A Microservice Architecture for Multimobility in a Smart City

Cristian Lai, Francesco Boi, Alberto Buschettu and Renato Caboni
CRS4, Center for Advanced Studies, Research and Development in Sardinia,
Loc. Piscina Manna, Ed. 1 09050, Pula (CA), Italy

Keywords: Internet of Things, Smart City, Transportation, Multimobility.

Abstract: In this paper we present a microservice architecture designed to build IoT services available on the Web. The high potential of microservice architectures will impact the design of complex systems more significantly in the coming years. We propose a draft of architecture that we have used to develop an application for multimobility services in a smart city using an ecosystem of devices. Such application, designed for a real case study, is extremely heterogeneous in terms of IoT devices and implements a wide range of services for citizens. It aims to give a contribution to reducing traffic generated by private vehicles in the city and to help drivers going towards high traffic areas by presenting real-time mobility data from different sources. The evaluation of the architecture carried out in this study allows to understand how it behaves under an increasing number of devices and users connected to the platform, particularly in terms of response time alterations caused by a large number of requests.

1 INTRODUCTION

During the recent years the amount of connected devices available in everyday life has significantly grown as crucial part of the Internet of Things (IoT). ICT is evolving toward more scalable architectures and those based on microservices are paving the way for modern applications (Baškarada et al., 2018; Dragoni et al., 2018; Kalske et al., 2018). This paper presents a microservice architecture specifically designed and developed for the IoT. This architecture is capable of integrating both physical and logical devices as well as of managing typical operations such as device registration, data storage and data retrieval.

Smart cities are more and more enriched with sophisticated services (Zanella et al., 2014), especially for citizens mobility. Multimobility combines different modalities of transportation, e.g., private car, bus, carsharing and bikesharing. The shift from homogeneous to multimodal mobility is growing in popularity, especially in urban centers with recurring problems associated with congestion, parking, and an overall lack of space (Shaheen et al., 2016). Everyday, citizens shift among multiple transportation modalities. Those who move by car may decide to park and proceed by bus or otherwise. This is why their main issues are related to finding a parking slot, catching the bus on time or choosing the suitable alternative. Other sources of information, such as traffic data, are not associated directly to any transportation service for the citizens but they rather supply data that users can consult or that can be exploited otherwise. Every single element, i.e., a parking area, a bus stop, a car or bike sharing station, takes part of an extremely sophisticated network of miscellaneous connected IoT devices that can grow incrementally. Within a scalable system, a single device is handled by a dedicated microservice.

A real case study has been set up in the metropolitan area of Cagliari. Drivers moving by car in the city centre have the available information necessary to elude high traffic intensity areas avoiding time and fuel wasting and also preventing increase of traffic.

In the IoT the volume of devices can be remarkable. It should be noted that by 2020 the IoT could reach 20 billion internet-connected things and it will increment revenues for products and services suppliers up to $300 billion (Hung, 2017). In light of this, scalability is a non-functional feature of the software which requires special attention. Scalable IoT systems ensure indeed adequate response times for data delivery to/from devices and data reading (Bondi, 2000). Microservice architectures, by nature, take into account and handle scalability (Hasselbring and Steinacker, 2017), but need additional efforts to calibrate the necessary number of instances of each microservice, to scale each microservice independently and to synchronize data among them. We have mea-

Lai, C., Boi, F., Buschettu, A. and Caboni, R.
A Microservice Architecture for Multimobility in a Smart City.
DOI: 10.5220/0007830506570664
In *Proceedings of the 14th International Conference on Software Technologies (ICSOFT 2019)*, pages 657-664
ISBN: 978-989-758-379-7

sured system performances, under an increasing number of devices registered to the platform and an increasing number of connected users, to address scalability. We have evaluated the ability of the system to guarantee adequate performance in case of high data volume (resilience) and an appropriate SLA (Service Level Agreements).

This paper is organized as follows. Section II reviews state of the art service oriented architectures. Section III describes the microservice architecture designed and developed for mobility services in smart cities. Section IV illustrates the implementation of a real case study. In Section V the evaluation of the implemented architecture is described. Finally, Section VI provides conclusions and future perspectives.

2 STATE OF THE ART

The enormous amount of connected devices has enabled a widespread diffusion of large IoT systems. Infrastructures, platforms and software applications are offered often as services using cloud technologies (Perera et al., 2013). In IoT systems a centralized architecture is responsible for offering one or more services to the user while the necessary data is produced by a set of devices deployed in different locations. This generates an amount of data easily available, used to create vertical applications. In the IoT, a single standard or implementation has not yet been established, but several competing paradigms are candidates to become the main reference. There are several middleware that act as layers of abstraction between heterogeneous IoT devices and IoT systems enabling interactions between heterogeneous devices. The cloud-based **ThingSpeak** platform (Lai et al., 2018; Abdul-Rahman and Graves, 2016), is an open source IoT application and API (Application Programming Interface) for storing and retrieving data from "things" using the HTTP protocol. The ThingSpeak API allows for numeric data processing such as time scaling, averaging, median, summing and rounding and it is used as a middleware to manage low-level network operations, data collection and real-time data annotation/transformation. It is useful for several real life IoT applications such as smart home applications and IoT agriculture, but it is mostly oriented to collect, analyze and trigger reactions. **Xively**[1] allows to connect devices over MQTT (Message Queuing Telemetry Transport) protocol. Used mainly in agriculture and domotics, it supports many data formats and offers fast and reliable database intelligence. Each source of information must be analyzed individually and the priority of each process must be scheduled individually. This can generate overloads and bottlenecks. **LinkSmart**[2] (Harlamova et al., 2017; Kostelnik et al., 2011), a service-oriented middleware, uses the SOA (Service-Oriented Architecture) architecture to generate services in home automation, healthcare and smart agriculture contexts. LinkSmart allows integration with existing systems and services through widespread standards and solves security problems through virtualization. **OpenHAB**[3] (Open Home Automation Bus) is an open-source middleware, born as an extension of the Eclipse SmartHome project and designed for smart environments development, such as smart offices and smart homes. Any OpenHAB instance contains modules called bundles that run within a single OpenHAB instance.

Many are successful state of art technologies but are monolithic applications built as a single unit, hence not offering the flexibility required to deal with heterogeneous devices efficiently. Often, they consist of three main parts: a client-side user interface, a database and a server-side application. Changes to the system require building and deploying a new version of each component. On the contrary, microservice architectures are divided into a number of small independent services, each of which implements a certain feature. A scalable architecture achieves a more efficient management of available resources by allocating only to those modules that are actually overstressed, rather than unconditionally to all the sub-components (Dragoni et al., 2018; Krylovskiy et al., 2015).

3 MICROSERVICE ARCHITECTURE

The microservice architecture discussed in this paper combines a variety of IoT devices and services. Namely CMC-IoT, is a fork and extends CMC-Core[4], our first open source project implementing a general purpose microservice architecture. CMC-IoT is the foundation of our first web application designed for citizen multimobility in a smart city, called SmartMobility.

[1] https://xively.com/

[2] https://www.linksmart.eu/

[3] http://www.openhab.org/

[4] https://github.com/smartenv-crs4

3.1 CMC-Core

CMC-Core provides independently deployable and loosely coupled basic services. Each microservice runs in its own process, communicates with lightweight mechanisms, such as HTTP resource API (Fowler and Lewis, 2014) and manages significant operations. The microservices composing CMC-Core are the following:

- **Cmc Auth** is a token generator that protects microservices from unauthorized access using a token-based authentication technique. Cmc Auth generates different types of tokens enabling access to specific endpoints of a protected microservice. The three types of tokens that can be generated are: i) *Microservice Token*, to access any resource made available by microservices and to allow communication and exchange data among them; ii) *User Token*, to access resources for user-related services; iii) *Application Token*, to access resources from general purpose third-party applications.
- **Cmc App** manages resources and applications sign-up and the subsequent sign-in phase. When resources or applications sign-in, Cmc App asks for an Application Token to Cmc Auth. Cmc App releases the token to the applicant that consequently can access any protected microservice (MS, see Fig. 1). Moreover, CMC Auth manages any authorization rule to access a specific microservice.
- **Cmc User** manages user access to protected microservices. Users can sign-in with credentials to obtain a user token. This token can be used only by the owner of specific resources related to the protected microservices.

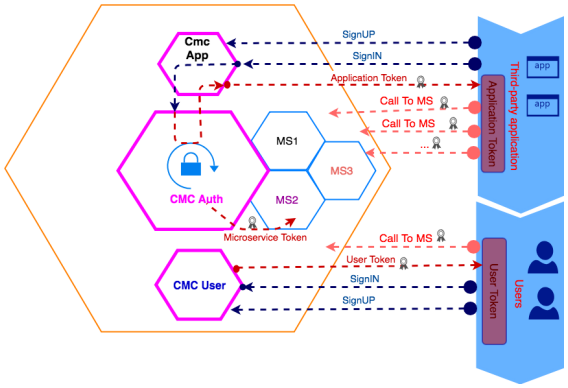

Figure 1: CMC-Core. A typical use case.

In Fig. 1, the microservices MS1, MS2 and MS3 grant access to a third-party application. The application is authorized to call these microservices through an Application Token. When the token expires, the application or user must sign-in again. The same procedure is used to authenticate and authorize user access to one or more microservices.

CMC-Core offers only the presented basic services but can be easily extended with new features using the available plugins. With this approach CMC-IoT has been developed, starting from the architecture of CMC-Core, to have a complete system more suitable for IoT and smart city applications.

3.2 CMC-IoT

CMC-Iot is compound of the following additional microservices:

- **Cmc Devices** manages the device functionalities providing the REST CRUD (Create Read Update Delete) operations. Each device must have a unique id to be unambiguously identified (for example the mac-address), a category and a physical connector. Optionally also its geolocation and a nickname can be specified. A connector allows direct interaction with CMC-IoT. For those devices unable to perform HTTP operations, a middleware (see Fig. 2) is required to mediate the communication to CMC-IoT. Devices can read from and write to Cmc Device using a REST API. External systems are synchronized with Cmc Devices for events such as new available data, addition of a new device to the platform. The most common protocols such as MQTT[5], COAP (Constrained Application Protocol) (Thangavel et al., 2014) and others are directly supported by Cmc Devices. In case of user related devices (e.g., wearable and home smart objects) User Tokens guarantee access to Cmc Devices. General purpose devices, not directly associated to any particular user (e.g., public traffic data, parking, open data), use Application Tokens.
- **Cmc History** stores and retrieves historical data produced by devices. Moreover, it allows to filter and search data by device id, time, device category, type of connector and others. In case of device failure last available data can be recovered.
- **Cmc Persistence** is a scheduler for general purpose devices that do not directly provide their data to CMC-IoT. It performs data reading using Cmc Devices and data saving through Cmc History.

Once a device has been added to CMC Devices and a User Token or Application Token has been released, data can be sent through token authorized

[5]http://mqtt.org/

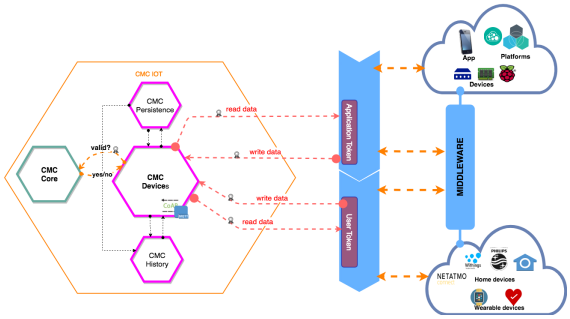

Figure 2: CMC IoT.

write operations and then read by other authorized devices, applications or users. Cmc Devices uses the core features of CMC-Core to validate tokens and to authorize queries on registered devices.

4 CASE STUDY

We have considered a real case study in the city of Cagliari, located in the Sardinia island, Italy. The aim is to contribute to the reduction of traffic generated by private vehicles in the city. SmartMobility, the application developed during this project, helps drivers going towards high traffic areas by presenting in a clear way and on the same platform real-time mobility data from different sources. SmartMobility is made up by the following macro components:

- CMC-IoT (see Section 3.2);
- data sources: any piece of hardware or software (including external services such as Web-APIs and portals) producing relevant data (see Section 4.2);
- Web portal: the user interface presenting the acquired data in an easy and understandable way to the user (see Section 4.3). Even though the best solution is a mobile APP, so far we have decided to develop a web application. This allows to design a complete user interface and quickly mock up a running system. Moreover, the web application with responsive capabilities can be used in mobile devices.

4.1 Scenario

Before entering the city centre the driver, using the SmartMobility application, can check available free parking spots in the monitored parking areas that are close to his destination. In this way he/she does not have to drive around the city looking for a free parking spot. Once a parking lot has been identified and chosen, the user can choose the fastest path to reach it according to real-time traffic information in the main city roads shown in SmartMobility. The driver can again check on the application the availability of mobility services around the parking area, such as bus stops and sharing services, and their reliability so that he/she can choose the one most suitable for his/her needs or alternatively walk to the final destination.

4.2 Data Sources

SmartMobility collects data from different data sources:

- real-time traffic information: the municipality of Cagliari has created an infrastructure of inductive-loop traffic detectors (commonly referred to as traffic sensors) that can detect vehicles passing by. These sensors are installed in the main roads connecting the suburbs to the city center. Data returned by REST APIs contain the flow of vehicles per unite of time and the average speed. This information is used to estimate traffic flow at the time the user is supposed to pass so that he/she have the necessary information to elude high traffic zones;
- parking space vacancy detection system: an automatic system installed in open-access parking lots. Our system uses cameras and image processing for detecting vacant and occupied parking spaces. SmartMobility is constantly updated on parking availability so that users can directly drive to the area closest to their destination where most likely they will find a free parking spot, avoiding in this way time and fuel wasting, and reducing traffic congestion;
- public transport information: most of the public transport companies in Cagliari and its extended area have offered data of their bus services for this project. Bus service information includes available bus lines, bus stops, the timeschedule and the reliability of each line;
- carsharing: a private company offers the fast-growing carsharing service in the city of Cagliari. They provide their service data through web services. Available information includes the current status of each carsharing parking area (number and models of the available cars, booked reservations etc.);
- bikesharing: data include the geographical position of each dock and the number of available bikes in it. Due to temporarily unavailability, data have been simulated keeping the same format.

Each data type has its own format and procedure for being retrieved. To communicate with such het-

erogeneous data sources, CMC Devices uses different types of connectors, one for each kind. A single CMC Devices instance manages many devices in order to avoid computational overhead. CMC Persistence periodically retrieves carsharing data from the proper endpoint while CMC History stores last month traffic data to predict the current traffic flow. In our architecture, the microservices are deployed in Docker[6] containers that provide the needed flexibility and keep the overhead sufficiently low, compared to traditional virtual machine solutions.

4.3 Web Portal

The SmartMobility web portal contains all the elements compounding the UI (User Interface) and corresponding to the variety of devices and services managed by CMC-IoT. The UI shows a map containing the geographical location of the devices (bus stops, parking lots, traffic. sensors, bike docks and car sharing parkings) composing the different mobility services. On the UI, graphic elements of the same kind are grouped together. The user selects the services in which he/she is interested from a drop-down menu (see Fig. 3).

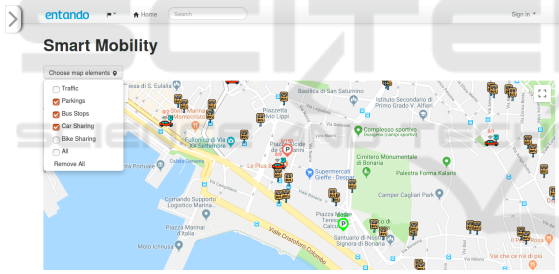

Figure 3: Available services in SmartMobility.

On the map the relevant elements are shown following a layered approach. Each item presents additional information. For example, when selecting a parking lot not only the overall parking capacity and current availability, but also further information related to the other mobility services in the area close to it is shown in real-time. The bus stop selection provides the available lines, the timeschedules and service reliability information. Clicking on a carsharing or bikesharing station displays the number of available vehicles in it. Fig. 4 shows an example in the case of the carsharing service.

The web interface is developed using the enterprise open source software Entando[7]. Entando is an open source Digital Experience Platform (DXP) for

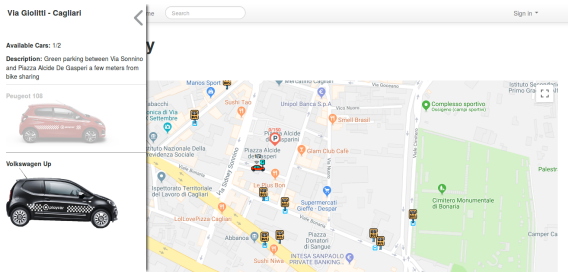

Figure 4: Additional information, carsharing.

vertical portals. The namesake enterprise is also partner of CRS4 in the project that partially supported this work (see Section 6). Extensions available in the platform have been exploited to create the software plugin modules needed for the integration with CMC-IoT and for the web portal development. The web portal is updated in real-time when new data are available on CMC-IoT. The result is pleasant visualization of smart city data on the graphical interface.

5 EVALUATION

We have identified a pattern of actions the user is supposed to perform on the web portal according to the scenario in Section 4.1.

1. Identify the parking lots.
2. Find the traffic sensors close to a chosen parking lot.
3. Find the bus stops, carsharing and bikesharing stations around the identified parking lot.

The supposed actions generate requests to CMC-IoT, referred to as user requests.

Tests simulating the pattern have been defined and performed to understand how CMC-IoT behaves when the number of devices and user requests increase. During these tests we have measured the response time of user requests and the number of failed requests. CMC-IoT has been run on a virtual machine with 8 GB of RAM memory, 4 cores and Ubuntu 18.04 as operating system.

5.1 Pattern of User Actions

The pattern detailed in the Algorithm 1 simulates human interaction with the SmartMobility application according to the scenario of Section 4.1.

First, the algorithm simulates the user destination by randomly choosing a point in the city centre. Then it proceeds with looking for the parking lots close to the destination within a given radius. After randomly

[6]https://www.docker.com
[7]http://www.entando.com

Algorithm 1: Algorithm simulating human interaction with the SmartMobility application according to the scenario described in Section 4.1. User's actions generate requests to CMC-IoT to retrieve data of the desired mobility services. r is the searching radius. The notation $el_i \overset{R}{\leftarrow} l_i$ means randomly choose an element el_i from the list of sensors l_i of type i.

```
p ← random point in the city centre;
r ← 500m;
l_1 ← get from CMC-IoT the parking areas
  around p within r;
if l_1 ≠ ∅ then
    el_1 ←R l_1;
    read el_1 most recent data from CMC-IoT;
    p ← (el_1 geographical position);
    l_2 ← get from CMC-IoT the traffic
      sensors around p within r;
    if l_2 ≠ ∅ then
        el_2 ←R l_2;
        read el_2 current data from CMC-IoT;
    end
    r ← 200m;
    l_3 ← get from CMC-IoT the bus stops
      around p within r;
    if l_3 ≠ ∅ then
        el_3 ←R l_3;
        read el_3 data from CMC-IoT;
    end
    r ← 500m;
    l_4 ← get from CMC-IoT the bikesharing
      docks around p within r;
    if l_4 ≠ ∅ then
        el_4 ←R l_4;
        read el_4 current data from CMC-IoT;
    end
    r ← 300m;
    l_5 ← get from CMC-IoT the carsharing
      stations around p within r;
    if l_5 ≠ ∅ then
        el_5 ←R l_5;
        read el_5 current data from CMC-IoT;
    end
end
```

choosing a parking lot from the retrieved ones, the algorithm detects all the other mobility services, including traffic sensors around it and within a maximum distance. As already explained in the scenario, this allows the driver to choose the fastest path to reach the parking area and the best alternative to move from the parking lot where the driver's car has been parked to the final destination of the user.

The searching radius has been empirically chosen for each device type in order to prevent the pattern from returning empty lists: the larger the radius, the more likely it is to find a sensor of that type around the given geographical point. However it still might happen that no device of that type is found: in this case the pattern will skip the next requests depending on this result and continue with the others, if any, or terminate.

5.2 Test Configuration

Starting from a basic test configuration compound of:

- 440 data sources (5 parking lots, 300 bus stops, 100 traffic sensors, 30 carsharing parking areas and 5 bikesharing docks);
- 10 users;

we perform four rounds of tests, each one consisting of four test runs (see table 1). Within the same round the number of devices is kept constant while the number of users is increased ten times at each test run. In the next round the number of devices is increased ten times and the number of users starts back from 10 as in the basic test configuration. At each test run each user performs the pattern 50 times.

Table 1: Table summarizing the number of devices and users for each test run. Each row is a single round of tests having the same number of devices. Columns are test runs with the same number of users.

Round	Test run							
	1		2		3		4	
	devices	users	devices	users	devices	users	devices	users
1	440	10	440	10^2	440	10^3	440	10^4
2	4400	10	4400	10^2	4400	10^3	4400	10^4
3	44000	10	44000	10^2	44000	10^3	44000	10^4
4	440000	10	440000	10^2	440000	10^3	440000	10^4

5.3 Measures

At each test run we collect the following data:

- average CPU usage of the virtual machine running CMC-IoT;
- average memory usage of the virtual machine running CMC-IoT;
- number of failed requests;
- response time of user requests: the average of time intervals between the moment the client performs a request and the time it gets a response, according to the technique explained in (Hoxmeier and Dicesare, 2000).

The operations depicted in the Algorithm 1 are performed by the Apache JMeter[8] testing tool . It

[8]https://jmeter.apache.org

allows to perform the requests to CMC-IoT and to store for each request the desired response information, such as response code and response time, on the user-side hardware. The average CPU usage and average memory usage are monitored with NetData[9] on the server-side (the virtual machine running CMC-IoT).

5.4 Results

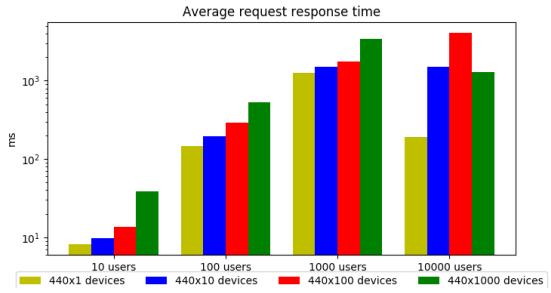

Figure 5: Average response time of user requests for each test run.

According to JMeter all requests are successful and no failure occurs. Fig. 5 shows the average response time of user requests for each test in logarithmic scale. Each bar is the average response time of user requests for the corresponding test run. Results of runs having the same number of users are grouped together, while results of test runs with the same number of devices have the same color. It is clear from the graph how the average response time of the requests increases with the number of users and with the number of devices. However in the group with 10000 users, we have an unexpected behaviour: within this group the test run with 440x1000 devices (the green bar in the last group) has an average response time smaller than the test runs (within the same group) with 440x10 (the blue bar) and 440x100 (the red bar) devices. Furthermore compared to the group of 1000 users, the test runs having a number of devices equal to 440x1, 440x10 and 440x1000 have a smaller average response time even of they have a larger number of users. Additional investigations will be carried out in future works to understand this behaviour but the hypothesis is that more computational power is needed to simulate such a large number of users.

As for the server, used memory is around 4 GB and it remains constant among all the tests. This indicates more instances of the microservices can be created even with this hardware configuration. The graph for the CPU percentage usage is reported in Fig. 6.

[9]https://my-netdata.io

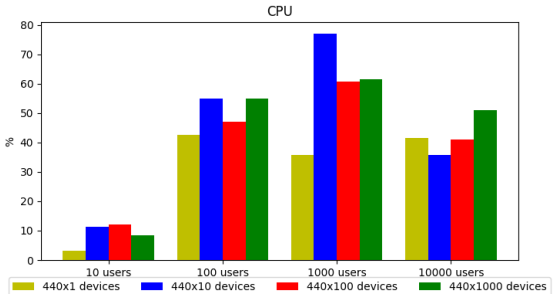

Figure 6: Average CPU usage percentage for each test run.

6 CONCLUSIONS

IoT in smart cities is becoming pervasive and will increase even more in the coming years. The network of connected devices requires more efficient infrastructures, platforms and services in the IoT ecosystem. Microservices, in contrast to aged monolithic architectures, have been proved to be a valid and robust solution for modern complex scenarios, and have been recognized to accelerate the evolution of software application models.

Our application is a demo validating the effectiveness of the drafted microservice architecture in a real case study. All the devices and services included in this specific implementation compound a comprehensive miscellaneous system. The potential of the adopted microservice architecture allows to design modular systems that can grow incrementally without continuous redesign, development and deployment of the entire application. The system is divided into small and lightweight services, purposely built to perform a very cohesive business function. Every single element, i.e., a parking area, a bus stop, a car or bike sharing station could be added as well as removed independently, while the system can be further enriched with new services. Both these actions can be performed by modifying only the directly interested modules without affecting the others. Services independence allows to reach for an extremely sophisticated network of connected IoT devices. On the contrary, in monolithic applications scaling highly demanded services requires a comprehensive effort in the whole system. All services are scaled at the same time, with resources allocated even for services barely used. Presently SmartMobility has been developed as a web portal. In the future we intend to design a more suitable APP for mobile devices that will perfectly fit the user experience.

Several tests have been performed to determine performances under an increasing number of requests per unit of time. Goals of these tests are to under-

stand how many concurrent requests the presented microservice architecture can handle given limited hardware resources and how scalability should be addressed. Regarding the scalability, as a follow up, additional tests will aim at finding out which services are more stressed and according to the results a proper algorithm for managing scalability will be studied and implemented. In future works scalability will be properly automated according to these results and each microservice will be instantiated according to its workload and independently from the others. When multiple instances are present, workload distribution and data synchronization among different instances must be addressed too.

ACKNOWLEDGEMENTS

This work has been partially supported under the ERDF (European Regional Development Fund), PO Sardegna FESR 2007-2013, PIA 2013 (project: n.295 PIA ENTANDO).

REFERENCES

Abdul-Rahman, A. I. and Graves, C. A. (2016). Internet of things application using tethered msp430 to thingspeak cloud. In *SOSE*, pages 352–357. IEEE Computer Society.

Baškarada, S., Nguyen, V., and Koronios, A. (2018). Architecting microservices: Practical opportunities and challenges. *Journal of Computer Information Systems*, pages 1–9.

Bondi, A. B. (2000). Characteristics of scalability and their impact on performance. In *Proceedings of the 2Nd International Workshop on Software and Performance*, WOSP '00, pages 195–203, New York, NY, USA. ACM.

Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., and Safina, L. (2018). Microservices: How to make your application scale. In Petrenko, A. K. and Voronkov, A., editors, *Perspectives of System Informatics*, pages 95–104, Cham. Springer International Publishing.

Fowler, M. and Lewis, J. (2014). Microservices.

Harlamova, M., Kirikova, M., and Sandkuhl, K. (2017). A survey on challenges of semantics application in the internet of things domain. *Applied Computer Systems*, 21(1):13–21.

Hasselbring, W. and Steinacker, G. (2017). Microservice architectures for scalability, agility and reliability in e-commerce. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 243–246.

Hoxmeier, J. A. and Dicesare, C. (2000). System response time and user satisfaction : An experimental study of browser-based applications.

Hung, M. (2017). Leading the iot. Technical report, Gartner.

Kalske, M., Mäkitalo, N., and Mikkonen, T. (2018). Challenges when moving from monolith to microservice architecture. In Garrigós, I. and Wimmer, M., editors, *Current Trends in Web Engineering*, pages 32–47, Cham. Springer International Publishing.

Kostelnik, P., Sarnovsky, M., and Furdik, K. (2011). The semantic middleware for networked embedded systems applied in the internet of things and services domain. *Scalable Computing: Practice and Experience*, 12(3):307–315.

Krylovskiy, A., Jahn, M., and Patti, E. (2015). Designing a smart city internet of things platform with microservice architecture. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 25–30.

Lai, C., Pintus, A., and Serra, A. (2018). Using the web of data in semantic sensor networks. In Barolli, L. and Terzo, O., editors, *Complex, Intelligent, and Software Intensive Systems CISIS 2017. Advances in Intelligent Systems and Computing, vol 611. Springer, Cham*, pages 106–116.

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2013). Sensing as a service model for smart cities supported by internet of things. *Transactions on Emerging Telecommunications Technologies*, 25(1):81–93.

Shaheen, S., Stocker, A., and Bhattacharyya, A. (2016). Multimobility and sharing economy. Technical Report E-C120, Transportation Research Board, 500 Fifth Street, NW, Washington, DC 20001.

Thangavel, D., Ma, X., Valera, A., Tan, H.-X., and Keng-Yan Tan, C. (2014). Performance evaluation of mqtt and coap via a common middleware.

Zanella, A., Bui, N., Castellani, A., Vangelista, L., and Zorzi, M. (2014). Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1):22–32.