Performance Engineering for Microservices and Serverless Applications: The RADON Approach

Alim U. Gias,[1] André van Hoorn,[2] Lulai Zhu,[1]
Giuliano Casale,[1] Thomas F. Düllmann,[2] Michael Wurster[2]
[1]Imperial College London, UK
[2]University of Stuttgart, Germany

ABSTRACT

Microservices and serverless functions are becoming integral parts of modern cloud-based applications. Tailored performance engineering is needed for assuring that the applications meet their requirements for quality attributes such as timeliness, resource efficiency, and elasticity. A novel DevOps-based framework for developing microservices and serverless applications is being developed in the RADON project. RADON contributes to performance engineering by including novel approaches for modeling, deployment optimization, testing, and runtime management. This paper summarizes the contents of our tutorial presented at the 11th ACM/SPEC International Conference on Performance Engineering (ICPE).

ACM Reference Format:
Alim U. Gias, André van Hoorn, Lulai Zhu, Giuliano Casale, Thomas F. Düllmann, Michael Wurster. 2020. Performance Engineering for Microservices and Serverless Applications: The RADON Approach. In *ACM/SPEC International Conference on Performance Engineering Companion (ICPE '20 Companion), April 20–24, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3375555.3383120

1 INTRODUCTION

Recent trends in cloud software technologies, such as microservices and serverless functions, are rapidly changing the way companies produce complex software architectures. When choosing to deliver business logic through serverless functions, microservices, or monoliths, end-users will experience different trade-offs in terms of non-functional properties such as performance, reliability, and cost. The assessment of these trade-offs calls for the development of specialized modeling, testing, and management techniques that can take into account such differences.

To aid in this scenario, the RADON [1] project aims to develop an advanced DevOps framework focusing on microservices and serverless Function-as-a-Service (FaaS).[1] A major challenge addressed by the RADON project is quality assurance, particularly considering non-functional properties like performance. Performance evaluation is always a challenging aspect in software development, and additional challenges arise for both microservices [8] and FaaS [18]. The objective of the tutorial is to demonstrate these challenges and how they are addressed in different contributions in the RADON context.

The tutorial will initially introduce the microservices and serverless FaaS application architecture, and performance engineering challenges in this context. The focus will then shift to how these applications can be modeled using the Topology and Orchestration Specification for Cloud Applications (TOSCA) [12] and how different performance specifications can be integrated into such models. After that, the tutorial will discuss how such models are utilized in RADON to determine optimal decomposition strategies of serverless functions. Once the application has been deployed, following the DevOps practice, it is necessary to generate and maintain performance test cases for continuous integration and deployment. Finally, the tutorial focuses on runtime resource management by demonstrating a fine-grained autoscaling approach.

This paper serves as a summary of the contents presented in the tutorial, including the references for further details. The mentioned tools are available online [2], including more comprehensive technical documentation and examples of use. The artifacts for the tutorial, including slides and examples, are also provided online [6].

2 MODELING

Modeling in RADON builds on the OASIS TOSCA standard [12]. TOSCA allows to model so-called service templates (or blueprints) that describe the topology — i.e., components and their relationships — of an application to be deployed to a cloud infrastructure. The core modeling concepts in TOSCA are nodes, relationships, and policies, for which respective types are defined. Being based on a YAML-based format, TOSCA models can, in principle, be edited

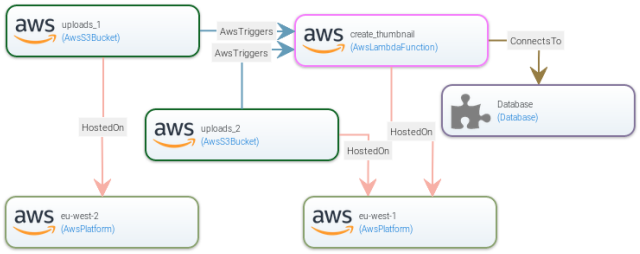

Figure 1: TOSCA model with RADON extensions in Winery

[1]http://radon-h2020.eu/

ICPE '20 Companion, April 20–24, 2020, Edmonton, AB, Canada

ACM ISBN 978-1-4503-7109-4/20/04.
https://doi.org/10.1145/3375555.3383120

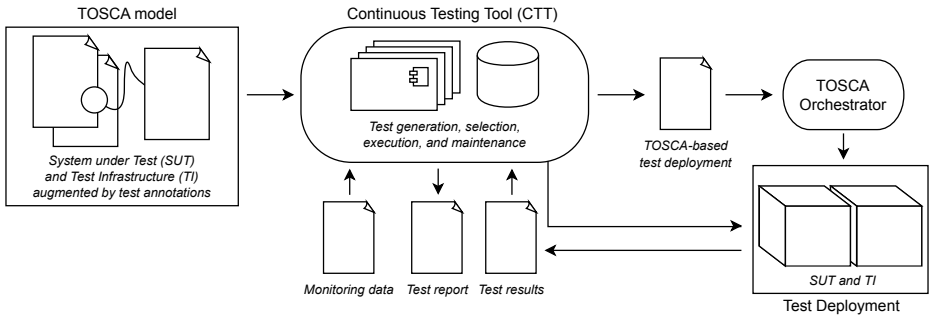

Figure 2: Workflow of the CTT tool

in any text editor. However, with Winery,[2] a web-based graphical modeling environment for more convenient modeling of TOSCA topologies and plans to manage these topologies is available. A TOSCA-compliant orchestrator takes care of the deployment and management of the application topology. One such orchestrator is xOpera.[3] Both Winery and xOpera are used and extended as part of the RADON toolchain.

In RADON, we extend the set of TOSCA types to allow TOSCA-based modeling and deployment of microservices and serverless applications. For instance, we have added new types for representing microservices and FaaS components that can be deployed to state-of-the-art proprietary and open-source infrastructures such as AWS, Google, Azure, and Apache OpenWhisk. Our extensions are available as part of RADON's *particles* repository [2]. Figure 1 shows a TOSCA topology for a FaaS-based application modeled in Winery.

3 DEPLOYMENT OPTIMIZATION

FaaS compute services, for example, AWS Lambda,[4] automatically scale a function on demand at runtime. The function owner is charged only for resources and time spent during invocations to the function. These characteristics make existing deployment optimization approaches not applicable in the context of serverless FaaS. To fill this gap, RADON introduces a novel model-driven approach to finding the optimal deployment scheme for a FaaS-based application, which minimizes the operating costs while satisfying the performance requirements on the target cloud platform.

Although the runtime resource management of serverless functions is opaque from developers, one can control the autoscaling of a function by adjusting its memory and concurrency. The former determines the amount of resources available for each function instance, while the latter specifies the maximum number of instances that the function can have. These two parameters are both critical to the operating costs and performance of a FaaS-based application. The goal of the deployment optimization program is, therefore, to find the optimal memory and concurrency configuration such that the total operation cost of the application is minimal under the performance requirements. A FaaS-based application typically consists of serverless functions, object storages, and other types of nodes, whose behavior can normally be well captured by appropriate structures in Layered Queueing Networks (LQNs) [4]. It thus suffices to predict the performance of a FaaS-based application using an LQN. However, creating a benchmark of the application for parameterizing the LQN is sometimes difficult due to the lack of ways to obtain accurate measurements of certain nodes.

With the extended definitions of TOSCA node, relationship, and policy types, a RADON user can build an LQN to model the behavior of their application and specify the performance requirements in a TOSCA service template. The aforementioned approach has been implemented by the RADON decomposition tool to consume such a TOSCA service template for deployment optimization. As part of the RADON toolchain, this tool is currently available on a public access server with a RESTful API [2].

4 CONTINUOUS TESTING

The most common approach for quality assurance in practice is testing. Hence, testing of functional and non-functional properties must be part of any DevOps-based process and infrastructure. RADON defines a continuous testing workflow that comprises the definition, execution, and maintenance of functional and non-functional tests.

A RADON user defines tests by adding them to the TOSCA service template for the application under test. We have extended the set of TOSCA node types, relationship types, and policy types for expressing different types of tests and including suitable test drivers. For instance, this allows the definition of a load test to be executed using a configured load driver such as JMeter. After being deployed by a TOSCA orchestrator, the tests are executed and the test results are made available to the RADON user.

Especially the testing of non-functional properties such as performance imposes several challenges in the context of DevOps, microservices, and serverless. Example challenges include *(i)* frequent changes of the application and its operational profile that require frequent adoptions and executions of the tests, *(ii)* the desire for fast deployment of changes that conflicts with the time-consuming nature of testing, as well as the *(iii)* cloud-based deployment infrastructures that make it difficult to get reliable and repeatable test results.

We address these challenges by *(i)* automatically extracting and evolving performance tests using operational monitoring data and API information [16, 19], *(ii)* the generation and selection of tailored

[2] http://eclipse.org/winery/
[3] https://github.com/xlab-si/xopera-opera
[4] https://aws.amazon.com/lambda/

tests based on current test concerns [14, 15], as well as *(iii)* exploiting recommending test strategies suitable for testing in unreliable infrastructures [3, 11].

The aforementioned approach is implemented by the Continuous Testing Tool (CTT), as a part of the RADON toolchain. Figure 2 shows the workflow of the CTT tool. Via its REST-based interface, RADON users can execute the continuous testing on-demand or include it as a part of the CI/CD process. CTT is designed as an extensible framework that allows the definition of new test types, metrics, and tools. CTT is publicly available under an open-source license [2]. CTT will integrate and extend parts of the ContinuITy approach and tools for performance testing in continuous software engineering.[5]

5 RUNTIME MANAGEMENT

A runtime resource manager for microservices cannot be governed by static rules only, like always scaling a microservice either horizontally or vertically. Although these rule-based practices are common in the industry [13], recently it has been emphasized by researchers that such resource allocations should be based on the system architecture and current workload [5, 10]. This can be achieved by a model-based resource manager that can ensure optimal resource allocation by analyzing multiple performance models for the current workload. Designing such a resource manager involves two major challenges — finding the appropriate modeling scheme and quickly analyzing the model at runtime.

The first issue can be resolved with a Layered Queueing Network (LQN) model, which can accurately represent different aspects of microservices like its fractional CPU share and working as a server and client simultaneously. The LQN models can be generated automatically using architectural to performance model transformation tools [7]. If such architectural models are not available, an LQN can be constructed by monitoring the communications among the microservices. An important parameter for an LQN model is the service demand of the microservices. These demands can be estimated using state-of-the-art techniques based on utilization [17] or response time [9].

To address the second issue, a meta-heuristic like genetic algorithm can be used that will suggest the optimal resource allocation scheme for a specific workload without exhaustively searching the sample space of different configurations. Considering that the CPU is the bottleneck, the configurations include the CPU capacity and the number of replicas of each microservice. Such configurations can be evaluated using the estimates from the LQN model, and the optimal configuration can be determined. If the resource manager follows a conservative strategy, a new configuration is only executed if the potential performance gain is significant. In addition, since the decisions need to be provided at runtime, the meta-heuristic algorithm can be time-bounded.

The overall methodology is presented in Figure 3. The model generator and the resource manager addresses the two discussed issues. The system monitor complements both components by providing necessary information like different system metrics and workload patterns.

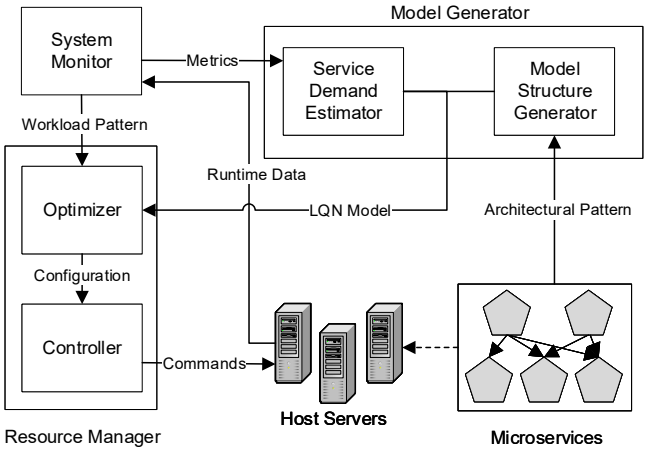

Figure 3: The methodology for runtime resource management of microservices

6 CONCLUSION

This paper outlined RADON's approach to performance engineering for microservices and serverless applications, which is comprised of TOSCA-based modeling, deployment optimization, continuous testing, and runtime management.

ACKNOWLEDGMENTS

This work is being supported by the European Union's Horizon 2020 research and innovation programme (grant no. 825040, RADON). Parts of this work are being conducted in collaboration with the SPEC RG DevOps Performance Working Group.[6]

REFERENCES

[1] Giuliano Casale, Matej Artač, Willem-Jan van den Heuvel, André van Hoorn, Pelle Jakovits, Frank Leymann, Mike Long, Vasilis Papanikolaou, Domenico Presenza, Alessandra Russo, Satish N. Srirama, Damian A. Tamburri, Michael Wurster, and Lulai Zhu. 2019. RADON: Rational Decomposition and Orchestration for Serverless Computing. *Software-Intensive Cyber-Physical Systems* (2019).

[2] RADON consortium. 2020. RADON Tools. https://github.com/radon-h2020/

[3] Simon Eismann, Cor-Paul Bezemer, Weiyi Shang, André van Hoorn, and Dušan Okanović. 2020. Microservices: A Performance Tester's Dream or Nightmare?. In *Proceedings of the 11th ACM/SPEC International Conference on Performance Engineering (ICPE 2020)*.

[4] Greg Franks, Tariq Al-Omari, Murray Woodside, Olivia Das, and Salem Derisavi. 2008. Enhanced modeling and solution of Layered Queueing Networks. *IEEE Transactions on Software Engineering* 35, 2 (2008), 148–161.

[5] Alim Ul Gias, Giuliano Casale, and Murray Woodside. 2019. ATOM: Model-Driven Autoscaling for Microservices. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS 2019)*. IEEE, 1994–2004.

[6] Alim U. Gias, André van Hoorn, Lulai Zhu, Giuliano Casale, Thomas Frederik Düllmann, and Michael Wurster. 2020. Performance Engineering for Microservices and Serverless Applications: The RADON Approach — Artifacts. https://dx.doi.org/10.5281/zenodo.3670013

[7] Gordon P Gu and Dorina C Petriu. 2005. From UML to LQN by XML algebra-based model transformations. In *Proceedings of the International Workshop on Software and Performance (WOSP 2005)*. 99–110.

[8] Robert Heinrich, André van Hoorn, Holger Knoche, Fei Li, Lucy Ellen Lwakatare, Claus Pahl, Stefan Schulte, and Johannes Wettinger. 2017. Performance engineering for microservices: research challenges and directions. In *Companion of the ACM/SPEC on International Conference on Performance Engineering (ICPE 2017)*. 223–226.

[5] https://continuity-project.github.io/

[6] https://research.spec.org/devopswg

[9] Stephan Kraft, Sergio Pacheco-Sanchez, Giuliano Casale, and Stephen Dawson. 2009. Estimating service resource consumption from response time measurements. In *Proceedings of the International ICST Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2009)*. 1–10.

[10] Anthony Kwan, Jonathon Wong, Hans-Arno Jacobsen, and Vinod Muthusamy. 2019. HyScale: Hybrid and Network Scaling of Dockerized Microservices in Cloud Data Centres. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS 2019)*. IEEE, 80–90.

[11] Dusan Okanovic, Samuel Beck, Lasse Merz, Christoph Zorn, Leonel Merino, Andre van Hoorn, and Fabian Beck. 2020. Can a Chatbot Support Software Engineers with Load Testing? Approach and Experiences. In *Proceedings of the 11th ACM/SPEC International Conference on Performance Engineering (ICPE 2020)*.

[12] Organization for the Advancement of Structured Information Standards (OASIS). 2019. TOSCA Simple Profile in YAML Version 1.3. http://docs.oasis-open.org/tosca/

[13] Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. 2018. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–33.

[14] Henning Schulz, Tobias Angerstein, Dušan Okanović, and André van Hoorn. 2019. Microservice-tailored Generation of Session-based Workload Models for Representative Load Testing. In *Proceedings of the 27th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2019)*.

[15] Henning Schulz, Dusan Okanovic, André van Hoorn, Vincenzo Ferme, and Cesare Pautasso. 2019. Behavior-driven Load Testing Using Contextual Knowledge—Approach and Experiences. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE 2019)*. ACM, 265–272.

[16] Henning Schulz, André van Hoorn, and Alexander Wert. 2020. Reducing the Maintenance Effort for Parameterization of Representative Load Tests Using Annotations. *Software Testing, Verification & Reliability* 30, 1 (2020).

[17] Simon Spinner, Giuliano Casale, Fabian Brosig, and Samuel Kounev. 2015. Evaluating approaches to resource demand estimation. *Performance Evaluation* 92 (2015), 51–71.

[18] Erwin Van Eyk, Alexandru Iosup, Simon Seif, and Markus Thömmes. 2017. The SPEC cloud group's research vision on FaaS and serverless architectures. In *Proceedings of the International Workshop on Serverless Computing (WoSC 2017)*. 1–4.

[19] Christian Vögele, André van Hoorn, Eike Schulz, Wilhelm Hasselbring, and Helmut Krcmar. 2018. WESSBAS: Extraction of Probabilistic Workload Specifications for Load Testing and Performance Prediction—A Model-Driven Approach for Session-Based Application Systems. *Journal on Software and System Modeling (SoSyM)* 17, 2 (2018), 443–477.