand, a private cloud is one in which the services and infrastructure are maintained on a private network. These clouds offer the greatest level of security and control, but they require the company to still purchase and maintain all the software and infrastructure, which reduces the cost
savings. A private cloud is the obvious choice when:

- The business is all about the data and applications. Therefore, control and security are paramount.
- The business is part of an industry that must conform to strict security and data privacy issues and software development frameworks.
- The software being used requires dedicated infrastructure for compliance.
- The company is large enough to run a next generation cloud data center efficiently and effectively on its own. In this case could exist a need for high performance access to a file system.
- The application has very predictable usage patterns and low storage costs.

Lastly a hybrid cloud includes a variety of public and private options with multiple providers. When spreading out over a hybrid cloud, every single aspect at the business is in the most efficient environment possible. The downside is that it is necessary to keep track of multiple different security platforms and ensure that all aspects of the business can communicate with each other. Here are a couple of situations where a hybrid environment is the best:

- The company wants to use a SaaS application but is concerned about security. The SaaS vendor can create a private cloud just for the company inside their firewall. In this case they provide a virtual private network (VPN) for additional security.
- The company offers services that are tailored for different vertical markets. It can be used a public cloud to interact with the clients but keep their data secured within a private cloud.

An important thing to understand about hybrid environments is that they are only as strong as the integrations that unite them. The performance monitoring, regular testing, and data ingress and egress procedures will reveal future areas of difficulty as well as signal when and how to further evolve into the application.

This work tries to implement this approach in this way such as the mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and mobile computing not just to smartphone users but also to a much broader range of mobile subscribers and device types.

MCC can be seen as the combination of Mobile Computing and wireless networks which bring rich computational resources to mobile users, as well as cloud computing providers. This also tries to bring down the gap between the limited computing resources of the mobile devices and the processing requirements of intensive applications, which are running on them. Hence, MCC at its simplest form uses computational augmentation approaches (computations and data storage are executed remotely instead of on the device) by which resource-constraint mobile devices can utilize computational and data storage resources of varied cloud-based providers.

In this way the concept described before is associated with Mobile Computing Offloading which is the task of sending intensive computational application components to a remote server. Recently, a number of computation offloading frameworks have been proposed with several approaches for applications on mobile devices. These applications are partitioned at different granularity levels and the components are sent (offloaded) to remote servers for remote execution. However, the computation offloading mechanisms are still facing several challenges. The potential of mobile offloading mainly depends on the mobile network technologies such as cellular and WiFi. They determine the viability of mobile offloading, given the amount of data involved. Today, the Wi-Fi technology is able to provide...
high bandwidth connections. However, the data transmission using the cellular network requires a considerable amount of energy from the mobile device as opposed to a Wi-Fi network. In this work we will study the energy consumption of our mobile application.

The Figure 2.4 illustrates the environment that supports computation offloading. In this overview, the mobile device decides to offload a method B to a cloud server or a powerful machine. The cloud here provides the virtual computation resources to run the offloaded components. The powerful machine can be a server or cluster in a computing center, or a computing grid, or a virtual server in the cloud.

2.1.3 Provisioning, deployment and monitoring APIs: the AWS cloud example

The AWS Elastic Compute Cloud (EC2) is one of AWS most well-known services that offers businesses the ability to run applications on the public cloud. Developers can create instances of virtual machines and easily configure the capacity of scaling instances using the EC2 web interface or through integration with Software Development Kit (SDK) over several programming languages. The SDK allows the communication with AWS engine, with its user credentials, in order to process API requests over EC2.

A public API is a particular set of rules (‘code’) and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers.
EC2 also allows users to build apps to automate scaling according to changing needs and peak periods, and makes it simple to deploy virtual servers and manage storage, lessening the need to invest in hardware and helping streamline development processes. This work is a case where a mobile application supplies the management of the SP computational resources (storage, processing and network infrastructure) and it allows to change them according to the SP production application needs which is running over a client node (an EC2 machine).

EC2 setup involves creating an Amazon Machine Image (AMI) which includes an operating system, software, and configurations. That AMI is loaded to the Amazon Simple Storage Service, and it is registered with EC2 at which point users can launch virtual machines as needed.

A number of benefits and features draw developers to EC2 for CC, among these are:

- **Responsiveness** to changing capacity requirements: The easy scaling of EC2 eliminates development obstacles that occur when applications require more resources.
- **Flexibility in configurations** Users can choose memory size, Computer Processing Unit (CPU) and boot partition size optimized for the operating system (OS) they choose and software development frameworks.
- **Integration** EC2 can integrate with other AWS services, such as RDS, SimpleDB, and SQS.
- **Precise control** Users get administrative access to their instances, can stop and start instances while retaining boot partition data, and can access console output for the instance.
- **Security** Users can control which instances remain private and which have internet exposure. EC2 leverages Amazon Virtual Private Cloud for security, and businesses can connect their secure IT infrastructure to resources in Amazon Virtual Private Cloud.
- **Cost** Among a number of pricing options, EC2 offers affordable hourly rates.

**AWS Java SDK**

The EC2 SDK for Java helps make AWS applications and services available to web browsers across many devices and OS. The AWS SDK for Java and other platform or language-specific software development kits are part of Amazon platform-agnostic approach in their quest to reach all browsers and devices with their Web store, Amazon Marketplace.

The AWS SDK for Java comes with:

- Java APIs (application programming interfaces) for connection to important AWS components such as Amazon EC2, Amazon S3, Amazon CloudWatch, etc.
- Class libraries that eliminate the need to deal with lower-level code for inner workings like networking redundancy and error checking.
- Code samples with common usage examples to build applications.
- Documentation and reference material for the workings of the AWS SDK for Java API.

**Amazon CloudWatch**

Amazon CloudWatch (ACW) is a component of AWS that provides monitoring for AWS resources and the customer applications running on the Amazon infrastructure. ACW enables real-time monitoring of Amazon resources such as EC2 instances, volumes, Elastic Load Balancers (ELB) and Relational Database Service database instances. The application automatically provides metrics for CPU utilization, latency, and request counts. Users can also stipulate additional
metrics to be monitored, such as memory usage, transaction volumes or error rates.

Users can access ACW functions through an API command-line tools, one of the AWS SDK or the AWS. The ACW interface provides current statistics that can be viewed in graph format. Users can set notifications (called alarms) to be sent when something being monitored surpasses a specified threshold. The application can also detect and shut down unused or underused EC2 instances.

2.2 Related Work

The previous work Cloud4Things (C4T) [20] is an architecture that uses the cloud platform layers to integrate IoT and CC. The proposed architecture is an online platform which accommodates PaaS, SaaS and allows system integrators and solution providers to leverage the complete application infrastructure for developing, operating and composing applications and services.

It explored the deployment of IoT applications for smart warehouses based on the RFID technology with two different approaches to deploy: one based in a traditional cloud deployment approach (cloud-based) and other according the FC platform (fog-based). It focuses to determine if a cloud-based approach is able to meet the low-latency requirements of many IoT applications, since that low-latency is an essential requirement of IoT applications.

The first solution consists of a traditional architecture in which all middleware is in the cloud. The second solution is based on non-cloud computing, a virtual platform that is close to a smart location and provides the previous resources between floorless electronic gadgets and a cloud provider. Thus it compares two solutions to provide the necessary resources in terms of processing, networking and storage as well as to exemplify a production environment, approaching a client / server architecture for deployment of an application that captures and processes signals RFID on the IoT context.

In this work an automated system for configuring virtual resources is created in the AWS cloud through the Chef devops tool that allows a client / server architecture in which it facilitates the provisioning of servers through scripts, over Secure Shell (SSH) connection with the client public key exchanged with the server. The solution also uses Docker to manage the RFID application images that allow the capture of RFID signals (by Fosstrak simulation platform with API for abstraction and collected information storage). It also has its own database as well as the server that allows to run the middleware that treats information from a message to the report of the order given in physical space.

It also uses several software like the Electronic Product Code (EPC) global framework which supplies standardized interfaces that isolate the hardware manufacturers from business applications, the Fosstrak platform that consists on an open source RFID middleware that implements GS1 network standards, ie it allows to control a processed information in RFID operation and store the relevant data in the database to be processed by the middleware that is the control of the smart place.

Finally, Marcus is also able to measure some relevant metrics to prove the benefits of a solution with Part-Time Event Cycle compared to the baseline EC in order to evaluate network performance, which is the factor that matters most in SP which will be the time difference between the event that captures the RFID information and its result after analysis by a rule engine that is server-side, but in this case, in virtualized hardware near the network, without being exclusively processed in the cloud. It also evaluates the cloud storage capacity over load. In this previous work, the application domain characteristics are
similar to the Smart Office/Home and it is based on the RFID technology and the EPC Framework.

2.2.1 EPC Framework

The RFID middleware is the component of a RFID system that sits between the low level components - e.g. readers and tags - and the business client application - e.g. Enterprise Resource Planning (ERP) systems, like it was described in Section 1.1.1. The next paragraphs describe the EPCglobal, a framework that provides standardized interfaces that isolates hardware vendors from business applications, and Fosstrak, a open-source RFID middleware platform that implements the GS1 EPC Network standards.

2.2.2 GS1 EPCglobal Architecture

GS1 is an organization that is responsible for the development and maintenance of standards for supply chain. One of the standards developed by GS1 is the EPC which is an unique serial identifier for RFID tags. GS1’s subsidiary EPCglobal Inc. created the EPCglobal Architecture Framework, that currently is the standard for RFID platforms. Figure 2.5 presents a high-level architecture of the EPCglobal framework that shows its main interfaces and roles.

![Figure 2.5: GS1 EPCGlobal Architecture Framework.](http://www.gs1.org)
The framework has a set of standardized interfaces that enables the interchange of information between entities. In the context of our work, the most relevant components of the framework architecture are:

- **Reader Interface** provides the interfaces that must be implemented by the RFID readers. The Low Level Reader Protocol (LLRP) standard provides interfaces that allows the control of all the aspects of RFID reader operation.
- **Filtering & Collection** is the module that coordinates the RFID readers that are in the same physical space and also abstracts the readers from the upper layers. It allows the execution of read and write operations on tags. Furthermore, it is responsible for filtering, aggregating and grouping the raw tag data when requested.
- **Application Level Events (ALE) Interface** defines the control and delivery of filtered and collected data from the Filtering & Collection module to the Electronic Product Code Information System (EPCIS) Capturing Application. The ALEs are a selection of the events that are meaningful for the client applications.
- **EPCIS Capture Application** supervises the operation of the lower EPC layers, and provides business context based on information involved in the execution of a particular step of a business process.
- **EPCIS Repository** is the module where all the business events generated by the EPCIS Capturing Applications are stored to later be accessed by the EPCIS Accessing Application. The EPCIS Query Interface defines how client applications can retrieve information from the repository.

### 2.2.2.1 Fosstrak Platform

The Free and Open Source Software for Track and Trace (Fosstrak) is an EPCglobal Network compliant RFID software platform that was developed by Floerkemeier et al [21]. Figure 2.6 presents the architecture of the Fosstrak platform.

![Figure 2.6: Fosstrak architecture.](https://example.com/fosstrak-architecture.png)

For more information about the standards of the EPCglobal Framework, the full documentation is available at [http://www.gs1.org/gs1-architecture](http://www.gs1.org/gs1-architecture)
The Fosstrak platform is composed of three modules that implements the corresponding roles in the EPC Network: Reader Module, Filtering and Collection Middleware Module and EPCIS Module. For our work the relevant modules of the platform are:

- **Filtering & Collection Server** is the module responsible for filtering and collecting data from RFID readers. To communicate with the readers, the module uses the LLRP standard for LLRP compliant readers and uses the Fosstrak Hardware Abstraction Layer (HAL) for unsupported readers. The module internally abstracts the readers as LogicalReaders instances that are defined and configured through a LRSpec document, as defined by EPCglobal. Fosstrak also implements the Event Cycle, that is an interval of time during which tags are collected. The output of an Event Cycle is the ECReport document that is sent to the Capturing Application.

- **Capturing Application** is part of the EPCIS module. This module is responsible for transforming uninterpreted events received on the ECReports into meaningful business events. Regarding its implementation, the Capturing Application is built on top of the Drools engine where rules can be specified in the form of: “when” something happens, “then” do “this”. Unfortunately, the rules are static and once defined they can not be updated in runtime.

- **EPCIS Repository** provides an EPCglobal-certified EPCIS Repository, which means that all Fosstrak EPCIS modules and interfaces are compliant with the EPCglobal standard. This module provides persistence for EPCIS events. For storing new events the module provides the capture interface and the query interface for retrieving historical events is provided. Furthermore, the module provides two EPC Network-conformant interfaces to a relational database (currently MySQL).

### 2.2.2.2 Chef

Chef is a configuration management tool that allows to describe the infrastructure as code. In that way it is possible to automate how the infrastructure is built, deployed and managed. Users describe the system resources and their desired state and the configuration management tool is responsible for enforcing the desired state. This tool allows the provisioning automation of the physical and virtual machines, perform dependency management of software components and it also performs the automation of management tasks. It deals with machine setup on physical servers, virtual machines and in the cloud.

Chef works with three core components: the Chef server, workstations, and nodes. The Chef server is the hub of Chef operations, where changes are stored for use. Workstations are static computers or virtual servers where all code is created or changed. Finally, nodes are the servers that need to be managed by Chef, hence they are the machines that changes are being pushed to, generally a fleet of multiple machines that require the benefits of an automation program.

The tool was built from the ground up with the cloud infrastructure in mind. With Chef, it is possible to dynamically provision and de-provision the application infrastructure on demand to keep up with peaks in usage and track. For instance, the knife command has a plugin for provisioning cloud resources across several cloud providers: AWS, Google Compute Engine and Openstack.

It helps solve this problem by treating infrastructure as code. Rather than manually changing anything, the machine setup is described in a Chef recipe. Collections of recipes are stored in a
cookbook. One cookbook should relate to a single task, but can have a number of different server configurations involved (for example a web application with a database, will have two recipes, one for each part, stored together in a cookbook).

There is a Chef server which stores each of these cookbooks and as a new chef client node checks in with the server, recipes are sent to tell the node how to configure itself. The client will then check in every now and again to make sure that no changes have occurred, and nothing needs to change. If it does, then the client deals with it. Patches and updates can be rolled out over the entire infrastructure by changing the recipe. So there is no need to interact with each machine individually. When there is only a single machine this is fairly simple. When there are five or ten servers, it is still possible to do this manually, but it may take all day. However, when the infrastructure scales up into the thousands we need a better way of doing things.

Recipes and cookbooks are the heart of the configuration management. They are written using the Ruby programming language, however, the domain specific language used by Chef is designed to be able to be understood by everyone. As the configuration is just code it can be tested and it can be version controlled. This means that there is less downtime, more reliable services and less stressed people on both the dev and ops sides.

The Chef architecture is showed in Figure 2.7. In the present work it was used the Fosstrak Chef recipes solution from the previous work and a new role and cookbooks were created for CentOS the SP operating system chosen.

![Figure 2.7: Chef architecture](https://centos.org/)
2.2.2.3 Docker

Docker [22] is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

In a normal virtualized environment, one or more virtual machines run on top of a physical machine using a hypervisor. Containers, on the other hand, run in user space on top of operating systems kernel. It can be called as OS level virtualization. Each container will have its isolated user space and you can run multiple containers on a host, each having its own user space. It means it can run different Linux systems (containers) on a single host. In the figure 2.8 it is shown that several applications can run on these containers that inherit the SP machine’s operating system.

![Docker architecture](image)

Figure 2.8: Docker architecture
Thus Docker is an open source project to pack, ship and run any application as a lightweight container. Since Docker containers are based on Linux Container (LXC), it can run anywhere from a laptop to a cloud instance. Another benefit that Docker platform provides is the Docker Hub service, a public repository that stores Docker images that are used to create the containers.

This technology is like a virtual machine but unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the host system that they’re running on and only requires applications be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application.

It was used the previous work Docker containers which are a containerized Fosstrak modules solution, grouped in four containers: fosstrak_epcis, fosstrak_db, fosstrak_capture and fosstrak_ale_server.

---

13 https://www.docker.com/
14 Initially, Docker required that the physical machine where the containers will be created is running a Linux kernel. Currently, Microsoft launched Windows Server Containers, an OS level virtualization mechanism where it is possible to perform the management of containers through Docker.
15 https://hub.docker.com/
16 https://fosstrak.github.io
Chapter 3

Solution Design

The importance of mobile phones is increasing in our daily lives. This is so because there is ongoing transformation in that mobile phones are no longer the ordinary communication device it used to be. It has become the point of attention for individuals and businesses alike, due to the various incredible features and opportunities that mobile phones offer. The cumulative progress of mobile technology, the availability and access to high speed internet and the remarkable communicative interface in these devices results into a whole level of new and innovative experience on mobile computing. This is made possible through the development of mobile applications.

The solution presented on this work is an automated \textit{provisioning} and \textit{monitoring} system composed of a mobile \texttt{APP} and a backend \texttt{API} in order to achieve benefits in computer efficiency, ability, and reliability. On the other hand, orchestration\cite{2} is also taken in consideration since it automates the execution of a workflow process. A provisioning process may be comprised of multiple tasks and involve multiple systems since a mobile application need the support of several web servers, app and database.

In this section we will describe how the solution was designed taking in account the business perspective and the system perspective. The business perspective (conceptual) view describes the system with informational entities and business processes that define the requirements. The system perspective (logic)view describes the formal or semiformal system, but still in an abstract way. That is, the specification of the solution is produced. After the perspectives analysis, it will be described how the solution was thought with regard to the description of the backend \texttt{API} and the mobile \texttt{APP}.

3.1 Business Perspective

The requirements of an information system can be divided into two types: functional or non-functional. The \textit{functional requirements} describe what the system should do. The non-functional requirements describe the qualities the system should present in its operation.

\textbf{User functional requirements} From a user’s perspective this system should be a tool to support the provision and management of a SP. It must first allow the user to choose the cloud platform and the type of smart place they want to provision in it. The user must then be able to choose the virtual resources and to create the instances in the previously chosen cloud provider.

Once created, the user should be allowed to monitor and manage the smart place, which include functions to list the created instances, delete instances, start/stop instances and finally obtain relevant
information for each instance such as processing, storage and latency metrics. With this metrics
the user will have the option to change the virtual resources of the smart place during its operation,
performing a vertically scaling.

**Functional requirements** The product requirements and the functionalities of the system are grouped
are grouped into three sets: R1 for Create Smart Place, R2 for Actions and Details on Smart Places
and R3 for Monitoring a SP. The first set of requirements includes requirements R1.1 and R1.2 that
chooses the SP creation type and name, the requirements R1.3 and R1.4 that choose the Cloud region
and provider, and lastly the requirements R1.5 and R1.6 that chooses CPU and storage size. The
second set of requirements includes requirements R2.1, R2.2, R2.3 and R2.4 let the application perform
the Create (R1), Terminate, List, Start and Stop actions over the SP. The R2.5 requirement provides
SP information of the Smart Place about its name, public ip, CPU type, memory size, region and cloud
provider.

The third set of requirements includes requirements R3.1 and R3.2 which let the application perform
the reading of memory and storage utilization for a SP. The requirements R1.3, R1.4 and R1.5 allow to
choose the time interval to read CPU utilization, Network In/Out (Bytes) and also to visualize them. The
requirements R1.6, R1.7 modify SP instance volume size and CPU type. Finally, the requirements R3.8
and R3.9 are for reading SP storage volume read and write bytes and to read the total time elapsed, in
seconds, from the time the load balancer sent the request to a registered SP instance until this instance
started to send the response headers.

1. **R1 - Create a SP**
   1.1. R1 - Create a SP
   1.2. R1.1 - Choose the SP type.
   1.3. R1.2 - Choose the SP name.
   1.4. R1.3 - Choose the SP Cloud Provider.
   1.5. R1.4 - Choose the SP Region.
   1.6. R1.5 - Choose the SP CPU.
   1.7. R1.6 - Choose the SP storage size.

2. **R2 - Actions and Details over Smart Places.**
   2.1. R2.1 - Create SP
   2.2. R2.2 - Terminate SP
   2.3. R2.3 - List SP
   2.4. R2.4 - Start and Stop SP
   2.5. R2.5 - Read SP Information.

3. **R3 - Monitoring Smart Places.**
   3.1. R3.1 - Read memory utilization for a SP
   3.2. R3.2 - Read storage utilization for a SP
   3.3. R3.3 - Choose time interval to read CPU utilization and Network In/Out(Bytes) for a SP
3.4. R3.4 - Graphics - Network In/Out for a SP

3.5. R3.5 - Graphics - CPU Utilization for a SP

3.6. R3.6 - Modify SP volume size.

3.7. R3.7 - Modify instance type.

3.8. R3.8 - Graphics - Read VolumeReadBytes and VolumeWriteBytes for a SP

3.9. R3.9 - Read Latencies for a SP

The general and detailed description of the functional requirements of the system are listed on the tables in Appendix A (SPM Solution Requirements) section.

### 3.2 System Perspective

Figure 3.1 gives us a global idea of the system where we can see the main actor of the system, named the SP owner responsible for managing the smart place.

The SPM backend API that is illustrated in the center of the figure is running on a cloud machine on the same Cloud Provider used for deploying the SP. This API also allows customization for multiple Cloud providers, but the implemented one was AWS. Below the developed system architecture we can see an example of a simulated RFID SP that communicates with the backend SPM API and the Cloud nodes.
3.3 Technology Choices

The API server is configured with automation and orchestration tools for the bootstrap and deploying process which are Chef (for bootstrap the new SP node), Docker (for SP APP containerization and image creation) and the tools needed from the cloud provider implemented AWS (Java EC2 API, ACW Load Balancer).

The experiments and tests were conducted on the AWS cloud provider with the smart places being created over EC2 machines running centOS 7 operating system. The experiments on the SPM API server were performed with a EC2 machine running a "t2.medium" instance which corresponds to a 2 virtual CPUs machine with 4 GB of Random-access memory (RAM) over a centOS 7 machine also on AWS EC2. The backend API was developed in Java 8 and it is running on the WildFly application server.

3.4 Designing of Smart Place Manager API

The Smart Place Manager API solution will reproduce the concept of an API for Mobile Backend As a Service, the SPM API that is going to be the backend of the Smart Places creation, management and monitoring. In this section we propose a solution based on a RESTful Java API - SPM API - that will automate and orchestrate the provisioning and the monitoring flows invoked by the iOS mobile application in order to create EC2 instances and the consequent build and deploy of the RFID-based APP.

The API server is configured with automation and orchestration tools for the bootstrap and deploying process which are Chef (for bootstrap the new SP node), Docker (for SP APP containerization and image creation) and the tools needed from the cloud provider implemented AWS (AWS EC2, Java EC2 API, ACW AWS Load Balancer).

3.4.1 Cloud automation and Cloud orchestration

The mobile APP is going to invoke web services requests to a RESTful Java SPM API that is running on a SPM backend server on AWS which will manage the creation, listing and monitoring of the SP workflows. This API is going to process the calls from the mobile application and forward them to the Chef server, EC2 or to ACW through the AWS user access credentials and keys, depending on the action requested on the mobile application. Hence the Cloud automation will consist on the EC2 instances creation over AWS. This automation consists on forwarding the requests made by the mobile application to the AWS which has web services exposed to provide instances interaction (through AWS EC2 SDK) and also instances monitoring (AWS CloudWatch SDK). The proposed network architecture between the iOS application and the AWS is shown on the figure 3.2.

The figure 3.2 shows the operation of the proposed system. The user on the left is responsible for managing the machines of the smart place for access control of the building has a mobile interface which allows him to manage the Smart Place and monitor it. Then the user actions on the mobile application go through an internet connection by webservies and are received and interpreted by an backend API.

---

1 https://www.chef.io/
2 https://www.docker.com/
3 https://www.chef.io/
4 https://www.docker.com/
5 https://aws.amazon.com/sdk-for-java/
6 https://aws.amazon.com/cloudwatch/
that will interact with the cloud provider, storing information about \textit{SP} in its own database, and returning the response to the mobile application.

The machine provisioning automation process is based on the Chef\textsuperscript{7} server-client architecture. A list of recipes, cookbooks and data bags will the pushed to the new client instance (for example a new node created on AWS for the RFID-based app) with the \textit{knife} server command. The automated deploy will be based on the approach of docker containers running the \textit{Fosstrak} software stack from the previous work on each of the new instances created.

The cloud orchestration consists on the capacity to choose and to change the Docker containers that are running on each instance. This ability gives access to update the containers that are running on the Smart Place and orchestration also includes changing the policies and the server software that is installed on each EC2 instance. This is also made through the chef client/server architecture which will act as a repository where the chef client is running and from where it will send the push order to the Hosted Chef server to update the recipes, cookbooks and data bags that are running on each instance created. This functionality is not implemented on the current solution but it could be a future work.

Thus the operation of the proposed system consists on an user that is managing the machines of the smart place for access control of the building and he has a mobile application with an interface to the Smart Places which allows him to manage the a \textit{SP} and to monitor it. Then the user actions on the mobile application go through an internet connection by web services and are received and interpreted by a backend API that will interact with the cloud provider, storing information about \textit{SP} in its own database, and returning the response to the mobile application.

\footnote{https://www.chef.io/}
3.4.2 SPM Backend server

The SPM API is running on the SPM backend server which is responsible for the Cloud Automation (CM) and Cloud Orchestration (CO) processes. The server plays several roles that make part of a solution like this and it is is shown on the figure 3.3. By observing this figure we can see that the backend server will orchestrate the provisioning and monitoring of the SP with requests from the mobile APP to the WildFly server that hosts the SPM API. After that the requests are forwarded either to start a new process of SP node creation (bootstrap) or either for exchanging monitoring and interaction orders with EC2 and ACW API from the AWS cloud provider.

![Diagram of Automatic Provisioning and Monitoring Workflow.](image)

Figure 3.3: Technological view of Automatic Provisioning and Monitoring Workflow.

It should also be noted that the SPM API has its own database and APP server, hence we will not use the amazon solution for database and APP servers, for reasons of autonomy. In a real system it was advisable to use the database inserted in the Cloud Provider due to performance, networking and high availability issues.

We can also check in the last figure that the Chef Client and the Docker container software solution (Fosstrak) is part of SPM backend server. In reality this work implemented the public Chef server (Hosted Chef) as well as the public repository of Docker (DockerHub). In this way these two repository will only be part of the backend server in a future customized solution for such.
3.5 Designing of Smart Place Manager App

In this section it will be shown how the Smart Place Manager (SPAPP) was designed and which design pattern was chosen. The network layer on the mobile application will describe the web services implementation and this is part of the solution implementation which will be described in the next chapter as well as the app screens developed.

3.5.1 MVC Design on Swift

The Model-View-Controller (MVC) design pattern assigns objects in an application one of three roles: model, view, or controller. The pattern defines not only the roles objects play in the application, it defines the way objects communicate with each other. Each of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries.

MVC is central to a good design for a Cocoa application which is the case. The benefits of adopting this pattern are numerous since many objects in these applications tend to be more reusable, and their interfaces tend to be better defined. The applications having an MVC design are also more easily extensible than other applications. Moreover, many Cocoa technologies and architectures are based on MVC and require that your custom objects play one of the MVC roles.

- **Platform Usability** A common frustration for mobile users is not having an app that works for their specific model of smartphone. Android and iOS, for example, are two different platforms. You can't simply clone your iOS app for Android and vice versa. Android comes with an entirely different set of programming, design, and user interface considerations. For example, navigation for each platform differs quite significantly. iOS for one, does not have a back button like Android.

- **View** The View layer is the face of the application because it is an object that the user can see in a user interface (UI). Its classes are typically reusable, since there isn't any domain-specific logic in them. For example, a UILabel is a view that presents text on the screen and its easily reusable.

- **Controller** The Controller controls all logic that goes between the View and the Model as also as it transports messages between the both of them, typically via the delegation pattern. In the ideal scenario, the controller entity won't know the concrete view its dealing with. Instead, it will communicate with an abstraction via a protocol. A classic example is the way a UITableView communicates with its data source via the UITableViewDataSource protocol.

The two types of flow on the next diagram are the following:

- the top flow from View Controller Model happens when a user interaction in the View causes the data in the Model to change.
- The bottom flow from Model Controller View happens when the data in the Model changes and the View needs to update accordingly.

**The View Layer.** When a user interacts with the app, they are interacting with the view layer. A view object is an object in an application that users can see. A view object knows how to draw itself and can respond to user actions. A major purpose of view objects is to display data from the applications model objects and to enable the editing of that data. Despite this, view objects are typically decoupled from model objects in an Model-View-Controller (MVC) application. Because you typically reuse and reconfigure them, view objects provide consistency between applications. Both the UIKit and AppKit frameworks provide collections of view classes, and interface builder offers dozens of view objects in its Library.

https://developer.apple.com/library/content/documentation/Swift/Conceptual/BuildingCocoaApps/WorkingWithCocoaDataTypes.html
View objects learn about changes in model data through the applications controller objects and communicate user-initiated changes for example, text entered in a text field through controller objects to an applications model objects.

The view shalt not contain any business logic and in code terms it is organized with:
- UIView subclasses that range from a basic UIView to complex custom UI controls.
- A UIViewController. Since a UIViewController is strongly coupled with its own root UIView and its different cycles (loadView, viewDidLoad), I personally consider it to be part of this layer, but not everyone agrees.
- Animations and UIViewController transitions.
- Classes that are part of UIKit/AppKit, core animation and core graphics.

**The Controller Layer.** A controller object acts as an intermediary between one or more of an applications view objects and one or more of its model objects. Controller objects are thus a conduit through which view objects learn about changes in model objects and vice versa. Controller objects can also perform setup and coordinating tasks for an application and manage the life cycles of other objects.

A controller object interprets user actions made in view objects and communicates new or changed data to the model layer. When model objects change, a controller object communicates that new model data to the view objects so that they can display it.

**The Model Layer.** Model objects encapsulate the data specific to an application and define the logic and computation that manipulate and process that data. A model object can have to-one and to-many relationships with other model objects, and so sometimes the model layer of an application effectively is one or more object graphs. Much of the data that is part of the persistent state of the application (whether that persistent state is stored in files or databases) should reside in the model objects after the data is loaded into the application.

Because model objects represent knowledge and expertise related to a specific problem domain, they can be reused in similar problem domains. Ideally, a model object should have no explicit connection to the view objects that present its data and allow users to edit that data it should not be concerned with user-interface and presentation issues.

User actions in the view layer that create or modify data are communicated through a controller object and result in the creation or updating of a model object. When a model object changes (for example, new data is received over a network connection), it notifies a controller object, which updates
the appropriate view objects. The following components to also be part of the model layer:

- Network Code. The shape should be something like this. Ideally, you'd only use a single class for network communication across your entire app.
- Persistence Code. You would implement this with Core Data or simply by saving an NSData blob directly to disk.
- Parsing Code. Any objects that parse network responses and the like should be included in the Model layer as well.

While the model objects and the parser are domain-specific, the network code will be highly reusable. The controller will then use all the elements in your model layer to define the flow of information in your app. The messages can flow from one component to the other through four ways: the View tells the Controller when a user interaction takes place; the Controller updates the View when the data changes; the Controller updates the Model when the data changes and the Model notifies the Controller when data changes.
Chapter 4

Solution Implementation

The solution implementation is divided on the specifications for both SPM APP and API. The implementation of the both parts is described on the next two sections.

4.1 Implementation of Smart Place Manager API

The present work is a RESTful Java API which implements the AWS Java SDK, which is a library that is used on the SPM API and it is responsible for calling the classes necessary to communicate with the EC2 and ACM APIs. It also is responsible to communicate with the Chef server to bootstrap a new node on the cloud which will further by Chef recipes create and deploy a new SP.

In the backend SPM API project, the following libraries were used:

- aws-java-sdk (1.11.235) [https://aws.amazon.com/sdk-for-java/](https://aws.amazon.com/sdk-for-java/)
- com.googlecode.json-simple (1.1.1) [https://github.com/danielgindi/Charts](https://github.com/danielgindi/Charts)
- javax.ws.rs (2.0) [https://docs.oracle.com/javaee/7/api/javax/ws/rs/package-summary.html](https://docs.oracle.com/javaee/7/api/javax/ws/rs/package-summary.html)
- org.wildfly (10.1.0.Final) [https://mvnrepository.com/artifact/org.wildfly](https://mvnrepository.com/artifact/org.wildfly)
- log4j (1.2.17) [https://logging.apache.org/log4j/2.x/](https://logging.apache.org/log4j/2.x/)

These libraries were installed in the Eclipse Java project with Maven which generated a pom.xml file with these dependencies. The Maven is a project management and a dependency manager software.

The SPM was developed with a Swift 4 Xcode Cocoa project. This tool can also help to scale the project by just installing its dependencies and running the following command over the Eclipse project folder, which are clean cleans up artifacts created by prior builds and install which installs the project SPM API package into the local repository, for use as a dependency in other projects locally.

```
1 mvn clean install
```

Listing 1: Maven command to Clean and Install a SPM API package.

---

1 https://aws.amazon.com/sdk-for-java/
2 https://cocoapods.org/
The SPM API was developed in Java 8 on Eclipse Oxygen and it uses Maven as a build management tool for defining compilation, packaging, Classpath management among other sorts of tasks that are required to build Java projects. The API project was integrated with several Maven dependencies including the AWS EC2 SDK and Log4j jars.

On the next sections it is described all of the components that make part of the system composed of the dev machine with the API running and playing a MBaaS role. The API main functions will also be listed as the network architecture that is behind all this solution.

4.1.1 Chef

The Chef automation and orchestration mechanism starts bootstrapping a new node on the backend api when the Create request arrives from the application. Then this process has a time period of two and a half minutes in average, in order to bootstrap the new node.

The implementation of the orchestration tool Chef was based in the necessary Role to install the Docker software on the machines of the SP and it also pulls the containers necessary to deploy the containerized Fosstrak solution, after installing Docker, as it is shown on the code 2.

```json
{
    "name": "fosstrak",
    "default_attributes": {},
    "run_list": [
        "recipe[docker-engine]",
        "recipe[fosstrak::fosstrak_db]",
        "recipe[fosstrak::fosstrak_epcis]",
        "recipe[fosstrak::fosstrak_capture]",
        "recipe[fosstrak::fosstrak_ale]"
    ]
}
```

Listing 2: Cloud provisioning role for Smart Place Manager and Fosstrak.

This role installs the docker-engine container on the client to install Docker itself and also the 4 containers corresponding to the Fosstrak Docker containerized solution.

\[\text{http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html}\]
\[\text{https://projects.eclipse.org/releases/oxygen}\]
\[\text{https://maven.apache.org/}\]
\[\text{https://docs.chef.io/roles.html}\]
Then it was made four recipes for the four containers that make part of the previous solution. The **EPCIS** and the **EPCIS DB** recipes are shown on the following code images 3 and 4.

These recipes configure the two containers mentioned above for the provisioning of the **EPCIS** repository as well as the associated database. In the first one the order of creation of container is given from the images that were made pulled in the role. Next, each container is configured with regard to name attribute, detach propriety, the links between both as well as their ports that are exposed to each other, and also the associated image repository for each.
The recipes that correspond to the ALE Server and Capture modules are shown on the next two following code images 5 and 6.

Listing 5: Fossstrak ALE Server Recipe.

```bash
# Pull latest image
docker_image 'cloud4things/fosstrak_ale'

# Run container exposing port 8081 and remaping to port 8080
docker_container 'cloud4things/fosstrak_ale' do
  container_name 'fosstrak_ale'
  port [8081:8080]
  repo 'cloud4things/fosstrak_ale'
  restart_policy 'always'
end
```

Listing 6: Fossstrak Capture Application Recipe.

```bash
# Pull latest image
docker_image 'cloud4things/fosstrak_capture'

# Run container exposing port 9999
docker_container 'cloud4things/fosstrak_capture' do
  container_name 'fosstrak_capture'
  links 'fosstrak_epcis:epcis'
  port [9999:9999]
  repo 'cloud4things/fosstrak_capture'
  restart_policy 'always'
end
```
The cookbooks that are part of the solution are the docker-engine(0.2.3) and Docker(2.17.0) for installing the Docker solution and Fosstrak (0.3.1) which contains the four recipes described before for the software stack solution with Fosstrak.

The Chef server that is used is Hosted Chef which is a public server that offers capability for hosting Chef nodes on a user account. When the server receives the order for bootstrapping, the SPM API runs Java process to execute the bootstrap process over the AWS cloud to create a new node on an EC2 machine. Hence this new SP becomes a Chef client.

For reasons of simplicity it is used a tool called Knife Plugin to facilitate the bootstrap process in order to get the exchange of the private keys between the Hosted Chef server and the new EC2 client instances.

### 4.1.2 Docker

The Docker is used on the previous work as the Fosstrak containerized solution to simulate a RFID SP. In the present work this experience is followed and so the Docker containers that are being used are: fosstrak_ale, fosstrak_capture, fosstrak_db, and fosstrak_epcis.

**Docker Registry** Docker registry is a service that is storing your docker images. Docker registry could be hosted by a third party, as public or private registry, like one of the following registries:

- Docker Hub which is a Docker Registry hosted by Docker
- Quay
- AWS Container Registry
- Create a docker registry

One of the things that makes Docker so useful is how easy it is to pull ready-to-use images from a central location, Dockers Central Registry. In this case the collection of tagged images as a Docker repository named C4T that stores the images from the Fosstrak RFID software stack are going to be pulled during the Chef bootstrap process from the DockerHub public repository into the new node with the help of a Chef recipe. The Docker public repository is a collection of different Docker images with same name, that have different tags. The tag is an alphanumeric identifier of the image within a repository.

### 4.1.3 Wildfly

The WildFly is a Java application server which runs the spapi.war and thus expose the spmapi to the internet on the addresses with the endpoints defined on the following URL and web context http://localhost:8080/aws/rest. In the current project the api is running over a EC2 machine and so the localhost string is replaced by the EC2 instance DNS The compilation of SPAPI Java Project results on a spmapi.war file that is going to be deployed at the WildFly application server. The command used to launch the server on debugging mode is:

---

7 https://supermarket.chef.io/cookbooks/docker-engine
8 https://supermarket.chef.io/cookbooks/docker
9 https://hub.docker.com/r/cloud4things/fosstrak_ale/
10 https://hub.docker.com/r/cloud4things/fosstrak_capture/
11 https://hub.docker.com/r/cloud4things/fosstrak_db/
12 https://hub.docker.com/r/cloud4things/fosstrak_epcis/
13 https://docs.docker.com/datacenter/dtr/2.4/guides/
Listing 7: Wildfly command to start SPM API backend server in debug mode.

4.1.4 SPM API Endpoints

The functions performed by the backend API try to answer for the functional requirements of the section 3.1. So it is composed of exposed Endpoints that can be called on the Endpoints described on the next section.

Endpoints The available SPM API RESTful webservice /GET calls or endpoints on the localhost are the following that are listed below with the parameters filled. If the SPM API backend server is running on the cloud AWS, the EC2 machine IP must be defined in spite of localhost.

- **Create SP** [http://localhost:8080/spm-api/rest/aws/createSmartPlace?smartPlaceName=Example&instanceType=t2.small&instanceImage=ami-5f76b626&instanceSecurityGroupId=default&smartPlaceType=Authentication-RFID&availabilityZone=eu-west-1&instanceSize=8&randomRequest=166724340&cloudProvider=AWS](http://localhost:8080/spm-api/rest/aws/createSmartPlace?smartPlaceName=Example&instanceType=t2.small&instanceImage=ami-5f76b626&instanceSecurityGroupId=default&smartPlaceType=Authentication-RFID&availabilityZone=eu-west-1&instanceSize=8&randomRequest=166724340&cloudProvider=AWS)
- **List SP** [http://localhost:8080/spm-api/rest/aws/listSmartPlaces](http://localhost:8080/spm-api/rest/aws/listSmartPlaces)
- **Start SP** [http://localhost:8080/spm-api/rest/aws/startSmartPlace?instanceId=i-0a6d592772a57729f](http://localhost:8080/spm-api/rest/aws/startSmartPlace?instanceId=i-0a6d592772a57729f)
- **Stop SP** [http://localhost:8080/spm-api/rest/aws/stopSmartPlace?instanceId=i-0a6d592772a57729f](http://localhost:8080/spm-api/rest/aws/stopSmartPlace?instanceId=i-0a6d592772a57729f)
- **Terminate SP** [http://localhost:8080/spm-api/rest/aws/terminateSmartPlace?instanceId=i-0a6d592772a57729f](http://localhost:8080/spm-api/rest/aws/terminateSmartPlace?instanceId=i-0a6d592772a57729f)
- **Monitor SP** [http://localhost:8080/spm-api/rest/aws/monitorSmartPlace?instanceId=i-0a6d592772a57729f&metricName=cpuUtilization&periodType=1W](http://localhost:8080/spm-api/rest/aws/monitorSmartPlace?instanceId=i-0a6d592772a57729f&metricName=cpuUtilization&periodType=1W) and [http://localhost:8080/spm-api/rest/aws/monitorSmartPlace?instanceId=i-0a6d592772a57729f&metricName=networkUtilization&periodType=1W](http://localhost:8080/spm-api/rest/aws/monitorSmartPlace?instanceId=i-0a6d592772a57729f&metricName=networkUtilization&periodType=1W)

4.1.5 Deployment

The deployment diagrams are used for describing the hardware components, where software components are deployed. Hence, the deployment diagram represents the deployment view of a system. In our solution it was drawn a deployment diagram of the SPM system as shown in the figure 4.1.

Through the observation of this figure we can conclude that the system is formed by several components. First we have the main actor, the SP node itself, consisting of a Docker container solution that simulates the Fosstrak software needed to deploy the SP. In addition to this, each node is also configured as a Chef Client and with a Docker service. This last pulls the docker solution from the Docker Hub public repository.

Then, there are the other components of the system such as the SPM backend API which performs the provisioning and management of each configured SP node. The SPM backend API is supported with a PostgreSQL database server and the application server is Tomcat.

The Chef server is the orchestrator and it accepts the bootstrap order to create a new node over the AWS EC2 cloud. Finally, through the SPM mobile APP interface, which is shown on the left, the user is able to control the SP node with provisioning, management and monitoring orders but also to visualize
processing, storage and networking metrics of the SPM such as memory, CPU utilization and network In/Out traffic.

4.2 Implementation of Smart Place Manager App

In the mobile iOS application Xcode project were used some libraries that are listed below:

- Alamofire (4.6.0) [https://github.com/Alamofire/Alamofire](https://github.com/Alamofire/Alamofire)
- Charts (3.0.4) [https://github.com/danielgindi/Charts](https://github.com/danielgindi/Charts)
- SwiftyJSON (4.0.0) [https://github.com/SwiftyJSON/SwiftyJSON](https://github.com/SwiftyJSON/SwiftyJSON)

These libraries were installed in the XCode project by the Cocoapods\(^\text{14}\) dependency manager. The SPM was developed with a Swift 4 Xcode Cocoa project. This tool can also help to scale the project.

\(^\text{14}\)https://cocoapods.org/
elegantly by just installing its dependencies and running the following command over the Xcode project folder:

```
pod install
```

Listing 8: Cocoa Pods command to install pods into the SPM Xcode project.

## 4.2.1 Network Layer on App

In the previous chapter we talked about the design of the application with the MVC model. In the implementation we will talk about the network layer in the mobile application which we build using the **Alamofire**

This library, which was written by the same creator as Objective-C, it’s a powerful library that helps to decode JSON. Alamofire is a Swift-based HTTP networking library for iOS and macOS. It provides an elegant interface on top of Apples Foundation networking stack that simplifies a number of common networking tasks.

Alamofire provides chainable request/response methods, JSON parameter and response serialization, authentication, and many other features like asynchronous requests. In this way you can easily send an application and navigate between other screens of the application while waiting for the response, in the case of this work in greater detail in the creation of Smart Place. We’ll use Alamofire to perform several basic networking requests for requesting data from the SPM RESTful API which has its endpoints listening on Wildfly. This library was used after a conceptual understanding of HTTP networking and some study over Apples networking classes such as URLSession that is going to be described on the next section.

**URLSession** Whether an app retrieves Applikation data from a server, updates information or downloads remote files to disk, its the HTTP network requests living at the heart of mobile applications that make the magic happen. To help developers with the numerous requirements for network requests, Apple provides URLSession, a complete networking API for uploading and downloading content via HTTP. The URLSession is technically both a class and a suite of classes for handling HTTP/HTTPS-based requests as it is shown on the picture.

---

15 [urlhttps://github.com/Alamofire/Alamofire](urlhttps://github.com/Alamofire/Alamofire)
In this case we will want to make REST API calls to the SPM API and then retrieve a response for the requests in the SPM App. The SPM API endpoints will receive and serve up JSON requests and responses with GET HTTP methods. This response will be received by the network layer of the application and then display the information on the SPM mobile application view.

Alamofire  Alamofire is an HTTP networking library written in Swift, and it's built on top of NSURLSession, but with less lines of code to do REST interactions with POST/ GET/ PUT requests. So only if it is needed to do a more specialized network call, it will be essential the use of URLSession. Hence this library was used to build all the /GET calls that were made to the SPM API REST backend API.

The library Alamofire seeks to simplify HTTP networking in iOS. It carries out its mission with incredible efficiency by harnessing NSURLSession and the Foundation URL Loading System to create a networking interface that feels native to swift. Swift forgoes the delegate pattern and instead uses callbacks. Further, its built to be asynchronous. Performing networking calls on the main thread is always not a good choice but Alamofire employs many best practices out of the box like caching, for example, which is handled by NSURLConnection which helps to prevent needless requests. It also handles responses in JSON. Alamofire also calls for a singleton pattern that's built on top of an NSURLSessionConfiguration.

App Requests  The content of the request was designed in JSON format in order to simplify the overall architecture, using the swift library SwiftyJSON[^17]. The requests type and content can be seen in the Appendix B (SPM API Request Calls) section.

App Responses  The responses content was also designed in JSON. This response JSON object is converted in a results array, which is then iterated in cases such as Smart Place Listing. The responses type and content can be seen in the ?? section.

[^17]: https://github.com/SwiftyJSON/SwiftyJSON
4.2.2 Smart Place Application Screens

These are the functions provided by the smart place mobile application in order to interact with the AWS EC2. The mobile application will allow to create an SP on EC2 with the specifications selected on the app create screen and it will also allow to monitor metrics/specifications about that instance and scale up the instance machine.

**Main Screen**  In the "Main Screen" the application must allow the choice of one of three actions: *Create Smart Place* for Smart Place creation, *List SP* for Smart Place listing and an *About* section that describes a resume of the application context and purpose.

![Figure 4.3: Main Screen SPM APP.](image)

**Create Screen**  In the "Create Screen" flow the application must allow the creation of a SP on EC2 and then return the list of Smart Places created. This screen have options to choose the smart place image type, the cloud provider, the instances geographical region and the type of virtual machine and storage size that is going to be chosen, like it is shown on picture 4.4.
Monitor Memory and Storage Screen  In the "Monitor Screen" flow the application shall allow to read information relative to each instance (available / total storage, average CPU processing, requests latencies, region, cloud provider) and it shall allow to edit some metrics in order to manager the smart place throughout its use, like it is shown on picture 4.5:

Monitor CPU Utilization Screen  In the "CPU Monitoring Screen" it is presented the graphics that show the average and maximum values of the Smart Place CPU.
Monitor Network Traffic Screen  In the "Network Monitoring Screen" it is presented the graphics that show the In and Out byte values of traffic for the Smart Place Network traffic.

List Screen  In the "List Screen" flow the application shall allow to read information relative to all the instances created and list them by id’s and names, like it is shown on picture.

Detail Screen  In the "Detail Screen" it is shown several information about the smart place and take actions into the machines that are running it, like it is shown on picture.
**About Screen**  In the "About Screen" it is presented a brief resume about the application context and purpose like it is shown on picture 4.10.
Smart Place Manager

The objective of this work is to build a mobile app - Smart Place Manager - on a mobile platform such as iOS, and also a backend API - SPM-API - based on backend as a service architecture, in order to create the smart place environment which allow automatic provisioning and configuration of servers in the Cloud platform, as well as deploying the IoT application to interact with the smart place.

This mobile application should also monitor the server processing and storage usage among other metrics, taking action when necessary to the management of the Smart Place throughout its use. The response time is also monitored, which is often decisive in applications for an IoT, requiring low or predictable latency.

The demonstration scenario is an automated building access control that uses a RFID event processing software to track accesses in the facilities.

© 2018 Pedro Valério/IST. All rights reserved.

Figure 4.10: About Screen Spm App.
Chapter 5

Evaluation

This chapter describes the evaluation methodology as well as the experiences and tests developed to evaluate the solution presented both qualitatively and quantitatively. The qualitative evaluation looks at the requirements from section 3 and checks if they have been fulfilled. It also studies the user experience and application usability with a public survey that let users evaluate the experience with the mobile application.

The quantitative evaluation looks at metrics that are key performance indicators such as the request latency which must be low on such a solution. The energy consumption analysis is also taken in account because mobile applications have limited battery energy. The following sections will be show the evaluation aspects that were considered on the present work.

5.1 Evaluation Methodology

In this section it will be described what evaluation methodology was followed to evaluate the solution presented in this work.

5.1.1 Requirements Achievement

In this chapter it is assessed whether the requirements defined in the business perspective were achieved. In this way in the following tables show the conclusion about of the requirements that were implemented, i.e. what are the characteristics that the solution has to offer.

5.1.2 Application Usability and User Experience

In this section it was done a public survey in order to get the application usability evaluation that is in general needed on all applications and which should be the starting point on the first step to design the user interface as it will make part of the user experience. First it will be described the difference between UI (UI) and UX (UX) designs. The User Interface Design (UI) is the means by which a person interacts and controls a device, software, or application. This control can be done through buttons, menus and any element that provides an interaction between the device and the user. A good UI project anticipates the needs of the user and ensures that the interface contains elements of easy access and use, providing what is called a user-friendly experience, that is, an experience that is friendly and does not cause frustrations to the user.
The User Experience Design, UX, is related to the user experience and its feelings, since the UI is the way the user has achieved this experience. There is an analogy cited by Dain Miller [23] of the Web Designer Depot blog that illustrates very well this difference: "UI is the saddle, the stirrups, and the reigns. UX is the feeling you get being able to ride the horse, and rope your cattle.". The true user experience goes far beyond just providing customers with what they say they want, and it’s necessary to make them happy to own your product and happy to use it. Thus, to work with UX it is necessary for the designer to study and evaluate how users feel about a system, taking into account aspects such as ease of use, perception of system value, utility, efficiency in the execution of tasks and other characteristics to propose the best solution to a given problem. It is also necessary for UX Designer to have a broad view of all the disciplines that involve it - and there are many of them, as you can see in the chart below - because it is the junction of all that result in the user’s final experience.

Jakob Nielsen and Donald Norman [24] summarize User Experience as a way to encompass all aspects of end-user interaction with the company, its services and products, that is, it is responsible for studying the best ways to meet users’ needs and leave them satisfied with the whole process. The several aspects of User Experience Design are illustrated on the picture 5.1.

It is important to distinguish the total UX from the User Interface and usability, even though the User Interface is obviously an extremely important part of the design. According to the definition of usability, it is a quality attribute of the User Interface, covering whether the system is easy to learn, efficient to use, pleasant, and so forth. The Mobile App Usability refers to the ease of access or effort it takes for a user to complete their goal. On this work it will be evaluated the Mobile App Usability Mobile App Usability (MAU) through a public survey inspired on System Usability Scale [25] which has become an industry standard. In spite of the five question model that was referred, our survey consists on a questionnaire with ten response options for respondents; from Strongly Agree/No to Strongly Agree/Yes.

The noted benefits of using SUS include that it is a very easy scale to administer to participants and it can be used on small sample sizes with reliable results. The best way to interpret the results involves normalizing the scores to produce a percentile ranking. Its use is in classifying the ease of use of the site, application or environment being tested. Hence the user experience is not going to be fully explored and so the focus remains on the UI since it is part of the MAU. There are many factors that contribute to a MAU that will impact the overall efficiency and effectiveness in which a user completes
their goal. A usable interface should have three main outcomes:

- It should be easy for the user to become familiar with the UI.
- It should be easy for users to achieve their goal when using the app.
- It should be error-free. If your app doesn’t work to begin with, the rest of your UX elements will not matter.

In a world dominated by smartphones, finding a way to interact and engage with customers is becoming increasingly important. While the UI is a crucial part of the UX, usability should always take precedence. If an app is aesthetically pleasing but difficult to use, the overall user perception of the app will be negative as a result. Then it was analyzed and studied seven mobile app usability issues that are often overlooked in mobile app development. This issues are going to take place on the questions that are going to be asked on the public survey:

1. Platform Usability - A common frustration for mobile users is not having an app that works for their specific model of smartphone. Android and iOS, for example, are two different platforms since Android comes with an entirely different set of programming, design, and user interface considerations. For example, navigation for each platform differs quite significantly. iOS for one, doesn’t have a back button like Android. With that in mind, you should tailor your app to work within the native Android experience so your users can interact intuitively. It’s a good idea to budget for this so that you can ensure your app is optimized for the most common Android smartphones.

2. Provide Value Right Away - If you want new users to return to your app, you need to make sure that they discover the value early on, preferably during the onboarding process. If you don’t convince them to return within the first week, you’re likely going to lose them forever. The market is saturated with thousands of apps competing for the attention of your users so make sure you offer value right from the start.

3. Simple Navigation - One of the main issues that users have when using mobile apps, particularly m-commerce ones, is poor navigation. This means that while they are using the app, they have difficulty finding what they are looking for, and have to navigate too long to find it. When a user first downloads your app, they need to clearly understand how to navigate in order to complete their goal, whether that’s booking an appointment, purchasing a product, or finding information. This means that your navigation should have as few barriers as possible. Many apps include unique features but struggle to fit them together in a way that makes sense for the user. The navigation should be comprehensible for the user so they won’t end up lost on a random page. When focusing on usability, make sure that your products and/or services are extremely easy to find. The more complex it is for users to navigate the app, the higher the rate of user abandonment.

4. Clear and Concise Content - It’s common knowledge to simplify content when designing for mobile. However, you need just enough content that is essential to the user in order to complete a goal. This is particularly true for a buying process. Consumers still need complete information to make their purchase, and withholding basic information will result in a lower conversion rate. You should be tailoring your content for mobile, rather than copying it verbatim from web. Including too much information in your mobile app will undoubtedly result in a poor user experience with frustrated users digging to find what they’re looking for. Make it as easy as possible for the user to consume your content with as little pinching and zooming as possible by presenting the information in a clear and concise way.
5. Minimize the Number of Steps - The fewer steps, pages, buttons, and fields to input data, the better. Every time your user needs to complete an action, check to see if there’s a simpler way that would make their experience easier. Consider each action you require of your user as an added barrier. The fewer steps you include, the closer your users are to completing their goal.

6. Reduce Scrolling - Content prioritization also contributes significantly to the usability of an app. Users should get most if not all of the information they need to make a decision within the limits of their screen. Scrolling down is sometimes unavoidable and required in a number of instances, but side scrolling is something that should definitely be avoided. If a user needs to scroll sideways, it typically means that valuable content is being hidden.

7. Consider Landscape Orientation - When developing a mobile app, many people don’t consider landscape orientation as important. A good mobile app should be designed for both portrait and landscape to accommodate for optimal usability and user experience, particularly for an app that contains video content. This application didn’t consider landscape orientation.

8. Color and theme - It is also evaluated the user’s opinion over the app look and feel.

9. Objective reached - At least it was very important to know if the people inquired could just make dawn clear that the application made the result of what was explained before the app usage, when the interlocutor was explaining for what the app stands for.

The public survey has three sections: Main, Application Usability and User Experience. The main section describes what is the work resume like it is shown on picture 5.2. This section presents a resume about the application purpose and the method of evaluation the questions.
The public survey was performed using the platform Google Forms. After a brief introduction to the application objective and purpose, the users were asked to test it for two minutes. After the time elapsed they were asked to evaluate the application in terms of its usability and user experience, by completing the online ten questions anonymous survey. The first part of the survey has 6 questions that pretend to evaluate the application usability and they are:

1. Did you find that the steps, pages, buttons and fields to complete the final goal were exaggerated?

2. Did you think the lack of landscape orientation was bad?

3. Was the navigation through the application easy?

4. did you find the content of the application clear and concise?

5. What do you think about the app look and feel? Colours, images, appearance?

6. Did it take a long time scrolling in the application or was the scroll something tiring?

The Application Usability section is showed on picture 5.3

1 https://www.google.com/forms/about/
This second part of the survey has four questions that pretend to evaluate the user experience and they are:

1. Did you find the application slow or fast?
2. Was the application easy to use?
3. Did you enjoy using the application?
4. After using the application did you get aware of the utility and purpose of the application?

The user experience section is showed on picture 5.4.
5.1.3 Energy Consumption

We performed experiments to measure how much energy is drained only by our mobile app for end users. Thus the second evaluation focuses on the power consumption in mobile applications. The smartphones have some limitations concerning to short battery life time due to energy drain which is noticed to be increased when running applications that require intensive computations on the mobile devices.

According to surveys and interviews, it is known that programmers lack knowledge and awareness about software energy related issues. More than 80 per cent of the programmers do not take energy consumption into account when developing software. However, more than 60 percent of the programmers consider software energy consumption to be an important factor when choosing a mobile development platform[26].

There are several ways to low the power consumption that a application drains. These are commonly suggested energy-saving and performance enhancing coding practices can help to save battery[27]. It can also be done through techniques that reduces the consumed energy by mobile applications without changing their source code and thus employing powerful program analysis and transformations as well as an adaptive runtime system that determines an optimal offloading strategy at runtime[28].

It is possible to measure power consumption on mobile devices instrumentally and using software, in this case it should be done on a iPhone 6. While using an app, the power consumption can be measured using a power monitor connected to the iPhone (which will have the power supplied by the power monitor) like it is shown on the picture 5.5, and a power monitor software to control the behavior of the...
hardware such as start and stop the power supply to the iOS device. 

In the present work the energy impact of SPM iOS application is not going to be measured through instrumentation. Instead it is going to be used two tools from the Apples options to Power Consumption Measurement. The first tool is Instruments which is a performance-analysis and testing tool thats part of the Xcode tool set. It is designed to help profiling the OS X and iOS apps, processes, and devices in order to better understand and optimize their behavior and performance. This tool combine several metrics like CPU processing, memory allocations and leaks, energy log, among others like it is shown on picture 5.6.
Since the energy consumption while using an application depends on many parts of the device, like the CPU processing, the energy drained on antennas during network activity and the application needs over Bluetooth/GPS usage. So this tool combine the power drained in every part hardware component as it is shown on picture 5.7.

![Figure 5.7: Cpu usage impact on energy using CPU activity log on Instruments](31)

The Instruments energy impact can be measured in a scale of 0 to 20 which Apple calls as **abstract relative power** within a duration period which is being recorded over the SPM application usage on the iPhone 6 device. The duration period is going to be 15 minutes and one hour while doing actions like Creating Smart Places, Listing Smart Places, Start, Stop and Terminate Smart Places and Monitoring Smart Places. It is going to be measured the energy impact of each action repeated several times over periods of 1 hour and the experiments are going to be repeated for 30 times for each action, during that time interval. The iphone is going to be put on power mode since it reduces energy consuption on not doing several actions like the following:

- Email fetch
- Background app refresh
- Automatic downloads
- Auto-Lock (defaults to 30 seconds)
- Some visual effects
- "Hey Siri" tool
- iCloud Photo Library Automatic Backup/Upload (temporarily paused)

The energy impact is also going to be measured using the debug navigator that makes part of the IDE Xcode. The debug navigator provides a series of gauges that analyze the energy impact of the SPM app while testing it within Xcode. These gauges are displayed when the application has been launched by Xcode and is actively running or paused. The Energy gauge tool provides live information about the apps energy usage as it runs, and displays a graph of recent energy-related activity.

The information gathered in all of these areas is used to present an energy rating for the application. When the user interacts with the application, energy impact should be low unless the user has chosen to initiate an intensive operation, like it is shown on picture 5.8.

On the previous image it is possible to understand that there are six measures:

![Diagram](53)
• Cost and overhead. Blue bars illustrate the energy your app itself uses to perform work. Red bars show additional energy used by system resources that must be powered up to perform your apps work.
• CPU. A gray square in this row indicates that your app has used the CPU to perform work.
• Network. A gray square in this row indicates that your app has performed network operations.
• Location. A gray square in this row indicates that your app has utilized location services.
• GPU. A gray square in this row indicates that your app has used the GPU to perform graphics-related activity, such as drawing content on screen or playing an animation.
• Background. A gray square in this row indicates that your app is in a background state, but is still keeping the system awake.

At last it was also studied that almost mobile applications perform network operations of some kind, and its essential that networking be employed efficiently. This means eliminating overhead cost whenever possible by reducing and scheduling transactions and using efficient APIs.

Whenever the mobile application performs network operations, there is substantial overhead cost involved. The networking hardware, such as cellular and Wi-Fi radios, are powered down by default to conserve battery. When activated these resources must be powered up to perform activity. Then, they remain up for the duration of the activity and for an additional period of time in anticipation of more work. Some sporadic network transactions result in high overhead and can quickly deplete the devices battery like it is shown in figure 5.9

A variety of factors affect the amount of energy required to perform network operations on a device such as the following:
• Cellular network activity requires significantly more energy than activity over Wi-Fi.
• Poor or fluctuating signal conditions may result in slower or problematic transactions, which must be retried.
• Low network throughput (bandwidth) means radios have to stay on longer to perform transactions.
• Even geographic location and choice of cellular provider can impact energy consumption, as signal conditions and throughput can vary.

As a result, a devices technical specifications typically provide battery life estimates for a variety of scenarios. For example, the specs for iPhone 6 indicate Internet use of up to 10 hours on 3G and LTE,
and 11 hours on Wi-Fi. The Application usage ultimately determines how much network traffic occurs. The more efficient the network use by applications, the longer the device battery will long.

### 5.1.4 Request Latencies

In this chapter we measured two of the three parts in which the requests durations are divided on this solution: SPM API processing and cloud processing. The time spent between the SPM App communication and the SPM API, which is consumed over the operations of HTTP /GET requests on a RESTful API, was not measured. Hence in third place it was measured the total solution time for several actions on the Application. The full lifecycle of the HTTP requests is showed on picture 5.10:

![Figure 5.9: Device Networking Overhead on iOS apps](image)

Figure 5.9: Device Networking Overhead on iOS apps

![Figure 5.10: SPM application and api latencies](image)

Figure 5.10: SPM application and api latencies

As shown in the figure above, these time intervals made up the solution composed by the mobile appli-
cation fed by a backend API, and its communication with cloud. Hence in conclusion, the most important times in this solution are:

- **Cloud Processment** - The time spent by cloud provider Amazon Web Services to process requests made by the mobile application to EC2 and CloudWatch APIs, but also the time spent by Chef Server for the new node’s bootstrap process in the case of Create action, together with the provisioning of the containerized Docker solution for Fosstrak.
- **SPM API Processing** - The time spent by the SPM API for backend server processing without cloud and using JVM.
- **SPM App communication with SPM API** - The time spent by the mobile data network for the communication between the mobile application and the backend server. This time was not measured on the present work.
- **Total Solution Time** - The time spent by all the process since the requests exit the mobile app and arrive at it with a response.

This study is interesting to infer about the viability of the bootstrap solution and deployment of a backend API for scalability for several mobile devices or systems integrated platforms, with a common database.

### 5.2 Evaluation Setup

In this section it is described how and under which conditions the evaluations and experiences will be demonstrated, concerning Application Usability, User Experience, Energy Consumption and Request’ Latencies.

#### 5.2.1 Application Usability and User Experience

The public survey was conducted on a thirty people group composed of 10 university students from engineering, 10 system administrators and 10 random people, 60% male and 40% female, aged from 19 to 35. The survey was performed using the platform [Google Forms](https://www.google.com/forms/about/). After a brief introduction to the application objective and purpose, the users were asked to test it for two minutes. After the time elapsed they were asked to evaluate the application in terms of its usability and user experience, by completing the online ten questions anonymous survey. It is important to remember that the rating scale of the application was defined on a scale of 0 (I did not like / No) to 10 (I liked a lot / yes) depending on the semantic focus of the question. Only one session of tests with end-users was performed due to time constraints.

The public survey was conducted on a thirty people group composed of 10 university students from engineering, 10 system administrators and 10 random people, 60% male and 40% female, aged from 19 to 35. The survey was performed using the platform [Google Forms](https://www.google.com/forms/about/). After a brief introduction to the application objective and purpose, the users were asked to test it for two minutes. After the time elapsed they were asked to evaluate the application in terms of its usability and user experience, by completing the online ten questions anonymous survey. It is important to remember that the rating scale of the application was defined on a scale of 0 (I did not like / No) to 10 (I liked a lot / yes) depending
on the semantic focus of the question. Only one session of tests with end-users was performed due to time constraints. The survey respondents were people from inside and outside the Computer Science professional/academic area. Only one session of tests with end-users was performed due to time constraints.

5.2.2 Energy Consumption and Request Latencies

The mobile phone used for the survey respondents, energy consumption analysis and Request Latencies measures was iPhone 6 using mobile data from the ISP Vodafone PT and the backend API server was running on the Cloud. The speed connection at the survey and requests latency measurement day was measured by speed test[^34] and the result was for the download speed of 71.5 Mbps and for the upload speed was 45.5 Mbps, with a ping of 17 ms and Jitter of 7.9 ms, which was receiving the SpeedTest connection at that experience day.

![Speedtest result on survey day.](image)

The energy consumption analysis was performed during hour for each action and the actions were repeated 30 times during that time interval. The actions performed using the mobile application were three actions: Create, List and Monitor a Smart Place. It is noticed that the measures that are going to be read are exactly equal to the real action that took place on the mobile application.

5.3 Evaluation Results

In this section it is going to be showed the results from the experiments performed on this work. This evaluation is based on several criteria: Requirements Achievement, Application Usability and User Experience, Application Energy Consumption and Smart Place Manager Request latencies.

[^34]: [www.speedtest.net/apps/mac](http://www.speedtest.net/apps/mac)
5.3.1 Requirements Fulfilment

In this chapter we will evaluate if the requirements proposed in the solution design have been achieved. Concerning to the requirements R1 these were fully implemented as shown in the table 5.1.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Description</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Create Smart Place</td>
<td>Yes</td>
</tr>
<tr>
<td>R1.1</td>
<td>Choose Smart Place Type</td>
<td>Yes</td>
</tr>
<tr>
<td>R1.2</td>
<td>Choose Smart Place Name</td>
<td>Yes</td>
</tr>
<tr>
<td>R1.3</td>
<td>Choose Smart Place Cloud Provider</td>
<td>Yes</td>
</tr>
<tr>
<td>R1.4</td>
<td>Choose Smart Place Region</td>
<td>Yes</td>
</tr>
<tr>
<td>R1.5</td>
<td>Choose Smart Place CPU</td>
<td>Yes</td>
</tr>
<tr>
<td>R1.6</td>
<td>Choose Smart Place Storage Size</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5.1: Requirements Achievement - R01 Create Smart Place

The table 5.2 shows the result of the implementation of the requirements R2 considered in the solution design. The requirements for action and detail on the smart place were all implemented.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Description</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2</td>
<td>Actions and Details</td>
<td>Yes</td>
</tr>
<tr>
<td>R2.1</td>
<td>Create Smart Place</td>
<td>Yes</td>
</tr>
<tr>
<td>R2.2</td>
<td>Terminate Smart Place</td>
<td>Yes</td>
</tr>
<tr>
<td>R2.3</td>
<td>List Smart Places</td>
<td>Yes</td>
</tr>
<tr>
<td>R2.4</td>
<td>Start and Stop Smart Place</td>
<td>Yes</td>
</tr>
<tr>
<td>R2.5</td>
<td>Read Smart Place Information</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5.2: Requirements Achievement - R02 Actions and Details

The table 5.3 shows the result of the implementation of the requirements R3 considered in the solution design. As far as are concerned the monitoring requirements R3, these requirements were implemented in almost 70%, with remaining: Read VolumeReadBytes and VolumeWrite bytes metric and to read latencies for the requests exchanged between the Smart Place instance and the Load Balancer associated. Regarding the two uncompleted requirements, these are useful metrics to inform about the number of bytes read/write on the Smart Place disk and also to evaluate the latency of requests exchanged between Smart Places instances and the load balancer of the cloud provider.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Description</th>
<th>Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3</td>
<td>Monitoring Smart Place</td>
<td>Partially</td>
</tr>
<tr>
<td>R3.1</td>
<td>Read Free and Used Memory Metrics from Smart Place</td>
<td>Yes</td>
</tr>
<tr>
<td>R3.2</td>
<td>Read Free and Used Storage Metrics from Smart Place</td>
<td>Yes</td>
</tr>
<tr>
<td>R3.3</td>
<td>Choose time interval for CPU and Network Metrics</td>
<td>Yes</td>
</tr>
<tr>
<td>R3.4</td>
<td>Read Network In/Out for a Smart Place</td>
<td>Yes</td>
</tr>
<tr>
<td>R3.5</td>
<td>Read CPU Utilization Av and Max for a Smart Place</td>
<td>Yes</td>
</tr>
<tr>
<td>R3.6</td>
<td>Modify Volume Size</td>
<td>Yes</td>
</tr>
<tr>
<td>R3.7</td>
<td>Modify Instance Type</td>
<td>Yes</td>
</tr>
<tr>
<td>R3.8</td>
<td>Read Volume Read and Write Bytes</td>
<td>No</td>
</tr>
<tr>
<td>R3.9</td>
<td>Read Latencies for an Instance with Load Balancer</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 5.3: Requirements Achievement - R03 Monitor Smart Place
5.3.2 Application Usability and User Experience

Application Usability  The first method of evaluation was regarding the public survey was to test the application usability and it was decided to firstly question the end users about the amount of steps, screens and actions required to create and interact with Smart Places were exaggerated as well as if the lack of landscape orientation was a minus. The results are on the following images 5.12 and 5.13.

![Figure 5.12: Application Usability Evaluation - Steps to complete final goal.](image1)

Figure 5.12: Application Usability Evaluation - Steps to complete final goal.

![Figure 5.13: Application Usability Evaluation - Landscape Orientation.](image2)

Figure 5.13: Application Usability Evaluation - Landscape Orientation.
The results allow us to conclude that end users enjoyed the steps and work flows implemented for the final goal of creating a Smart Place in the cloud and performing its monitoring/maintenance. The lack of landscape orientation was not also considered relevant to the user. The evaluation two this two aspects ended with results mostly below 5/10. On the next step it was decided to evaluate the navigation throughout the application and also about its content quality and conciseness. The results are shown in images 5.14 and 5.15.

![Application Usability Evaluation - Navigation](image)

**Figure 5.14: Application Usability Evaluation - Navigation**

![Application Usability Evaluation - Content](image)

**Figure 5.15: Application Usability Evaluation - Content**

The results above show that users enjoyed the navigation through the application screens, with some of them giving lower scores, but mostly with scores in a range of 57 out of 10. Regarding the question about whether the content of the application was clear and concise, most users rated this question with 5-7 score. It should be noted that not all the target survey respondents has deep knowledge in the area. In general, the survey respondents found a correlation between the actions carried out in the application and its content, as shown by positive results greater than 5.

In the next point we tried to evaluate the look and feel of the application and hence, if it was pleasant for the users its use with the current color scheme and design. We also asked the survey respondents about their experience with the scrolling in the application. The results are shown on the pictures 5.16 and 5.17.
From the results obtained it can be concluded that the users liked the look and feel of the application, giving scoring values generally between 5 and 7 on 10. We also can conclude that users found appropriated the amount of scroll that was used and at this question it was given scores mostly between 1 and 2.

**User Experience** The second method of evaluation on the public survey was about its user experience. We began by evaluating whether the application was quick and how easy it was to use it. This application was designed and developed to be a simple tool easy to use by sysadmins or even non-IT people, and through it to be able to respond to the main problems with servers and machines that support the SP in the cloud. The results are shown in the figures 5.18 and 5.19.

In relation to the application speed, the survey respondents gave average and high scores, with values between 510. The ease of use of the application also had a positive evaluation with high values of 58. Finally, we asked the final questions about whether people liked to use the application and whether, at the end of the application use, they were able to understand its utility and purpose, which were explained before the use and the survey completion. The results are shown in the figures 5.20 and 5.21.

We can conclude that the survey respondents enjoyed using the application, found it easy to use and on average they were able to understand the ultimate goal of the mobile application, with its use and
consequent actions on the Smart Places, as well as their monitoring functions. The inquiries were made to people in the area and to people outside the area of interest in this work.

5.3.3 Energy Consumption

The energy impact was measured by the energy analysis of the Instruments software that is part of XCode, which measures are shown on picture 5.22. The measured that was obtained by Instruments[?] which calls the power drained by the application process as Abstract Power. This value is represented on the Energy Log application tool which indicates a level from 0 to 20, indicating how much energy the
Apple refers that these numbers are subjective because if the energy usage level is occasionally high, it doesn’t necessarily mean that the app has a problem. It may simply require more energy for some of the tasks it performs such as the asynchronous requests for the creation of the smart place, which are waiting for the response after the bootstrap process of the new Smart Place, indicating on the application the message that it was started the creation of the smart place and returning the user to the main menu. When the process finishes the response arrives at the application and the action of Create comes to an end in terms of consumed memory and CPU utilization, thus having an effect on reducing the energy consumption since the usage of memory and the number of CPU cycles represents an energy consumption that is not specified in this work. Other hardware components may affect this value, for example the wi-fi antenna and the use the GPS while performing complex network operations, the last is not the case of this work.

The energy impact results from the evaluation methodology are shown in the following tables. The average values were measured and then presented for the following actions: creating, listing and monitoring (visualizing CPU utilization and network traffic). While in standby the relative abstract power consump-
In the creation action the consumed energy registered a peak of 14/20 of relative abstract power, like it is shown on picture 5.24. This value was registered on average since it also had values of 11/20. In this action is sent information about the parameters for the Smart Place creation request and the response is waiting until the end of the bootstrap process and its consequent return response to the application.

In the listing action the consumed energy registered a value of 13/20 like it is shown on picture 5.25. The Listing values vary depending on the amount of information loaded in the request sent. In this
way, the more SPs that are currently running, the more energy will be expected to be wasted during the process of extracting, transforming and loading the Smart Places information into the Views.

In the monitoring action, the consumed energy registered a peak of 17/20 relative abstract power, like it is shown on picture 5.26. This peak is related to the information and data points loaded into the chart in order to visualise CPU utilization and Network traffic.

In conclusion we can say that the energy consumption of the application is acceptable. However, this issue can be further examined in terms of reading the number of objects created and removed from memory, throughout the lifetime and application flows. The summary results are shown in table 5.4.
5.3.4 Request Latencies

The latency tests were performed to measure the times spent on all Smart Place Manager life cycle consisting of a mobile application and a backend API. Twenty repeated tests were performed for each of the three actions analyzed in the energy impact. These, namely the Creation, Listing and a specific Monitoring action (in this case for request latency it was the monitoring task that was evaluated, that consists on instance vertical scale - change instance CPU and memory type/size). The table 5.5 shows the several times, firstly the time spent on the backend SPM API, Chef bootstrap plus AWS processing to create a Smart Place, and in second place the time spent by the SPM API and AWS cloud processing for List and Change Instance Type (monitoring) actions. The total time elapsed on application since the request leaves it till when the response arrives, the total solution time for each action is also showed. The total time on mobile application is also showed.

<table>
<thead>
<tr>
<th>Action / Duration</th>
<th>Action / Duration</th>
<th>API (ms)</th>
<th>Cloud (ms)</th>
<th>Total Solution (ms)</th>
<th>Iteractions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Smart Place</td>
<td>3122</td>
<td>159370</td>
<td>243806</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>List Smart Place</td>
<td>180</td>
<td>235</td>
<td>861</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Change Instance Type (Scale Up)</td>
<td>7</td>
<td>108</td>
<td>329</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.5: Smart Place Manager Request' Latencies table

Observing these results, it can be concluded that the Create Smart Place process during the twenty iterations performed recorded a mean time of bootstrapping of 159370 ms which represents approximately 2.6 minutes. The processing time for the SPM API was the highest recorded in the three actions that were evaluated, with a time spend of 3 seconds, and the additional time to the previous bootstrap was due to necessary database operations as well as other interactions with the EC2 API such as the Smart Place registering with the Load Balancer. These last actions are needed because they can’t be performed using the Chef bootstrap process in which the EC2 Knife Plugin is used to make the bridge between the Chef Server and instance creation on AWS and thus this plugin does not execute the actions described before. In this way the average time of the total Chef bootstrap process of the new smart place was in 243806 ms which corresponds approximately to 4.16 minutes. This corresponds to the total time of bootstrapping a new node with the Chef tool, followed by the creation of the node with the Docker containerized Fosstrak solution and lastly other necessary actions in the Cloud for that Smart Place instance, until the app’s creation response arrives to the app. The other Smart Place actions like Start, Stop and Terminate but also the memory and storage metrics reading and the CPU and network monitoring data for graphics are not measured in this current version. Thus we can conclude that the SPM solution formed by the App and the API backend is a solution with low response times, although the bigger time spent in requests corresponds to the create request. This is justified because the the bootstrap process has the need to be triggered from a machine (in this case the backend server) which executes chef-client directly on the node, and then installs all the software provided on the Chef cookbooks, including the docker images pulled for the Fosstrak platform.

5https://github.com/chef/knife-ec2
Chapter 6

Conclusion

The present work explored the provisioning, deployment and monitoring of IoT applications running over virtual cloud machines for smart buildings based on the RFID technology. A mobile application was developed that can install the Smart Place in the cloud with a backend and database. The cloud-based backend deployment and orchestration is able to quickly create and run a smart place software stack in a specific cloud provider, within an available region in order to meet the low latency requirements of many IoT applications, and also manage and monitor the cloud instances.

To improve the provisioning of RFID application middleware in the cloud, a mechanism based on Docker containers and the Chef tool that automates the installation and configuration of the modules that composes the example platform RFID middleware called Fosstrak. This mechanism is part of the backend API because it allowed to perform the application provisioning and monitoring of the cloud Smart Places in a very practical and fast approach. Although the experiments were conducted in a single cloud provider (Amazon Web Services), on a single region (Ireland) and with only a Smart Place software stack (RFID), the developed mobile application was designed to allow to choose between several cloud providers to provision several RFID or other SP software.

Regarding the system evaluation, we defined several requirements for the mobile application solution which were met to perform creating, listing, and monitoring tasks over a SP. With this work we also have demonstrated how to elaborate an application usability and user experience public survey, with a set of ten questions that allow us to identify the strengths and weaknesses of the application in terms of its design and sensations felt by the user while using it, but also concerning its speed and comprehension of its final goal for use. It was shown that the survey respondents enjoyed using the application and find it useful even by whom had no background on the area. The obtained latencies evaluation results show that the solution here designed and implemented to reproduce the concept of Mobile Backend As a Service, consisting of a mobile application and an API backend, is well suited to orchestrate Smart Place provisioning and deployment automated processes in Cloud. This work allowed to conclude on the processing times in cloud processing itself and on the RESTful backend API in Java, hence the conclusions were also drawn regarding the time spent in these processes when they are triggered by a mobile application. We performed tests and scripts to evaluate the latency of the request through this system lifecycle. Finally, through the energy analysis of this application, we were able to perceive the approximate energy consumed by the mobile SPM application in actions that mainly correspond to HTTP requests to a backend API, as well as in the rendering of data to feed the Smart Places’s monitoring charts on the mobile application.
6.1 Future Work

There are some aspects that require future work.

**Evaluation Scenario.** In the evaluation scenario we used a virtual RFID processing platform (Fosstrak) but we did not simulate the Smart Place events themselves although the solution is running on the cloud containers or neither monitor the events that could be occurring if it was being fed like in the past work. A future improvement is to conduct the system evaluation in a real scenario. It is interesting to test a different solution for Smart Place software with iBeacon. Various vendors have been making iBeacon-compatible hardware transmitters typically called beacons a class of Bluetooth low energy (BLE) devices that broadcast their identifier to nearby portable electronic devices. Hence, an evaluation scenario could be a store for collecting items for shopping.

**High Availability.** The solution that was presented in this work proposes that the Smart Place is deployed on a virtual cloud machine and its modules composed of capturing and filtering plus aggregation are running on the same cloud node, with a bootstrap process that occurs on the backend following a full cloud-based deployment. Since in this case it has been admitted that the number of RFID events in a building may have a peak but always corresponds to a finite maximum number of RFID access readings, it is not like a website that may need to replicate the solution for more machines automatically to respond to a very large unknown demand. So the interesting thing about this scenario will be to scale out the Smart Place software solution that will be used, with the Cloud provider High Availability solution (horizontally scaling) for larger environments, thus it can replicate the smart places software stack according to the load demand that they are exposed to, with the AWS Elastic Beanstalk tool from Amazon or other from the cloud provider chosen. In the future people will have to manage the smart places in their lives. Their homes, their workplaces. Smart place management, with digital assistants, will play a role. As people nowadays are able to manage their personal computers and phones without deep technical expertise, in the near future they will do the same with the smart places. This tool, the Smart Place Manager mobile application and API are a technical contribution that moves us closer in this future direction.

**Architecture and Tests improvements** In terms of security, the Static and Dynamic application security testing was not done on the application neither on the API. In this way, for future work would be interesting to do a security analysis of the Smart Place Manager Application and API. It is also interesting to do a future work on using automatic analysis of metrics, to allow an automated management also, besides automated provisioning, deployment and monitoring.

**Functional improvements for App and API** In terms of functional requirements for the application, some cases were intended for implementations in future releases. Apply some changes to several rooms inside a Smart Place, such as to change a user’s permissions, changing their access to certain rooms or even a time control system in the context of the Smart Site under study which that allows the access to doors inside a building or a room. An event alarmist logic could be implemented for the user to receive alerts directly in the application about events that are of interest about the Smart Place be it networking, computing or for general troubleshooting purposes.
6.2 Final words on the future of IoT applications

With this work we have contributed to the validation of the suitability of the Mobile Backend solution for the automated provisioning and deployment of the Smart Places software stack in cloud machines. We believe that using the cloud infrastructure to support IoT applications with a common backend and database supplying every type of device, it will be the most adopted approach, even for applications that have strict requirements such for low-latency. The flexibility provided by the backend automated orchestration and deployment, as also the cloud paradigm will allow that IoT applications from several domains may be deployed in a cloud infrastructure, and be monitored or scaled up with the advantage of the benefits offered by this paradigm.

In the future people will have to manage the smart places in their lives. Their homes, their workplaces. Smart place management, with digital assistants, will play a role. As people nowadays are able to manage their personal computers and phones without deep technical expertise, in the near future they will do the same with the smart places. This tool, the Smart Place Manager mobile APP and API, are a technical contribution that moves us closer in this future direction.
Bibliography


Appendix A

SPM Solution Requirements

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.01.1</td>
<td>Choose Smart Place Type</td>
<td>It must be possible to choose the Smart Place image type from the list.</td>
</tr>
<tr>
<td>R.01.2</td>
<td>Choose Smart Place Name</td>
<td>It must be possible to write in a textfield the new Smart Place name.</td>
</tr>
<tr>
<td>R.01.3</td>
<td>Choose Cloud Provider</td>
<td>It must be possible to choose the cloud provider from the list.</td>
</tr>
<tr>
<td>R.01.4</td>
<td>Choose Availability Zone</td>
<td>It must be possible to choose the Smart Place availability zone type from the list.</td>
</tr>
<tr>
<td>R.01.5</td>
<td>Choose Smart Place CPU</td>
<td>It must be possible to choose the Smart Place CPU type from the list.</td>
</tr>
<tr>
<td>R.01.6</td>
<td>Choose Smart Place Storage Size</td>
<td>It must be possible to choose the Smart Place storage size from the list.</td>
</tr>
</tbody>
</table>

Figure A.1: SPM R1 - Create Smart Place Requirements summary.
<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.02.1</td>
<td>Create Smart Place</td>
<td>It must be possible to create a Smart Place on the chosen cloud provider.</td>
</tr>
<tr>
<td>R.02.2</td>
<td>Terminate Smart Places</td>
<td>It must be possible to terminate a Smart Place on the cloud provider chosen.</td>
</tr>
<tr>
<td>R.02.3</td>
<td>List Smart Places</td>
<td>It must be possible to list all existing Smart Places on all cloud providers.</td>
</tr>
<tr>
<td>R.02.4</td>
<td>Start and Stop Smart Place</td>
<td>It must be possible to start and stop a Smart Place.</td>
</tr>
<tr>
<td>R.02.5</td>
<td>Read Smart Place Information</td>
<td>It must be possible to read several information about a given Smart Place like: name, type, cloud provider, public ip, availability zone, instance Id, status, image type, instance type and Storage.</td>
</tr>
</tbody>
</table>

Figure A.2: SPM R2 - Actions and Details Requirements summary.
<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.03.1</td>
<td>Read Memory utilization for a Smart Place</td>
<td>It must be possible to read storage utilization for a Smart Place. This metric identifies the memory, cached, available, used and free upon a selected Smart Place.</td>
</tr>
<tr>
<td>R.03.2</td>
<td>Read Storage utilization for a Smart Place</td>
<td>It must be possible to read storage utilization for a Smart Place. This metric identifies the storage used and available upon a selected Smart Place.</td>
</tr>
<tr>
<td>R.03.3</td>
<td>Choose time interval to read Cpu utilization and Network In/Out (Bytes)</td>
<td>It must be possible to choose the time interval to read Cpu utilization and Network In/Out traffic from a Smart Place.</td>
</tr>
<tr>
<td>R.03.4</td>
<td>Graphics - NetworkIn or NetworkOut for a Smart Place</td>
<td>It must be possible to visualize graphics about NetworkIn or NetworkOut during a time interval. This metric identifies the volume of outgoing network traffic from a single Smart Place.</td>
</tr>
<tr>
<td>R.03.5</td>
<td>Graphics - Cpu Utilization for a Smart Place</td>
<td>It must be possible to read CPU utilization average and maximum values for a Smart Place during a time interval. This metric identifies the processing power required to run an application upon a selected Smart Place.</td>
</tr>
<tr>
<td>R.03.6</td>
<td>Modify Smart Place volume size or type</td>
<td>It must be possible to modify the size or the type of volume attached to an instance without stopping it, from a picker view with values.</td>
</tr>
<tr>
<td>R.03.7</td>
<td>Modify instance type</td>
<td>It must be possible to modify instance virtual resources such as instance type which includes CPU type and RAM Memory associated from a picker views with values.</td>
</tr>
<tr>
<td>R.03.8</td>
<td>Graphics - Read VolumeReadBytes and VolumeWriteBytes for a Smart Place</td>
<td>It must be possible to read information on the I/O operations of a volume attached to an instance in a specified period of time.</td>
</tr>
<tr>
<td>R.03.9</td>
<td>Modify the Smart Place availability zone</td>
<td>It must be possible to modify the Smart Place availability zone in order to reduce requests latencies.</td>
</tr>
</tbody>
</table>

Figure A.3: SPM R3 - Monitor Smart Places Requirements summary.
<table>
<thead>
<tr>
<th><strong>R.01</strong></th>
<th><strong>Choose Smart Place Type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td><strong>R.01.1 Choose Smart Place Type</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>It must be possible to choose the type of the Smart Place to be deployed.</td>
</tr>
<tr>
<td><strong>Entrance</strong></td>
<td>Query criteria for all existing Smart Places images.</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>Database</td>
</tr>
<tr>
<td><strong>Exit</strong></td>
<td>List of all existing Smart Places images.</td>
</tr>
<tr>
<td><strong>Destination</strong></td>
<td>User mobile interface</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>The user press the button to create a new Smart Place and the first create Smart Place screen appears showing that he has to choose the Smart Place type on the first picker view.</td>
</tr>
<tr>
<td><strong>Needs</strong></td>
<td>Existing smart place docker image name on database</td>
</tr>
</tbody>
</table>

Figure A.4: SPM Requirement R1.1 - Choose Smart Place Type.

<table>
<thead>
<tr>
<th><strong>R.01</strong></th>
<th><strong>Choose Smart Place Name</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td><strong>R.01.2 Choose Smart Place Name</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>It must be possible to type the Smart Place name to be deployed.</td>
</tr>
<tr>
<td><strong>Entrance</strong></td>
<td>None.</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>User input</td>
</tr>
<tr>
<td><strong>Exit</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Destination</strong></td>
<td>User mobile interface</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>The user press the button to create a new Smart Place and the first create Smart Place screen appears showing that he has to choose the availability zone on the first text field.</td>
</tr>
<tr>
<td><strong>Needs</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

Figure A.5: SPM Requirement R1.2 - Choose Smart Place Type.

<table>
<thead>
<tr>
<th><strong>R.01</strong></th>
<th><strong>Choose Cloud Provider</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td><strong>R.01.3 Choose Cloud Provider</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>It must be possible to choose the cloud provider.</td>
</tr>
<tr>
<td><strong>Entrance</strong></td>
<td>Query criteria for all existing cloud providers.</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>Database</td>
</tr>
<tr>
<td><strong>Exit</strong></td>
<td>List of all cloud providers available.</td>
</tr>
<tr>
<td><strong>Destination</strong></td>
<td>User mobile interface</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>The user press the button to create a new Smart Place and the first create Smart Place screen appears showing that he has to choose the cloud provider on the second pickerview.</td>
</tr>
<tr>
<td><strong>Needs</strong></td>
<td>Existing cloud providers name on database</td>
</tr>
</tbody>
</table>

Figure A.6: SPM Requirement R1.3 - Choose Smart Place Type.
### R.01 Choose Availability Zone

<table>
<thead>
<tr>
<th>Function</th>
<th>R.01.4 Choose Availability Zone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>It must be possible to choose the availability zone for the smart place to be deployed.</td>
</tr>
<tr>
<td>Entrance</td>
<td>Query criteria for all existing availability zones.</td>
</tr>
<tr>
<td>Source</td>
<td>Database</td>
</tr>
<tr>
<td>Exit</td>
<td>List of all availability zones.</td>
</tr>
<tr>
<td>Destination</td>
<td>User mobile interface</td>
</tr>
<tr>
<td>Action</td>
<td>The user press the button to create a new Smart Place and the first create Smart Place screen appears showing that he has to choose the availability zone on the third pickerview.</td>
</tr>
<tr>
<td>Needs</td>
<td>Existing cloud providers name on database.</td>
</tr>
</tbody>
</table>

Figure A.7: SPM Requirement R1.2 - Choose Smart Place Type.

### R.01 Choose Smart Place Cpu Type

<table>
<thead>
<tr>
<th>Function</th>
<th>R.01.5 Choose Smart Place Cpu Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>It must be possible to choose the Smart Place cpu type.</td>
</tr>
<tr>
<td>Entrance</td>
<td>Query criteria for all existing Smart Place cpu types.</td>
</tr>
<tr>
<td>Source</td>
<td>Database</td>
</tr>
<tr>
<td>Exit</td>
<td>List of all Cpu types available.</td>
</tr>
<tr>
<td>Destination</td>
<td>User mobile interface</td>
</tr>
<tr>
<td>Action</td>
<td>The user press the button to create a new Smart Place and the first create Smart Place screen appears showing that he has to choose the Smart Place Cpu type on the fifth pickerview.</td>
</tr>
<tr>
<td>Needs</td>
<td>Existing cpu types on database.</td>
</tr>
</tbody>
</table>

Figure A.8: SPM Requirements R1.5 - Choose Smart Place CPU Type.

### R.01 Choose Smart Place Storage size

<table>
<thead>
<tr>
<th>Function</th>
<th>R.01.6 Choose Smart Place storage size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>It must be possible to choose the Smart Place storage size.</td>
</tr>
<tr>
<td>Entrance</td>
<td>Query criteria for all existing storage sizes.</td>
</tr>
<tr>
<td>Source</td>
<td>Database</td>
</tr>
<tr>
<td>Exit</td>
<td>List of all storage sizes available.</td>
</tr>
<tr>
<td>Destination</td>
<td>User mobile interface</td>
</tr>
<tr>
<td>Action</td>
<td>The user press the button to create a new Smart Place and the first create Smart Place screen appears showing that he has to choose the Smart Place storage size on the sixth pickerview.</td>
</tr>
<tr>
<td>Needs</td>
<td>Existing storage sizes on database.</td>
</tr>
</tbody>
</table>

Figure A.9: SPM Requirement R1.6 - Choose Smart Place Storage Size.
<table>
<thead>
<tr>
<th>R.02</th>
<th>Manage Smart Place provisioning and interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>R.02.1 Create Smart Place</strong></td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>It must be possible to create a Smart Place on the cloud provider chosen and automatically bootstrap a chef node with cookbooks and it also must start docker containers on that instance from the docker images that were chosen on the requirement R.01.2.</td>
</tr>
<tr>
<td><strong>Entrance</strong></td>
<td>Instance Name; Smart Place Type; Cloud Provider Type; Instance Type; Volume Type.</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>Information selected by the user for Smart Place name; query criteria for instance type and storage volume existing in database.</td>
</tr>
<tr>
<td><strong>Exit</strong></td>
<td>Instance Id created.</td>
</tr>
<tr>
<td><strong>Destination</strong></td>
<td>User mobile interface</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>The user press the button Create and the mobile app enters the first create instance screen where it appears showing that he has to choose the Smart Place type and the cloud provider on the both picker views. After selecting the Next button the second create screen is displayed where the user has to manually input the desired instance name, choose the instance type and storage volumes from the pickerviews and then finally hit the create button.</td>
</tr>
<tr>
<td><strong>Needs</strong></td>
<td>Existing Smart Place image names and cloud providers names on the database.</td>
</tr>
</tbody>
</table>

Figure A.10: SPM Requirement R2.1 - Create Smart Place

<table>
<thead>
<tr>
<th>R.02</th>
<th>Manage Smart Place provisioning and interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>R.02.2 Terminate Smart Place</strong></td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>It must be possible to terminate one Smart Place.</td>
</tr>
<tr>
<td><strong>Entrance</strong></td>
<td>Instance Id.</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>Information selected by the user for Smart Place name and instance types, storage volumes existing in database.</td>
</tr>
<tr>
<td><strong>Exit</strong></td>
<td>instanceId terminated; currentState; previousState.</td>
</tr>
<tr>
<td><strong>Destination</strong></td>
<td>User mobile interface</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>The user presses the List button in order to list the current Smart Places and then the user press the info button on the desired Smart Place to list the information about it. On this screen the user has to press the terminate instance button.</td>
</tr>
<tr>
<td><strong>Needs</strong></td>
<td>Existing Smart Place image names on database.</td>
</tr>
</tbody>
</table>

Figure A.11: SPM Requirement R2.2 - Terminate Smart Place.
### R.02 Manage Smart Place provisioning and interaction

<table>
<thead>
<tr>
<th>Function</th>
<th><strong>R.02.3 List Smart Places</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>It must be possible to list all the Smart Places on the cloud provider chosen.</td>
</tr>
<tr>
<td><strong>Entrance</strong></td>
<td>Authentication Parameters on the cloud provider chosen.</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>Information selected by the user.</td>
</tr>
<tr>
<td><strong>Exit</strong></td>
<td>For each instance: instanceId; instanceState; dnsName; instanceType; availabilityZone; subnetId; architecture; publicIp; publicDnsName;</td>
</tr>
<tr>
<td><strong>Destination</strong></td>
<td>User mobile interface</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>The user press the button to list the instances and on the next screen it appears a list of cells each one corresponding to an instance.</td>
</tr>
<tr>
<td><strong>Needs</strong></td>
<td>One or more existing instances that are started or stopped on the cloud provider chosen.</td>
</tr>
</tbody>
</table>

Figure A.12: SPM Requirement R2.3 - List Smart Places.

<table>
<thead>
<tr>
<th>Function</th>
<th><strong>R.02.4 Start and Stop Smart Places</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>It must start or stop an Smart Place that was previously stopped / started.</td>
</tr>
<tr>
<td><strong>Entrance</strong></td>
<td>Instance Id</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>Information selected by the user.</td>
</tr>
<tr>
<td><strong>Exit</strong></td>
<td>Information about the previous and actual Smart Place state.</td>
</tr>
<tr>
<td><strong>Destination</strong></td>
<td>User mobile interface</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>The user presses the List button in order to list the current instances and then the user press the info button on the desired instance to list the information about it. On this screen the user has to press the start/stop button.</td>
</tr>
<tr>
<td><strong>Needs</strong></td>
<td>One or more existing instances that are started or stopped on the cloud provider chosen.</td>
</tr>
</tbody>
</table>

Figure A.13: SPM Requirements R2.4 - Start and Stop Smart Place.
### R.02 Manage Smart Place provisioning and interaction

<table>
<thead>
<tr>
<th>Function</th>
<th>R.02.5 Read Smart Place Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>It must return information for the Smart Place selected like name, type, cloud provider, public ip, availability zone, instance id, status, image type, instance type and Storage.</td>
</tr>
<tr>
<td>Entrance</td>
<td>Instance Id</td>
</tr>
<tr>
<td>Source</td>
<td>Information from Database and Cloud Provider.</td>
</tr>
<tr>
<td>Exit</td>
<td>Information about the desired Smart Place.</td>
</tr>
<tr>
<td>Destination</td>
<td>User mobile interface</td>
</tr>
<tr>
<td>Action</td>
<td>The user presses the List button in order to list the current instances and then the user visualizes the information of the desired instance on the Instance Detail Screen.</td>
</tr>
<tr>
<td>Needs</td>
<td>One or more existing instances on the cloud provider chosen.</td>
</tr>
</tbody>
</table>

Figure A.14: SPM Requirement R2.5 - Read Smart Place Information.

### R.03 Manage Smart Place monitoring

<table>
<thead>
<tr>
<th>Function</th>
<th>R.03.1 Read Free and Used Memory Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>It must be possible to read Free and Used Memory Metrics for a given volume attached to an instance.</td>
</tr>
<tr>
<td>Entrance</td>
<td>instanceId;</td>
</tr>
<tr>
<td>Source</td>
<td>Information passed by inside the app for instanceId;</td>
</tr>
<tr>
<td>Exit</td>
<td>Memory Free; Memory Used;</td>
</tr>
<tr>
<td>Destination</td>
<td>User mobile interface</td>
</tr>
<tr>
<td>Action</td>
<td>The user press the button Monitor and then on the Monitor screen that is opened the user can visualize these metrics.</td>
</tr>
<tr>
<td>Needs</td>
<td>Existing instance on the cloud provider.</td>
</tr>
</tbody>
</table>

Figure A.15: SPM Requirement R3.1 - Read Memory Utilization for a Smart Place.
### Figure A.16: SPM Requirement R3.2 - Read Storage Utilization for a Smart Place.

<table>
<thead>
<tr>
<th>R.03</th>
<th>Manage Smart Place monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td>R.03.2 Read Free and Used Storage Metrics</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>It must be possible to read Free and Used volume space for a given volume attached to an instance.</td>
</tr>
<tr>
<td><strong>Entrance</strong></td>
<td>instanceId;</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>Information passed by inside the app for instanceId;</td>
</tr>
<tr>
<td><strong>Exit</strong></td>
<td>VolumeReadBytes; VolumeWriteBytes</td>
</tr>
<tr>
<td><strong>Destination</strong></td>
<td>User mobile interface</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>The user press the button Monitor and then the user selects one instance. On the new screen that is opened the user can visualize these metrics.</td>
</tr>
<tr>
<td><strong>Needs</strong></td>
<td>Existing instance on the cloud provider with monitor instances enabled.</td>
</tr>
</tbody>
</table>

### Figure A.17: SPM Requirements 3.5 - Choose time interval for CPU and Network Metrics.

<table>
<thead>
<tr>
<th>R.03</th>
<th>Manage Smart Place monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td>R.03.3 Choose time interval for Cpu utilization and Network Metrics.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>It must be possible to choose the time interval to read Cpu utilization and Network In/Out (Bytes) traffic from a Smart Place.</td>
</tr>
<tr>
<td><strong>Entrance</strong></td>
<td>instanceId;</td>
</tr>
<tr>
<td><strong>Source</strong></td>
<td>Information passed by inside the app for instanceId;</td>
</tr>
<tr>
<td><strong>Exit</strong></td>
<td>1W; 1D; 1H; 5M; 30; 1D;</td>
</tr>
<tr>
<td><strong>Destination</strong></td>
<td>User mobile interface</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>The user press the button Monitor and then on the Monitor screen that is opened the user can visualize these metrics for selection over a picker view.</td>
</tr>
<tr>
<td><strong>Needs</strong></td>
<td>Existing instance on the cloud provider.</td>
</tr>
</tbody>
</table>
### R.03 Manage Smart Place monitoring

<table>
<thead>
<tr>
<th>Function</th>
<th>R.03.4 Read Network In and Network Out for an instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>It must be possible to read Network In or Network Out for a given Smart Place.</td>
</tr>
<tr>
<td>Entrance</td>
<td>Smart Place Name;</td>
</tr>
<tr>
<td>Source</td>
<td>Information selected by the user for instance name;</td>
</tr>
<tr>
<td>Exit</td>
<td>Network In; Network Out;</td>
</tr>
<tr>
<td>Destination</td>
<td>User mobile interface</td>
</tr>
<tr>
<td>Action</td>
<td>The user press the button Monitor and then the user selects one instance. On the new screen that is opened the user can visualize these metrics.</td>
</tr>
<tr>
<td>Needs</td>
<td>Existing instance on the cloud provider with monitor instances option enabled.</td>
</tr>
</tbody>
</table>

Figure A.18: SPM Requirement R3.4 - Read Network In and Network Out for and Smart Place.

<table>
<thead>
<tr>
<th>Function</th>
<th>R.03.5 Read CPUUtilization Average and Maximum for a Smart Place</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>It must be possible to read CPUUtilization for an instance. This metric identifies the processing power required to run an application upon a selected instance.</td>
</tr>
<tr>
<td>Entrance</td>
<td>Instance Name;</td>
</tr>
<tr>
<td>Source</td>
<td>Information selected manually by the user for instance name;</td>
</tr>
<tr>
<td>Exit</td>
<td>Maximum value; Unit on Percent;</td>
</tr>
<tr>
<td>Destination</td>
<td>User mobile interface</td>
</tr>
<tr>
<td>Action</td>
<td>The user press the button Monitor Instance and then the user selects one instance. On the new screen that is opened the user can visualize these metrics.</td>
</tr>
<tr>
<td>Needs</td>
<td>Existing instance on the cloud provider with monitor instances option enabled.</td>
</tr>
</tbody>
</table>

Figure A.19: SPM Requirement R3.5 - Read CPU Utilization Average and Maximum values for a Smart Place.
<table>
<thead>
<tr>
<th>Function</th>
<th>R.03.6 Modify volume size or type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>It must be possible to list the volumes attached to an instance and then to modify the size without stopping it.</td>
</tr>
<tr>
<td>Entrance</td>
<td>Volume Id; Volume Size; Volume Type;</td>
</tr>
<tr>
<td>Source</td>
<td>Information selected by the user for volume id, volume size and type.</td>
</tr>
<tr>
<td>Exit</td>
<td>volumeModification; modificationState; targetSize; volumeId; targetVolumeType; progress; originalSize; originalVolumeType; originalloqs; targetloqs</td>
</tr>
<tr>
<td>Destination</td>
<td>User mobile interface</td>
</tr>
<tr>
<td>Action</td>
<td>The user press the button Monitor and the mobile app enters the monitor Smart Place screen. On this screen the user has to select a cell that corresponds to a Smart Place and then press the edit button. On the new edit screen the user has to choose other value for volume type and volume size on the pickerviews that are attached to the volume-id that appears for that Smart Place. Then the user has to hit the save button.</td>
</tr>
<tr>
<td>Needs</td>
<td>Existing instance with an existing volume attached.</td>
</tr>
</tbody>
</table>

Figure A.20: SPM Requirement R3.6 - Modify volume size.

<table>
<thead>
<tr>
<th>Function</th>
<th>R.03.7 Modify instance type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>It must be possible to change the instance type to the specified value.</td>
</tr>
<tr>
<td>Entrance</td>
<td>InstanceId; InstanceType;</td>
</tr>
<tr>
<td>Source</td>
<td>Information selected by the user for instance type.</td>
</tr>
<tr>
<td>Exit</td>
<td>requestId; returnflag;</td>
</tr>
<tr>
<td>Destination</td>
<td>User mobile interface</td>
</tr>
<tr>
<td>Action</td>
<td>The user press the button Monitor and the mobile app enters the monitor Smart Place screen. On this screen the user has to select a cell that corresponds to an Smart Place and then press the edit button. On the new edit screen the user has to choose other value for instance type on the pickerview that is attached to the instanceType that appears for that Smart Place. Then the user has to hit the save button.</td>
</tr>
<tr>
<td>Needs</td>
<td>Existing Smart Place.</td>
</tr>
</tbody>
</table>

Figure A.21: SPM Requirement R3.7 - Modify instance type.
### R.03 Manage Smart Place monitoring

**Function**  
R.03.8 Read `VolumeReadBytes` and `VolumeWriteBytes`

**Description**  
It must be possible to read `VolumeReadBytes` and `VolumeWriteBytes` for a given volume attached to an instance.

**Entrance**  
Smart Place Name; Volume Id;

**Source**  
Information inserted by the user for Smart Place name and volume name.

**Exit**  
`VolumeReadBytes`; `VolumeWriteBytes`

**Destination**  
User mobile interface

**Action**  
The user press the button Monitor and then the user selects one instance. On the new screen that is opened the user can visualize these metrics.

**Needs**  
Existing instance on the cloud provider with monitor instances enabled.

---

**Figure A.22: SPM Requirement R3.8 - Read Volume Read and Write Bytes.**

---

### R.03 Manage Smart Place monitoring

**Function**  
R.03.9 Read Latencies for an instance

**Description**  
It must be possible to read latencies from the instance selected.

**Entrance**  
Instance Name; EnvironmentName; AttributeNames:"All"

**Source**  
Information selected manually by the user for instance name;

**Exit**  
metricName; start-time; end-time; period; statistics average; namespace; dimensions.

**Destination**  
User mobile interface

**Action**  
The user press the button Monitor Instance and then the user selects one instance. On the new screen that is opened the user can visualize these metrics.

**Needs**  
Existing instance on the cloud provider with monitor instances option enabled and AWS load balancer enabled.

---

**Figure A.23: SPM Requirement R3.9 - Read Latencies for a SP machine registered on a Load Balancer.**

---

84
### Appendix B

## SPM API Request Calls

<table>
<thead>
<tr>
<th>GET /aws/createInstance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creates an instance on EC2</td>
</tr>
</tbody>
</table>

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>availability-zone</td>
<td>us-east-1</td>
</tr>
<tr>
<td>instance-type</td>
<td>m1.large</td>
</tr>
<tr>
<td>hostname</td>
<td>myinstance1</td>
</tr>
<tr>
<td>os</td>
<td>Amazon Linux</td>
</tr>
</tbody>
</table>

**Response Body (consumes application/json)**

```json
{"InstanceId":"5f9adeaa-c94c-42c6-aee5-28a5376002cd"}
```

**Response HTTP Code**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>400</td>
<td>bad request</td>
</tr>
</tbody>
</table>

Table B.1: SPI API create instance request.
Table B.2: SPI API Start Instance request.

GET /aws/startInstance

Starts an instance on EC2

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance-ids</td>
<td>i-1234567890abcdef0</td>
</tr>
</tbody>
</table>

Response Body (consumes application/json)

```json
{
  "StartingInstances": [
    {
      "InstanceId": "i-1234567890abcdef0",
      "CurrentState": {
        "Code": 0,
        "Name": "pending"
      },
      "PreviousState": {
        "Code": 80,
        "Name": "stopped"
      }
    }
  ]
}
```

HTTP Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>400</td>
<td>bad request</td>
</tr>
</tbody>
</table>
**GET /aws/stopInstance**

Stops an instance on EC2

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance-ids</td>
<td>i-1234567890abcdef0</td>
</tr>
</tbody>
</table>

**Response Body (consumes application/json)**

```json
{
  "StoppingInstances": [
    {
      "InstanceId": "i-1234567890abcdef0",
      "CurrentState": {
        "Code": 84,
        "Name": "stopping"
      },
      "PreviousState": {
        "Code": 16,
        "Name": "running"
      }
    }
  ]
}
```

**Response HTTP Code**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>400</td>
<td>bad request</td>
</tr>
</tbody>
</table>

Table B.3: SPI API Stop Instance request.
Table B.4: SPI API List Instances request.