Deployment of APIs on Android Mobile Devices and Microcontrollers

Sergio Laso*, Marino Linaje*, Jose Garcia-Alonso*, Juan M. Murillo* and Javier Berrocal*
*University of Extremadura
Caceres, Spain
Email:{slasom, mlinaje, jgaralo, juanmamu, jberolm}@unex.es

***Abstract*—The high penetration and acceptance of smart devices has encouraged the development of IoT applications. The increase in the capabilities of these final devices has also led to the development of paradigms such as Fog and Edge Computing. Through these new architectural paradigms, developers can exploit the capabilities of the end devices to store, process and provide services, in order to improve the application by reducing the response time and the network overload. Currently, most applications are developed following a Server-Centric architecture because there are a lot of tools facilitating their development. However, these types of tools are not available for these emerging paradigms, which means an extra effort for developers. This paper shows a tool that semi-automatize the generation of applications based on the Edge Computing paradigm, thus reducing the developers' works and the application of these new architectural paradigms.**

***Index Terms*—Microservices, Android, Microcontroller, OpenAPI, Edge Computing**

I. Introduction

During the last few years, the capabilities of Internet-connected devices have increased enormously. Their processing and storage capacity have multiplied in order to be able to sense and/or process more information [1]. This increase has led to the development of the Internet of Things paradigm [2], in which these devices are used and coordinated to make people's life easier.

IoT applications require devices to be almost constantly sensing the environment, processing the gathered data and executing some actions for adapting the environment. For these applications to be really useful and accepted by end-users, they have to fulfill very stringent requirements regarding response time, location-awareness, etc. [3].

To meet these requirements, new paradigms such as Fog Computing, Edge Computing or, even, Mobile Centric architectures [4] have arisen for deploying applications closer to the end-user [5]. Fog Computing aims to to provide computing, storage and networking services between end devices and traditional cloud computing data centers. On the other hand, Edge Computing brings computational and storage capacities to the edge devices. Among other benefits, these paradigms reduce the network load and dependency on the cloud environment; improving the response time and providing better user experience. For example, using data closer or even inside the end device.

Currently, the predominant paradigm for deploying applications or APIs has been the Server-Centric or Cloud-Centric. In which the data gathered by IoT devices is sent to the cloud where it is stored and processed. Initially, that architectural style was applied because Internet-connected devices did not have enough capabilities for processing the gathered data [6]. It is still the most applied and used architectural style. This is because currently there is a myriad of tools and services facilitating the development of applications applying this style. For instance, OpenAPI [7] allows one to generate some of the code for both the server and the client from an application specification. Likewise cloud environment providers such as AWS [8] or Google [9] provide different tools.

Nevertheless, the number of tools providing support for the application of Fog, Edge or Mobile-Centric styles is much lower. This is because end devices or IoT devices were initially conceived as simple clients, not as cloud environment. In addition, in order to improve the security, they have a closed operating system. Therefore, implementing application making use of these paradigms require an extra effort by developers and a higher cost for software companies.

In this paper, we present a tool that helps developers to implement and deploy APIs on end devices, in order to easily apply the Edge and Mobile-Centric paradigms [10]. Concretely, it allows developers to deploy an API on Android devices and microcontrollers. To that end, common tools that are usually used by developers when they apply a Server-Centric paradigm have been adapted, reducing the impact and the learning curve. The presented approach requires an application designed with the OpenAPI specification. That design is used for generating the scaffolding of the application and all the communication interface required for deploying it on end devices.

The rest of the paper is structured as follows. Section II shows several related works. Section III explains the tool. Subsection III-A explains the process of deploying applications to end devices. Section IV describes a demonstration example. Finally, Section V details the conclusions.

II. Related Work

Few commercial tools support architectural styles such as Edge or Fog-based styles like AWS IoT Greengrass [11] or Azure IoT Edge [12]. However, at the research level, more and more proposals are presented fostering the use of the available specifications and resources to facilitate the development process of any application following these paradigms. For

instance, Noura et al. propose the WoTDL2API tool, which automatically generates a RESTful API for controlling different IoT devices [13]. This API is generated from an OpenAPI-based specification and using its toolchain for code generation. The generated source code can be deployed on peripheral devices, such as gateways, routers, etc. Nevertheless, it cannot make use of the capabilities of end devices.

In [14], the authors present a virtualization of IoT devices to improve response time for intelligent city applications. In the proposal, they apply a shared use of microservices and dynamic scaling of resources to improve the use of computing resources and the quality of the application. In [15], the authors also present a modular and scalable architecture based on lightweight virtualization. The modularity provided, combined with the orchestration provided by Docker, simplifies the administration and enables distributed deployments, creating a highly dynamic system. Both approaches distribute the computing load among different layers. However, the data is stored in a cloud layer. Besides, they do not provide tools for developers to easily apply their approaches.

In this paper, we present the modification of existing tools for applying the server-centric architectural style in order to also be able to deploy APIs on end devices, making the application of the Edge paradigm easier.

III. APIGEND - API Generator for End Devices

Currently, the specification and development of microservices are supported by a large number of tools that facilitate the work of the developer. Specifications such as OpenAPI are widely used to detail an API, generate documentation, perform tests and even, generate the API skeleton. This type of tools can generate the source code for different technologies such as Node.JS, Kotlin, JAX-RS, etc., but they focus on deploying the API on a server or on a cloud environment.

This section shows an extension of the OpenAPI Generator [16] to create and deploy an API on Android-based end devices (from API 23) and microcontrollers (ESP32), simplifying the development of these applications. In this way, these APIs can be consumed by third parties as if they were deployed on a cloud environment.

A. Deployment Process

APIGEND is part of a set of steps you need to follow to deploy APIs on end devices. This process can be seen in Figure 1. These steps are:

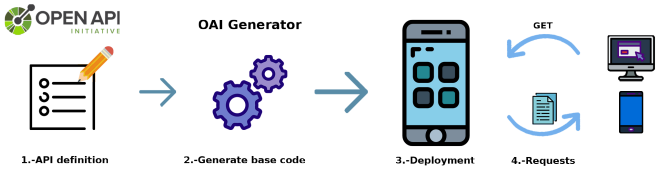

Fig. 1. Deployment Process.

1) **API Definition.** In this step, the API characteristics are defined, this task will be done through the OpenAPI Specification following the same notation as if it were developed and deployed on a cloud environment. The API specification will describe the different resources and endpoints that will be available to be invoked.
2) **Generate Base Code.** In this step, the API base code is generated with the presented tool. The tool also generates the source code to simulate the communication logic of an API Rest, reducing the effort required by developers. This communication logic has been implemented using Firebase Cloud Messaging [17] (for Android devices only) and MQTT [18], being able to decide during the generation of the source code what type of communication should be used.
3) **Deployment.** In this step, the developers only have to finish the implementation of the generated API and configure the messaging services.
4) **Requests.** Finally, with the deployed application, one can invoke the defined resources.

IV. Demo Case

In this demo, we will show attendees how to perform the whole process to deploy an API in an Android device and ESP32 Microcontroller and invoke the available services. We are going to define an API called Event Alerts. This API serves to post different cultural events within an area. You can also get the preferences of different users to organize events more adapted to them. Each device will represent a fictitious user and, therefore, each one will be able to alert with an event to the other users. They will also be able to obtain the preferences of each one to know which event is the most suitable to organize.

The steps to be realized in the demonstration are described below (following the steps described in Subsection III-A).

1) **API Definition.** We define the API with the OpenAPI Specification (OAS). This API will contain two microservices or endpoints:
   a) *Post Events:* This microservice is a POST type operation in which in the *body* of the request an object Event is included, which contain the information of the event and the area (lat,long,rad) in which we want to publish the event.
   b) *Get Users:* This microservice is a GET type operation that will take as parameters the area from which we want to get the users' preferences. This microservice will return the the user's information in an object detailing the user's id and a list of strings with their preferences.
2) **Generate Base Code.** In this step, the API base code is generated with *APIGEND*. For this demo, the MQTT communication interface will be used, as is shown in Figure 2.
3) **Deployment.** In this case, the MQTT communication interface of both devices will be configured, indicating the IP address and port of the MQTT broker and a small implementation will have to be made to send a predefined request from both devices.

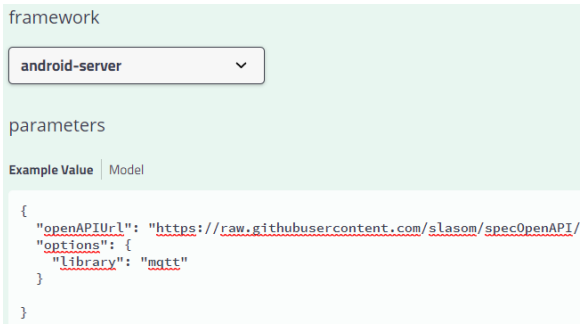

Fig. 2. Parameters to generate the Android application using the MQTT library.

4) **Requests.** After the deployment, we will be able to make requests. In this demo, both devices will be able to send predefined requests, the Android device will do it through a button in the application and the microcontroller through a physical button. As for the reception, the Android device will receive a push notification and will display the information in the main screen of the application as shown in Figure 3. The microcontroller will display the alert information a screen associated to the microcontroller.

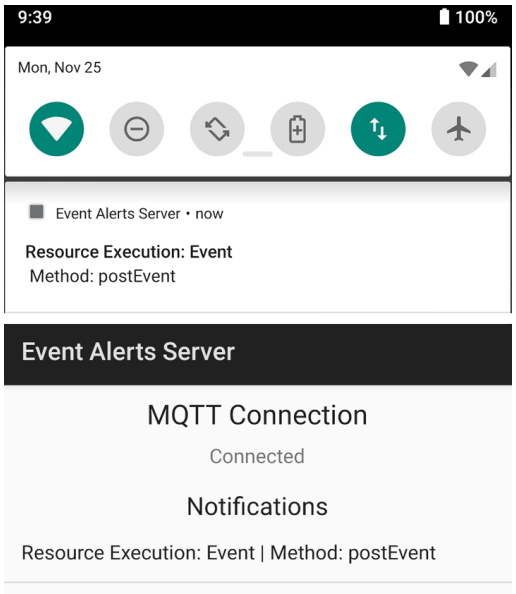

Fig. 3. Notifications received on the Android device

V. Conclusions

The increase of the computing capacities of the end devices opens the possibility to the application of architectural paradigms that until recently could only be used in very specific scenarios, obtaining benefits such as lower latency and lower consumption of network traffic. This paper discusses the lack of tools to help in the development of these alternative architectures. Therefore, it presents a tool that automates the creation of the skeleton of an API in order to be easily deployed on final devices such as Android devices or microcontrollers. This approach is based on tools weel-known by developers, so it reduces the learning curve and expands the target audience that could apply it.

Acknowledgment

This work was supported by the 4IE+ project (0499_4IE_PLUS_4_E) funded by the Interreg V-A España-Portugal (POCTEP) 2014-2020 program, by the Spanish Ministry of Science, Innovation and Universities (MCIU) (RTI2018-094591-B-I00) (MCIU/AEI/FEDER, UE), by the Department of Economy and Infrastructure of the Government of Extremadura (GR18112, IB18030), and by the European Regional Development Fund.

References

[1] J. Berrocal, J. Garcia-Alonso, C. Vicente-Chicote, J. Hernández, T. Mikkonen, C. Canal, and J. M. Murillo, "Early analysis of resource consumption patterns in mobile applications," *Pervasive and Mobile Computing*, vol. 35, pp. 32 – 50, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574119216300797

[2] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad hoc networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[3] H. Qian and D. Andresen, "Extending mobile device's battery life by offloading computation to cloud," in *2015 2nd ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft 2015, Florence, Italy, May 16-17, 2015*, A. Abadi, D. Dig, and Y. Dubinsky, Eds. IEEE, 2015, pp. 150–151.

[4] J. Guillén, J. Miranda, J. Berrocal, J. Garcia-Alonso, J. M. Murillo, and C. Canal, "People as a service: A mobile-centric model for providing collective sociological profiles," *IEEE Software*, vol. 31, no. 2, pp. 48–53, 2014.

[5] K. Dolui and S. K. Datta, "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing," in *2017 Global Internet of Things Summit (GIoTS)*. IEEE, 2017, pp. 1–6.

[6] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini, and A. Zanni, "A survey on fog computing for the internet of things," *Pervasive and Mobile Computing*, vol. 52, pp. 71 – 99, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574119218301111

[7] "The OpenAPI Specification Repository. Contribute to OAI/OpenAPI-Specification development by creating an account on GitHub." [Online]. Available: https://github.com/OAI/OpenAPI-Specification

[8] "Amazon Web Service." [Online]. Available: https://aws.amazon.com/

[9] "Amazon Cloud." [Online]. Available: https://cloud.google.com/

[10] "Openapi Generator Spilab." [Online]. Available: https://openapi-generator-spilab.herokuapp.com/

[11] "Aws IoT Greengrass." [Online]. Available: https://aws.amazon.com/greengrass/

[12] "Azure IoT Edge." [Online]. Available: https://azure.microsoft.com/services/iot-edge/

[13] M. Noura, S. Heil, and M. Gaedke, "Webifying heterogenous internet of things devices," in *International Conference on Web Engineering*. Springer, 2019, pp. 509–513.

[14] K. Ogawa, K. Kanai, K. Nakamura, H. Kanemitsu, J. Katto, and H. Nakazato, "Iot device virtualization for efficient resource utilization in smart city iot platform," in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2019, pp. 419–422.

[15] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for iot using docker and edge computing," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 118–123, 2018.

[16] "Openapi Generator." [Online]. Available: https://github.com/OpenAPITools/openapi-generator

[17] "Firebase Cloud Messaging." [Online]. Available: https://firebase.google.com/docs/cloud-messaging

[18] "MQTT." [Online]. Available: http://mqtt.org/