Towards autonomic microservices

Claudio Guidi
italianaSoftware s.r.l
cgudi@italianasoftware.com

1 Introduction

Autonomic computing is a key challenge for system engineers [1]. It promises to address issues related to system configuration and maintenance by leaving the responsibility of configuration and reparation to the components themselves.

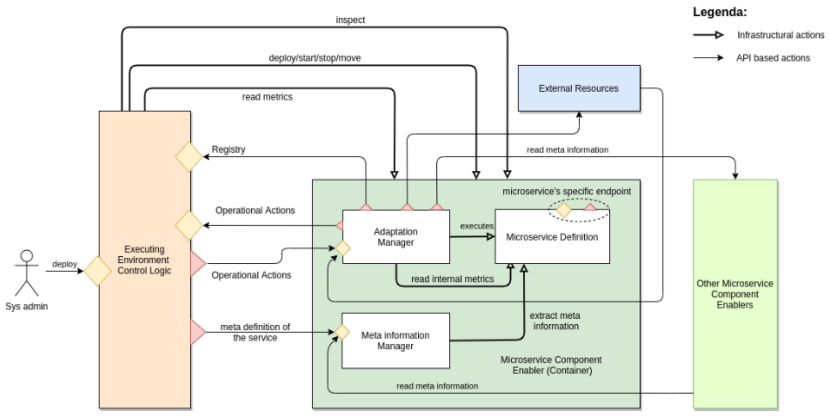
Operational activities always represent a cost in terms of required human skills and delivering time, and the minimization of their impact is a challenging task. DevOps is often used as a reference model for dealing with microservices lifecycle that allow for the automation of some steps like tests and deployment. In any case, it is not able to eliminate all the operational activities such as component configuration and runtime activities related to system maintenance, especially when an important refactoring of the system or some of its parts comes into play. In this context, at present, NoOps is a debated approach that aims at overcoming DevOps by skipping all operational activities. Such an objective is very ambitious; the hidden dream is to move all the activities necessary for the management of a component to the development phase.

Microservices are usually not equipped with any self-adaptation logic, they are never aware of the context where they are executed. Every operational activity on a microservice is always performed from an external layer able to govern the overall system. External tools like component orchestrators and monitors are fundamental for governing the complexity of a microservice based system. They allow for the setting of configuration parameters, and the retrieving of useful information like the resource consumption or the logs of the computational activities. Nevertheless, they are still poor of functionalities for dealing with system self-adaptation depending on the current load or other requirements. Some orchestrators like Kubernetes provide auto scaling functionalities for containers but they treat all the containers as *black-boxes* which can be cloned depending on the consumption of the resources without taking care about their internal logics and functionalities. Containerization technologies indeed, are agnostic w.r.t. the logics implemented by a microservices thus every time a component, or a set of components, require a main refactor there is a consequent operational impact that must be taken into account.

An autonomic computing science for microservices could facilitate the transition towards a model where operational activities are fully or mainly self-managed by the components. Such an approach require adding extra functionalities both at the level of components and at the level of the external environment. The main idea is to build a fully automatic ecosystem where microservices based applications are able to re-arrange themselves depending on the context. Some authors already initiated such an investigation [2,3], but they do not consider the idea to equip microservices with additional components for self-changing their own structure.

In this presentation, I propose a reference architecture for allowing microservices to interact with the execution environment in order to modify their own structure. A proof of concept example will be provided in Jolie[4]: an auto-splitting monolith. It is an application that is deployed as a monolith and that asks to change its architecture at runtime into a microservice based one and viceversa. The main ambition of this work is to trigger a discussion for reasoning about a standard reference architecture for autonomic microservices.

2 Reference Architecture



In the picture above an initial proposal of a reference architecture for autonomic microservice is reported. In orange, the *Executing Environment Control Logic* deals with all the functionalities that the execution environment must expose to microservices for allowing their requests of self-changement. It allows for deploying new microservices; starting or stopping a microservice; reading some executing metrics of the component like the CPU or the memory consumption; keeping a registry of all the components in execution and their dependencies; etc. In blue, the *External Resources* block just represents other sources of information for the microservices, which can be used for updating the current version of the microservice or other internal components. In green the *Microservice Component Enabler* represents the single component execution environment (ex: a container) that can be manipulated by the execution environment. It is worth noting that some components inside are required for enabling the autonomic behaviour of the microservice. The *Meta Information Manager* is in charge to expose all the meta-information of the actual microservice, such as the list of the available operations and their related signatures, or the list of the external dependencies. The *Adaptation Manger* is in charge: (i) to interact with the execution environment for requesting architectural modifications, for example the microservice could ask to deploy an internal component as an external one in order to scale it independently; (ii) to executes the actual microservice components following the self-adaptation logics programmed for that microservice; (iii) to interact with external resources, for example for retrieving new versions of the current microservice; (iv) to interact with other microservices, for example for retrieving the meta information. The *Microservice Definition* represents the definition of the microservice's components in a given technology that are internally run by the *Adaptation Manager*.

3 References

[1] Kephart, Jeffrey & Chess, D.M.. (2003). The Vision Of Autonomic Computing. Computer. 36. 41 - 50. 10.1109/MC.2003.1160055

[2] "Developing Self-Adaptive Microservice Systems: Challenges and Directions", NC Mendonça, P Jamshidi, D Garlan, C Pahl - IEEE Software, 2019

[3] "Towards service discovery and autonomic version management in self-healing microservices architecture", Yuwei Wang, Proceedings of the 13th European Conference on Software Architecture - Volume 2, September 2019 Pages 63–66

[4] "Jolie" – http://www.jolie-lang.org