Sinplafut: A microservices – based application for soccer training

Sinplafut: A microservices – based application for soccer training

F H Vera-Rivera[1,2], J L Vera-Rivera[3,4], and C M Gaona-Cuevas[2]
[1] Grupo de Investigación y Desarrollo de Ingeniería de Software, Universidad Francisco de Paula Santander, San José de Cúcuta, Colombia
[2] Grupo de Estudios Doctorales en Informática, Universidad del Valle, Cali, Colombia
[3] Grupo de Investigación en Deporte de Rendimiento, Universidad del Valle, Cali, Colombia
[4] Fundación of Researchers in Science and Technology of Materials, Colombia

E-mail: fredyhumbertovera@ufps.edu.co, fredy.vera@correounivalle.edu.co

Abstract: Microservices are an architectural style of service-oriented applications that allow the application to be divided into independent and autonomous units, which can be individually developed, tested, deployed (mainly in the cloud), scaled and monitored. The application then becomes a composition and integration of small distributed systems. In this paper, a case study is presented, where the Information system for the planning of soccer training, is developed, following the architectural style of microservices. It presents the architecture defined for the information system and its implementation using DevOps practices, among them, continuous delivery, continuous deployment and automated tests. The level of granularity of each microservice is evaluated using domain-driven design and the definition of delimited contexts. Defining the optimal size of microservices is fundamental, directly affecting performance, maintainability, storage, transactions, distributed queries, use and consumption of network and computational resources. On the other hand, the development of the information system allows improving the planning of soccer training using modern sports training techniques. System for the planning of soccer training can be used by soccer teams, clubs, physical trainers, coaches and technical directors both amateur and professional, as a software as a service application.

1. Introduction

Software as a SaaS service is a service delivery model of cloud computing that provides on-demand applications through the Internet. Examples of SaaS providers include Salesforce.com, Rackspace, SAP Business ByDesign, Google apps, and others [1]. The applications work in the cloud, on demand, you pay for use under the service modality, for your access a browser or specialized client is required through the internet. The software service is paid monthly and paid for what the user hires and consumes. The advantages for users and clients of using software as a service are mainly: they require low investments to use the applications and in specialized hardware, they avoid the problems of support and maintenance, short learning curves, software always updated and by subscription [2].

Quality attributes are fundamental and propose challenges for SaaS applications. Availability, performance, automatic scaling, automated testing, continuous integration and continuous deployment, security and fault tolerance are essential features that every SaaS application must handle. The microservices architecture helps reduce the complexity of managing and operating these features. The development of applications based on microservices allows permanent, faster and automated updates

using the practices of DevOps (Dev: Developers - Ops: Operations), achieving shorter, automated and quality deliveries. The microservice systems require a more sophisticated DevOps infrastructure, which generally requires the construction of a pipeline of continuous implementation and continuous deployment that guarantees the quality of the microservices and faster production. The use of cloud technologies, especially containers, allows the construction of those pipes. The architectural style of microservices changes the way applications are created, tested, implemented and maintained. Microservices facilitate the migration of applications to the cloud infrastructure, which allows automatic scaling, load balancing and fault tolerance. By using microservices, a large application can be implemented as a set of small applications (microservices) that can be developed, deployed, expanded, managed and monitored independently. Agility, cost reduction and granular scalability entail some challenges such as the complexity of managing distributed systems [3].

The constructions of SaaS applications using microservices pose research challenges and discussion topics, for example, the use of multitenant architectures with microservices. A software application as a multi-tenant service satisfies the needs of multiple user groups, organizations or departments, allows the instances of the application to be shared by multiple clients; these instances can be configured and adapted to meet the requirements of specific clients [4].

This work details the development of the information system for the planning of the soccer training (Sinplafut), a SaaS type application that allows to realize the planning of the sessions training in the soccer using modern techniques of the sport training. Sinplafut can be used by soccer teams, clubs, physical trainers, coaches and technical directors both amateur and professional level. Sinplafut developed following the approaches made in [5], where a process of development of microservices – based applications is proposed and an overview of Sinplafut is made. The continuous deployment automation method proposed in [6] was also used. This paper is organized as follows. Section 2 presents the methods used in this research work, section 3 presents the characterization of the development process of microservices – based applications and the way it was used in the development of Sinplafut, then in section 4 the case study is presented, finally in section 5 we summarize the conclusion of this work.

2. Methods

The methodology of this research work is based on the approaches of Wohlin, *et al*, who propose the guidelines for experimentation in Software Engineering. The area of software engineering involves development, operation, and maintenance of software and related artifacts. Research on software engineering is to a large extent aimed at investigating how development, operation, and maintenance are conducted by software engineers and other stakeholders under different conditions. Individuals, groups and organizations, carry out software development, and social and political questions are of importance for this development. The case studies are fundamental and appropriate in this research process [7].

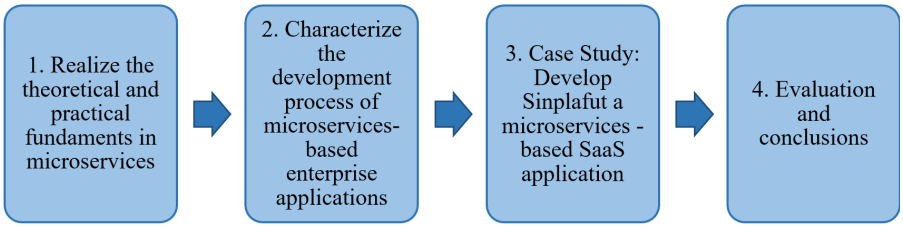

Figure 1. Research model.

Case study in software engineering is: an empirical enquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its reallife context, especially when the boundary between phenomenon and context cannot be clearly specified [8]. In this work we want to evaluate the process of development

of microservices – based applications proposed in [5], the case study consists in implementing Sinplafut, a SaaS application following this development process. Once the case study is completed, the development process is evaluated, improvements are proposed and conclusions are given. The Figure 1 shows the research model carried out in this work.

3. Characterization of the development process of microservices-based applications

The process is divided into two fundamental parts, first the development of each microservice and second, the development of applications that use those microservices. The process is presented in the Figure 2. Some adaptations were made to the process presented in [5]; here we included the tools that allow the services discovery and the tools that allow managing the sidecars. The sidecar is an auxiliary process that runs next to the application; it runs next to each microservice, in order to provide additional features such as fault tolerance, registration, discovery and orchestration, avoiding coupling [9].

We can highlight that the microservices architecture helps to reduce the complexity of managing and operating the quality characteristics necessary for SaaS applications, such as: automatic scaling, automated testing, continuous integration and continuous deployment, security and fault tolerance.

In the process of developing applications based on microservices, permanent, faster and automated updates are required using DevOps practices. Applications based on microservices require a more sophisticated DevOps infrastructure, which generally requires the construction of a pipeline of continuous implementation and continuous deployment that guarantees the quality of the microservices and faster production. The use of cloud technologies, especially containers, allows the construction of said pipelines.

DevOps (Dev: Developers - Ops: Operations) is a term that emerges with the collision of two new trends, agile infrastructure or agile operations, and collaboration between development and operations personnel throughout all stages of the life cycle of development [10]. DevOps is about fast, flexible development and provisioning business processes. It efficiently integrates development, delivery, and operations, thus facilitating a lean, fluid connection of these traditionally separated silos [11].

Each microservice is developed independently of the others, the development can be done in parallel, each microservice by an independent development team, keeping in mind the dependencies between each of them, in this way the development of an application can be perform in less time. There are still research challenges at this point, J. Soldani et al, claim that having many autonomous teams that develop services deployed independently can be a double-edged sword. On the one hand, each team can make local decisions without having to negotiate with other teams and there is an increased risk that teams will not see the big picture, that is, they will understand if their local decisions are justifiable and coherent in the context of the general architecture and the commercial objectives of the application [12].

Another important point in the process of developing applications based on microservices is the identification and discovery of microservices. In a microservices application, the set of running service instances changes dynamically, including network locations. Consequently, in order for a client to make a request to a service, it must use a service-discovery mechanism. A key part of service discovery is the service registry. The service registry is a database of available service instances. The service registry provides a management API and a query API. Service instances are registered with and deregistered from the service registry using the management API, this API is responsible for ensuring that the registered microservices are available and without errors. The query API is used by system components to discover available service instance [13]. The tools that allow managing the service-discovery are: Etcd, Consul, Nerve and Synapse.

Finally, the integration and composition of the application allows, from the user stories or functional requirements of the application to be built, identify, integrate and/or adapt the microservices that will be part of the application. We do not know way to represent the user stories or the requirements of an application as a set of microservices that interact and communicate with each other to comply with the logic of the business, for this reason they are working on the proposal of a new practice agile called "service backlog" that allows this. With this practice you can model, compose and evaluate (in terms of granularity, dependencies and performance) the microservices that will be part of an application.

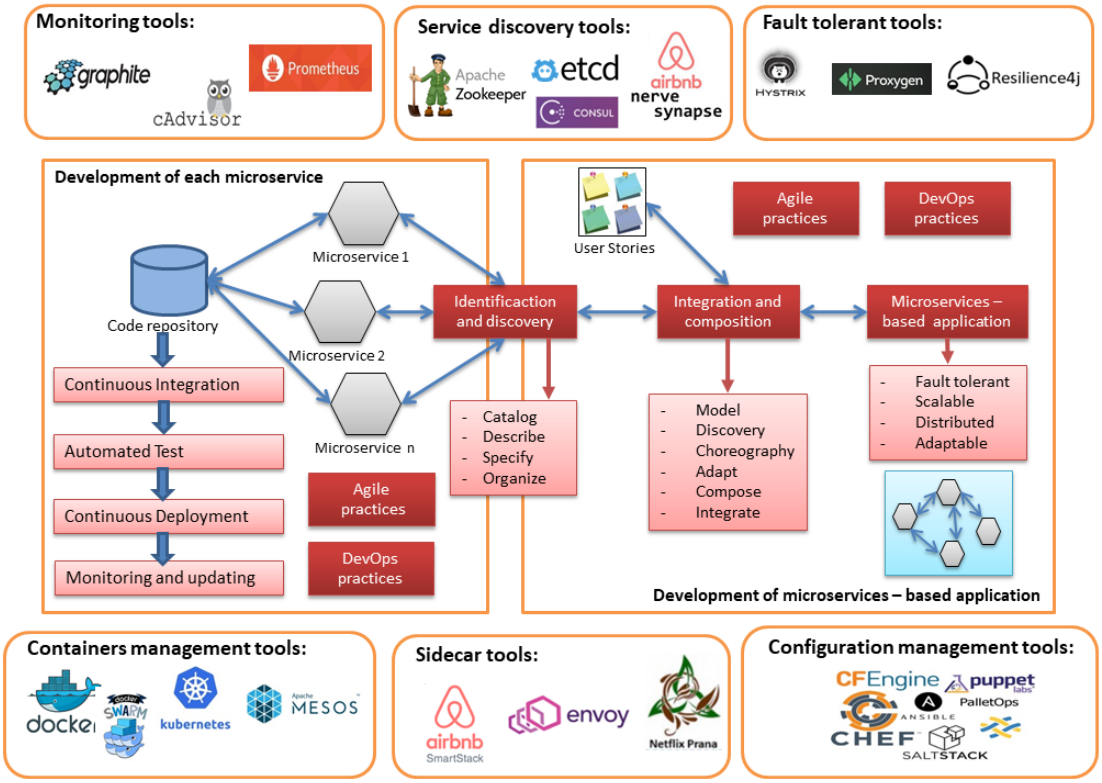

Figure 2. Main features of the development process of application with microservice [5].

4. Case Study: Develop Sinplafut a microservices – based SaaS application

Sinplafut is the information system allows improving and expediting the planning of football training using modern sports training techniques. Sinplafut can be used by football teams, clubs, physical trainers, coaches and technical directors both amateur and professional, as a software as a service application. With Sinplafut you can manage your profile, each player's card, physiological information and injuries. You can create and configure your macro-cycles, month-cycles, micro-cycles and training sessions, using pre-established methods and tests. Furthermore, it allows to register the data of competitions and you can realize analyzes and controls on each player.

The development of Sinplafut begins with the specification of the functional requirements as user stories, prioritized and detailed in a "product backlog", for the management of the development and following the Agile Scrum methodology the Taiga tool was used, Taiga is a project management platform for startups and agile developers and designers, it takes control of the progress of the development of the application.

Having identified and prioritized the functional requirements, the granularity of the microservices that will be part of Sinplafut was defined, following the principle proposed by Evans "A delimited context must be defined for each domain concept that will be exposed as a service" [14]. When the implementation began, the Sinplafut development team did not have microservices already implemented; therefore each microservice must be developed from scratch. Table 1 shows this result, detailing the microservices and user stories assigned. This assignment was made to trial and error, at the discretion of the development team, at this time there is no known or published any model, method or tool that allows us to define under some criteria the optimal granularity of microservices. This point is a subject of open research and much discussion; this is confirmed by Lewis et al, stating that the problem of granularity is the lack of agreement on the correct size of microservices. The fact that they are labeled as "microservices" shows that there is the possibility of establishing a set of patterns to help with design decisions when dividing a domain into microservices and sizing each service [15].

Table 1. Microservices – requirements relations.

User story	Microservice
1. Sport club management 2. Management of club teams 3. Management of Team's players 4. Management of coaching staff	EquipoApp
5. Management of the training plan 6. Management of the month-cycles 7. Management of the micro-cycles 8. Management of training sessions 9. Management of sports training methods 10. Generate the schedule of training sessions	PlanEntrenamientoApp
11. Web site of Sinplafut, Front-End	SinplafutApp
12. Sports test management	TestDeportivosApp
13. User management and security	SeguridadApp

The main idea is to develop each microservice individually and independently of the others, therefore it is necessary to create a code repository for version control for each microservice, in the same way a platform of continuous integration, automated testing and continuous deployment is required, continue to be managed individually for each microservice. Figure 3 shows the development platform implemented for each microservice. In Sinplafut, 5 pipelines of continuous integration, continuous deployment and automated tests were created.

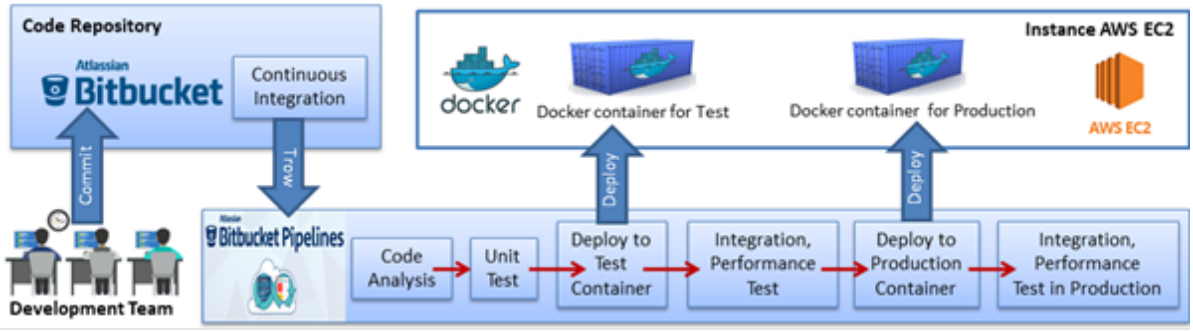

Figure 3. The development platform implemented for each microservice: With continuous integration, continuous deployment, automated test, it uses Docker and AWS EC2.

The microservices were implemented in Django (Front-End) and Rest framework. Each microservice was implemented, tested and deployed using the devops practices and a deployment pipeline in Bitbucked. Bitbucket pipeline allows writing scripts inside Docker containers where you can install and run tools to perform testing and deployment. With the continuous deployment pipeline, fast and quality deliveries are guaranteed, tested both in a testing and production environment, reducing the time to market.

Figure 4 shows the implementation architecture of each microservice. The use of sidecars can be seen with the SmartStack tool developed by Airbnb which uses the Apache Zookeeper registry together with Nerve and Synapse for service discovery and fault tolerance. Nerve is handled of verifying that each microservice is online and working correctly, in this case it registers in ZooKeeper the port and the microservice status, and otherwise it deletes it from zookeeper and marks it as not available. API gateway is the single-entry point for all clients. It is also capable of exposing different APIs to different clients. All requests from clients are first directed to an API gateway. Then the API gateway routes them to their corresponding microservice. It also performs basic request validation and response caching. [16].

The monitoring allows to maintain a centralized control of the microservices and to be able to prevent failures and falls of the application. CAdvisor in a tool implemented by Google, which allows to monitor

the EC2 instance where the containers and each one of the microservice containers are displayed. The use of CPU, Ram Memory and the available disk capacity can be displayed. Prometheus is a more specialized tool for monitoring, allows creating alerts and metrics for microservices that are part of an application. As future work, we intend to create a scorecard with different alerts and metrics with the use of Prometheus for Sinplafut.

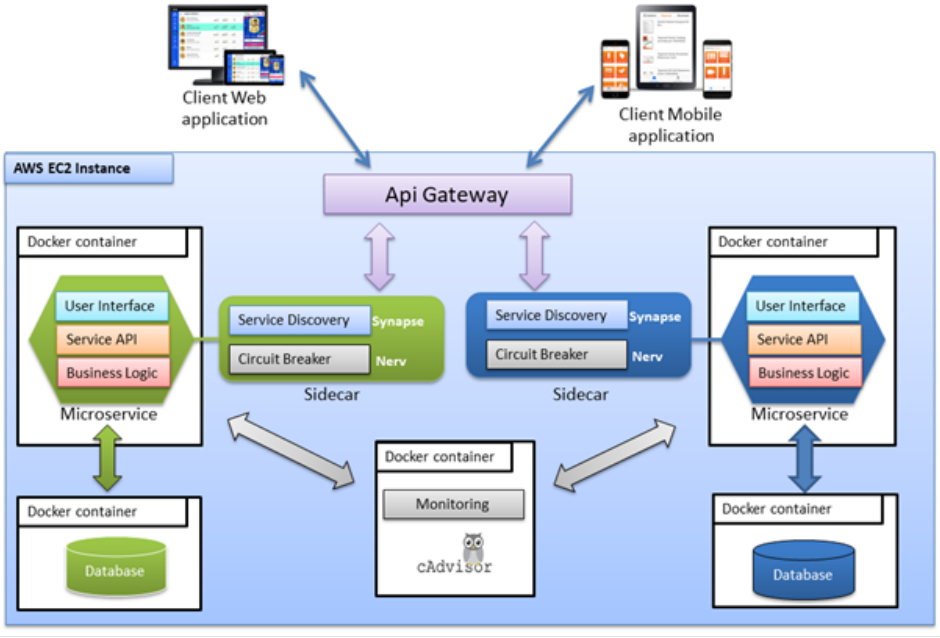

Figure 4. Microservices dependences diagram of Sinplafut. A microservice is an independent application.

5. Conclusions

In this article a case study was presented where Sinplafut is implemented a SaaS application following the development process of applications based on microservices proposed in section 3, it was observed that the definition of the process facilitates the implementation of the microservices because the developers have an initial and detailed guide of the phases, methods, tools and good practices that can be used during the construction of applications based on microservices. In order for the proposed process to be complete, we want to propose and detail design and development patterns that allow developers to speed up the construction of microservices.

The development of applications based on microservices is complex, involving the use of a large number of tools necessary for its development, deployment, monitoring and maintenance. It is required to manage a more complex and distributed infrastructure than that managed in a monolithic application. The configuration management tools are essential and can reduce the complexity of the provisioning and deployment of containers and microservices. The use of containers in development, testing and production environments allows reducing errors caused by differences in configurations, libraries, versions and dependencies, achieving the immutable server concept.

Once the entire technological platform necessary for the development of each microservice has been supplied, the benefits are amplified considerably, the time to market is reduced, and the quality of this increase is guaranteed by means of the continuous integration, the deployment pipeline, and the automated testing. The application based on microservices is tolerant to failures, if a microservice fails the entire application is not affected. The management of computational resources is done individually according to the needs of each microservice, in the same way that scaling is done according to the needs of each microservice.

References

[1] Zhang Q, Cheng L and Boutaba R 2010 Cloud computing: state-of-the-art and research challenges. *J Internet Serv Appl.* **1(1)** 7–18
[2] Gaona Cuevas CM 2017 *El modelo software como servicio* (Cali: Universidad del Valle)
[3] Villamizar M, Garcés O, Castro H, Verano M, Salamanca L, Casallas R and Gil S 2015 *10th Computing Colombian Conference (Bogotá)* (Colombia: Sociedad Colombiana de Computación and IEEE Computer Society) pp 583–590
[4] Furda A, Fidge C, Zimmermann O, Kelly W and Barros A 2018 Migrating enterprise legacy source code to microservices: on multitenancy, statefulness, and data consistency *IEEE Software* **35(3)** 63–72
[5] Vera-Rivera F H 2018 *J. Phys.: Confer. Ser.* **1126** 012017
[6] Vera-Rivera F H 2018 Método de automatización del despliegue continuo en la nube para la implementación de microservicios *Proc. XXI Conf. Iberoamericana de Ingenieria del Software CIBSE* (Bogotá) (Colombia: Universidad de los Andes)
[7] Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B and Wesslén A 2012 *Experimentation in software engineering* (New York: Springer Heidelber)
[8] Runeson P, Höst M, Rainer A and Regnell B 2012 *Case study research in software engineerign* (New Jersey: Wiley)
[9] Calçado P 2017 *Pattern: Service Mesh* consulted on: http://philcalcado.com/2017/08/03/pattern_service_mesh.html
[10] Mueller E, Wickett J, Gaekwad K and Karayanev P 2017 *What Is DevOps? The agile admin* Consulted on: https://theagileadmin.com/what-is-devops/
[11] Ebert C, Gallardo G, Hernantes J and Serrano N 2016 DevOps *IEEE Software* **33(3)** 94–100
[12] Soldani J, Tamburri D A and Van Den Heuvel W J 2018 The pains and gains of microservices: a systematic grey literature review *Journal of Systems and Software* **146** 215–232
[13] Bakshi K 2017 Microservices-based software architecture and approaches *IEEE Aerosp Conf Proc.* (Big Sky, MT) (United States of America: IEEE) pp 1-7
[14] Evans E 2004 *Domain-driven design* (Boston: Addison Wesley)
[15] Lewis J, Tilkov S, Jamshidi P, Pahl C and Mendonça N C 2018 Microservices the journey so far and challenges ahead *IEEE Software* **3(35)** 24-35
[16] Jayawardana Y, Fernando R, Jayawardena G, Weerasooriya D, Perera I 2018 A Full Stack Microservices Framework with Business Modelling *Proc. 18th Int. Conf. Adv. ICTer 2018* (Colombo) (Srilanka: IEEE) p 78-85