Packaging Microservices in Jolie

Dan Sebastian Thrane

University of Southern Denmark

Microservices is an emerging paradigm where components (even the internal ones) are autonomous and reusable services [1]. Applications are built by composing services as black boxes, using message passing.

The nature of microservices fosters granularity, and a MicroService Architecture (MSA) typically consists of many individual services. Since services are independent and their coordination happens only through message exchanges, code reuse (the focus of this talk) takes a different form than that found in standard approaches based on software packages. Typically, in other paradigms, packages are software libraries, i.e., pieces of source or compiled code that become a part of the execution of the main application (e.g., through source inclusion, or static/dynamic linking). While this approach can be used for developing an "atomic" microservice (a service that does not contain other services), it falls short of capturing the essence of the paradigm and how it is used.

There are two key aspects that we need to keep in mind when dealing with code reuse in microservices. First, a common pattern in service development is to resolve the dependency of a service simply by *binding* it to an externally-provided service (available somewhere else in the network), instead of importing code to be run locally. Second, if we do decide to import some code to be run locally, that code should still be run as a separate and independent "local" service. This way, if we need to change strategy later on (say, when we go from development to production) and switch from running a dependency locally to binding our service to an external provider, we can do it without changing our implementation.

Package managers for mainstream technologies were not built with MSAs in mind, so these two patterns are not natively supported. Microservice developers must instead typically resort to ad-hoc conventions to deal with these problems.

In this talk, we report on the development of a module system and package manager for the Jolie programming language [2][3]. Jolie supports microservices natively, so it is a prime case study for the development of a package system for microservices.

We will demonstrate both the Jolie module system along with the Jolie Package Manager. This will include demonstrating the configuration, creation and use of Jolie modules. Furthermore, Jolie Modules support a notion of interface parametricity (polymorphism), which can be used to develop services whose behaviour is determined by the interfaces of the other services that they are bound to at deployment time. Parametricity is necessary because these interfaces are known only when packages are "linked" to each other (to solve dependencies).

This work has already been presented at DAIS 2017.

References

[1] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch-Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. In *Present And Ulterior Software Engineering (PAUSE)*. Springer, 2017. To appear. Available at https://arxiv.org/abs/1606.04036.

[2] F. Montesi, C. Guidi, and G. Zavattaro. Service-oriented programming with Jolie. In *Web Services Foundations*, pages 81–107. Springer, 2014.

[3] The Jolie Team. Jolie programming language - official website. http://jolie-lang.org/.