Beyond auto-scaling: application-aware optimal elasticity

Jacopo Mauro[1], Iacopo Talevi[2], and Gianluigi Zavattaro[2]

[1] Southern Denmark University, Denmark
mauro@sdu.dk
[2] FOCUS Research Team, University of Bologna/INRIA, Italy
iacopo.talevi@studio.unibo.it gianluigi.zavattaro@unibo.it

Microservices recently gained popularity as an architectural style that, inspired by service-oriented computing, structures software applications as compositions of fine-grained and loosely-coupled services [4]. This approach is intended to increase the modularity and scalability of applications. For this reason, modern software engineering practices, like continuous delivery/deployment [9] have microservices at their base.

Current state-of-the-art deployment technologies, e.g., container-based solutions like Kubernetes [8], are based on the specification of two kinds of information: the dependencies among the microservices composing the architecture to be deployed and autoscaling metrics. In particular, autoscaling supports the independent increase or decrease of microservice instances, based on the values of monitored metrics (CPU average load, response time, ...). Autoscaling independence is strongly grounded on the principle of loose-coupling at the base of microservices. However, it does not take advantage of the information about the interdependencies among the microservices in the same architecture. For instance, if we detect a peak of inbound requests on the microservice that works as entry point of a pipeline of sequentially-interdependent services, it would be more efficient to immediately scale all the microservices in the pipeline, instead of letting each microservice successively autoscale. Similarly, by knowing that more than one microservice needs to horizontally scale, it is possible to compute optimal deployment strategies where instances of different microservices share resources. For example, replicas of different microservices can be installed in bulk on a new computing node added to the system. An optimisation impossible to achieve by solely relying on horizontal autoscaling.

In this extended abstract, we advocate and present on-going work [1] for an alternative strategy for deployment and scaling. The idea is to compute reconfiguration plans that, reasoning at the global architectural level, instead of the local microservice level, add several instances of different services at once, reaching optimal placement of such instances.

The first step towards the realization of such a strategy is the definition of an appropriate model for the representation of microservice interdependencies. To this aim, we take inspiration from the *Aeolus component model* [3], which was used to formally define the problem of deploying component-based software systems and to prove that, in the general case, such problem is undecidable [3]. The basic idea of Aeolus is to enrich the specification of components with a finite state automaton that describes their deployment life-cycle. Differently from Aeolus components, the specification of the deployment life-cycle of microservices does not have the full power of finite state automata (like in Aeolus and other TOSCA-compliant deployment models [2]), and instead have only two states: (i) creation and (ii) binding/unbinding.

To address the optimality of resource consumption, we are interested in modeling also resource/cost-aware deployments. To this aim, we assume microservices to be enriched with the specification of the amount of resources they need to run. In a deployment, a system of microservices runs within a set of computation *nodes*. In our model, nodes represent computational units (e.g., virtual machines in an Infrastructure-as-a-Service Cloud deployment). Each node has a cost and a set of resources available to the microservices it hosts. The *optimal deployment problem* is therefore defined as follows: given an initial microservice system, a set

of available nodes, and a target goal of microservices to be deployed, find a sequence of reconfiguration actions that, once applied to the initial system, leads to a new deployment that satisfies the target goal. The *optimal deployment* has two properties: i) each used node has at least as many resources as those needed by the hosted microservices; ii) the total cost (i.e., the sum of the costs) of the used nodes is minimal.

We demonstrated that the optimal deployment problem for microservices is decidable. This is proved by presenting an algorithm that i) generate a set of constraints whose solution indicates the microservices to be deployed and their distribution over available nodes; ii) generate another set of constraints whose solution indicates the connections to be established; iii) synthesize the corresponding deployment plan. The generated set of constraints are enriched with optimization metrics that minimize the overall cost of the computed deployment.

The algorithm has NEXPTIME complexity because, in the worst-case, the length of the deployment plan could be exponential in the size of the input. However, in practice, it is reasonable to assume that each node can host at most a polynomial amount of microservices as a consequence of its resource limitations. In this case, the optimal deployment problem is an NP-optimization problem that can be solved by using state-of-the-art constraint solvers [6, 7].

One of the main challenges in our approach for optimal deployment is concerned with the possibility in practice to compute the optimal plan. We have concretely evaluated our approach, by considering a real-world microservice architecture, inspired by the reference email processing pipeline from Iron.io [5]. We modeled that architecture in the Abstract Behavioral Specification (ABS) language, a high-level object-oriented language that supports deployment modeling [10]. We used our technique to compute two types of deployments: an initial one, with one instance for each microservice, and a set of deployments to horizontally scale the system depending on small, medium or large increments in the number of inbound requests. Our experimental results are encouraging: we can compute deployment plans that add more than 30 new microservice instances, assuming availability of hundreds of machines, and guaranteeing optimality.

References

[1] Mario Bravetti, Saverio Giallorenzo, Jacopo Mauro, Iacopo Talevi, and Gianluigi Zavattaro. Optimal and Automated Deployment for Microservices. In *FASE*, 2019.

[2] Antonio Brogi, Andrea Canciani, and Jacopo Soldani. Modelling and analysing cloud application management. In *ESOCC*, volume 9306 of *LNCS*, pages 19–33. Springer, 2015.

[3] Roberto Di Cosmo, Jacopo Mauro, Stefano Zacchiroli, and Gianluigi Zavattaro. Aeolus: A component model for the cloud. *Inf. Comput.*, 239:100–121, 2014.

[4] 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.*, pages 195–216. Springer, 2017.

[5] Ken Fromm. Thinking Serverless! How New Approaches Address Modern Data Processing Needs. https://bit.ly/2DtNFQN.

[6] GECODE. An open, free, efficient constraint solving toolkit. http://www.gecode.org.

[7] Google. Optimization tools. https://developers.google.com/optimization/.

[8] Kelsey Hightower, Brendan Burns, and Joe Beda. *Kubernetes: Up and Running Dive into the Future of Infrastructure*. O'Reilly Media, Inc., 1st edition, 2017.

[9] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.

[10] Einar Broch Johnsen, Reiner Hähnle, Jan Schäfer, Rudolf Schlatte, and Martin Steffen. ABS: A Core Language for Abstract Behavioral Specification. In *FMCO*, 2010.