A microservice-based platform for IoT application development

Guido Porruvecchio, Alessandro Romanino, Carlino Casari, Raffaella Sanna
CRS4 - Center for Advanced Studies, Research and Development in Sardinia
Loc. Piscina Manna, Edificio 1, 09050 Pula (CA) Italy
Email: {guido, aromanino, casari, raffa}@crs4.it

I. INTRODUCTION

Over the past years, software architectures has shifted away from the traditional monolithic approach. An urgent need for systems to be more scalable, resilient, efficient and cloud-ready, paved the way for modular architectures, which are nowadays largely adopted in many domains and drive the development of modern software systems. In this context, Microservices represent a state-of-the-art architectural model [1] [2], especially in domains where distributed and pervasive computing is of paramount importance, as in Internet of Things [3]. IoT systems are built upon a large number of connected devices communicating through the Internet, with strict requirements for device coordination and for management of a large amount of device generated data.

This paper is structured as follows. Section II analyzes the state of the art of IoT platforms and middleware, with a focus on their architectural peculiarities. Section III introduces CRS4 Microservice Core (CMC), a microservice-based framework for the development of modular and scalable applications, created at CRS4 research center. Section IV presents CMC-IoT, an IoT platform developed as a service of CMC framework, and compares its software architecture with an IoT reference architecture. Further developments and final considerations are discussed in Section V.

This work was partially supported by Italian Ministry of University and Research (MIUR), within the Smart Cities framework (Project Cagliari 2020, ID: PON04a2-00381)

II. BACKGROUND AND RELATED WORK

As soon as a variety of heterogeneous connected devices became the backbone of pervasive and ubiquitous networks, emerged the necessity for middleware systems that could act as an abstraction layer between applications and Things, integrating additional services [4] and supporting interoperability among these applications and services [5]. As a consequence, many IoT platforms and middleware were developed, and industry and academia made an effort to survey these systems, classifying them with respect to their architecture [6] as well as functional/non-functional features [5] [4]; furthermore, objective criteria were generally defined for their evaluation and adoption in different scenarios. An insight on cloud IoT platforms is given in [7], [8] focuses on middleware particularly suited for application development, while a comparative analysis on Open Source IoT middleware platforms is performed by [9]. [4] underlines the centrality, in IoT platforms and middleware, of the different aspects lying under the notion of interoperability [10], which is one of the requirements for implementing the abstraction functionalities. Network interoperability acts as a bridge between the different communication protocols used by Things; syntactic interoperability deals with the format of data exchanged among Things; semantic interoperability defines the semantic domain-specific model, which establishes the rules for understanding the meaning of information content. The semantic model will be then translated into a data model used by the concrete software system.

[11] proposes a classification of the different types of architecture for IoT middleware, which are identified as SOA, cloud-based and actor-based. Service Oriented Architectures represent a well-established approach in software design [12], and in IoT systems as well [13] [14] [15] [16]. Microservices further developed and refined SOA concepts, focusing on the subdivision of large monolithic applications in small, highly decoupled, independently deployable and scalable services, performing a single, well-defined and domain-oriented task, and communicating among themselves by means of a simple protocol, e.g. HTTP [1] [2].

A comprehensive review on the adoption of microservices for IoT systems can be found in [17]; previous studies already highlighted how well microservices could help IoT platforms to overcome limitations of SOA approach, such as scalability.
and maintainability [18] [19], providing the design principles and technological tools to build distributed and large-scale applications deployable in the cloud [18] [20] [21] [22]. Consequently, several IoT platforms in various domains applied microservice architectural style, gaining benefits in terms of resilience, scaling, modularity, heterogeneity and independence of technology [23] [24] [25] [26] [27] [28] [29]. In the next sections we propose our contribution to the above cited challenges, represented by a microservice architecture acting as a framework for building scalable and cloud-ready services and applications, and an IoT platform deployed as a microservice on top of that architecture.

III. CMC - CRS4 MICROSERVICE CORE

CRS4 Microservice Core (CMC)\(^1\) is a high-level and general purpose framework, built upon a microservice architecture and conceived for supporting the development of vertical services and applications. It provides cross-cutting functionalities, each of them deployed as a microservice, that allow the developers to focus on the business logic of their application. These basic microservices, called “Core” services, are the following:

- **CMC-Auth** is the central service of CMC. It takes care of service registration, token management and authorization to all services and vertical applications. Three categories of JWT-Base64 tokens are generated by CMC-Auth:
  - **Microservice token**, authorizing communication among internal microservices
  - **User token**, granting access to authenticated CMC users on registered services
  - **Application token**, granting access to third-party applications on registered services

CMC-Auth also implements a rule engine to manage access to every single HTTP resource exposed by services. Each token belonging to one of the three categories can also have a type defining authorizations with a finer level of granularity.

- **CMC-User** exposes a number of features for user management. It takes care of user registration and authentication, providing a user token (generated by CMC-Auth) required to access a protected resource (if the access to that resource has been enabled by Auth service)
- **CMC-App**, similarly to CMC-User, allows the complete management of third-party applications, and in general any system calling CMC services

This architecture allows a seamless extension with further microservices, which can be registered in the platform and communicate with other CMC services or the external world (Fig. 1). In order to be integrated in CMC, a service must comply with few specifications:

- It must expose an authenticated API, using CMC-Auth
- It must be registered in CMC using CMC-Auth

Once registered in the platform, any service can talk to other CMC services via REST API, if allowed by authorization rules managed by CMC-Auth. It can also independently scale according to user’s needs.

IV. CMC-IoT

CMC-IoT\(^2\) is a general purpose IoT platform and middleware, developed as a custom service compliant with CMC specifications. In literature there is already a platform named CMC-IoT proposed by CRS4 [30]; however, this paper presents a complete rework of that system, from both architecture and features perspective.

CMC-IoT is basically a middleware supporting a wide class of devices, since it implements an abstraction between the Things and the applications interacting with them. Therefore, it provides a uniform interface to Things, which can be native compatible (e.g. a sensor developed in-house) or need a specific driver or connector to be integrated in the platform.

In the system architecture (Fig. 2), Connector Middleware depicts the logical block encompassing any kind of connector available.

![Fig. 1. CMC services.](image)

Fig. 2 in its entirety describes a four layer architecture:

- The base layer is composed by *Things*, the physical objects hosting one or more *Devices*, where a Device is a sensor capable of performing a single physical measurement (e.g. temperature, humidity, voltage) or an actuator

---

\(^1\)https://github.com/smartenv-crs4/cmc

\(^2\)https://github.com/smartenv-crs4/cmc-iot
A. Reference architecture

At the beginning of Section IV, the CMC-IoT architecture was described in order to give an insight into the functional decomposition of the system, and how the modules interact and exchange data. Several IoT reference architectures, as well as models and taxonomies, have been previously proposed with the aim of providing frameworks and conceptual models that could be generic and flexible enough to be adopted and supported by concrete software platforms [31] [32] [33]. In this paper the reference architecture proposed by [6] has been chosen to perform a mapping onto CMC-IoT architecture. By adopting abstract concepts and a common terminology, the platform will be better described, capturing it in a framework that, as [6] stated, could help in comparing the existing platforms in terms of features and further developments. Every concept of the Reference Architecture (RA) will be mapped onto each logical block of CMC-IoT architecture, evaluating the similarities and differences of “overlapping” elements (Fig. 3).

Starting from the bottom layer, a Sensor in RA represents a hardware component that captures information from the physical environment, while an Actuator manipulates that environment. So they fit almost perfectly with the CMC-IoT concept of Device, which performs a single measure or a single action. The RA Driver takes care of communication between external systems and those RA Devices that are not capable of managing that kind of connection. This task perfectly fits with CMC-IoT Connector Middleware; the only difference lies in the possibility, for a CMC-IoT connector, of running inside a Thing or in an external system. In the former case, that Thing would be CMC native compatible, and the connector would be acting more as a RA Driver rather than a middleware.

The IoT Integration Middleware in the Reference Architecture models an integration layer for Sensors, Actuators, Devices

As a general purpose IoT platform, CMC-IoT can be adopted in a broad range of scenarios; its customizable data model is based on the concept of Device Type, allowing users to map their own devices onto custom categories with custom properties. As an example, CMC-IoT could manage a network of hygrometers placed in a humidity controlled environment; a Hygrometer device type will be defined, associated with a Relative Humidity observed property and a Percentage unit of measurement. In an analogue manner, a smart mobility application could acquire traffic information from sensors registered in CMC-IoT as belonging to Traffic Sensor device type, observing the Number of Vehicles property.
and Applications, thus enabling a common interface to them, abstracted from their actual implementation. With respect to this concept, the CMC-IoT REST API performs the same task by means of HTTP protocol, while other protocols may be adopted in the RA. The exposed API is the interface to some common IoT functionalities provided by both architectures, mainly the management of Devices/Things and sensor data. Further features are made available, but in CMC-IoT some of them are provided by means of the CMC environment, with microservices each exposing a narrow set of features, for example the core functionalities of authentication and user management. Other higher level features would need ad-hoc microservices, such as graphical dashboards or business intelligence on sensor data.

Finally, the RA Application represents any software system which calls the IoT Integration Middleware to use the connected Devices, so any third-party application (whose access is managed by CMC-App) or CMC service invoking its REST API is fully equivalent.

V. CONCLUSION

CMC-IoT as a service satisfies the main functional requirements of an IoT platform, such as abstraction, context awareness and resource management; at the same time, it is deployed in the context of CMC, a general-purpose and microservice-based framework addressing non functional requirements in terms of availability, scalability, security [5]. From this perspective, the overall CMC-IoT architecture can be defined as a two-layer framework: the high level provides vertical features to ease the development of modern IoT applications and services; these features have their solid foundations in the lower level, a microservice architecture that takes care of all the cross-cutting concerns that every modern application has to address, allowing developers to focus on the implementation of domain functionalities.

The purpose of mapping the CMC-IoT architecture onto a generic and more abstract reference is to describe it in a uniform and consistent way, by using a common basis of concepts and terms that ease the understanding of its main features and the comparison with other IoT platforms.

Future work will expose a more in-depth review of the IoT platform technological features, as well as the implications of a microservice approach in terms of performance and resilience of the system. Furthermore, a real-world use case of CMC-IoT will be presented in order to better clarify how an application can benefit from its adoption.

REFERENCES


