RADON: Rational Decomposition and Orchestration for Serverless Computing

G. Casale[1], M. Artač[2], W.-J. van den Heuvel[3], A. van Hoorn[4], P. Jakovits[5], F. Leymann[4], M. Long[6],
V. Papanikolaou[7], D. Presenza[8], A. Russo[1], S. N. Srirama[5], D. A. Tamburri[3], M. Wurster[4], L. Zhu[1]
1: Imperial College London, UK; 2: XLAB, Slovenia; 3: Jheronimus Academy of Data Science, the Netherlands;
4: Universität Stuttgart, Germany; 5: Tartu Ülikool, Estonia; 6: Praqma, Norway;
7: Athens Technology Center, Greece; 8: Engineering Ingegneria Informatica, Italy.

***Abstract*—Emerging serverless computing technologies, such as function-as-a-service (FaaS) offerings, enable developers to virtualize the internal logic of an application, simplifying management of cloud native applications and allowing cost savings through billing and scaling at the level of individual function calls. Serverless computing is therefore rapidly shifting the attention of software vendors to the problem of developing cloud applications that can use these platforms.**

The RADON research agenda centers around creating a model-driven DevOps framework to create and manage microservices-based applications that can optimally exploit serverless computing technologies. RADON applications will include fine-grained and independently deployable microservices that can efficiently exploit FaaS and container technologies. The methodology strives to tackle complexity, harmonize the abstraction and actuation of event processing chains, avoid FaaS lock-in, and optimize decomposition and reuse through models.

***Keywords*-Function as a Service, Serverless Computing, DevOps, Software Models**

I. Introduction

The success of IT-driven companies like Amazon, Google and Tesla, has shown that businesses investing in advanced software technologies can build a strong lead across a variety of market domains. A recent development with the potential to radically evolve the software technology landscape is serverless computing, a cloud-computing execution model proposed to entirely bypass user involvement in managing compute resources [1]. Function as a service (FaaS) is as of today the most prominent example of serverless technology. Using FaaS, individual function calls can be served remotely and automatically from the cloud [2]. This powerful idea brings us at a crossroads where current cloud software architectures are changing rapidly, increasing in flexibility, and adaptability, calling for new research in the area.

To better understand the advantages of FaaS and the implied research challenges, it is important to realize that the sudden popularity of platforms such as AWS Lambda [3], Google Cloud Functions [4] and Microsoft Azure Functions [5], may be attributed to the following two main promises of the serverless FaaS paradigm, among other benefits:

1) Fine-grained auto-scaling. This refers to the ability to virtualize the internal logic of cloud-native applications, so that individual function calls are served remotely from the cloud and can thus harness autoscaling upon demand. That is, serverless FaaS adds compute resources only to the portion of code that consumes them, simplifying scaling and saving costs compared to container as a service (CaaS) or other offerings. Transparent and fine-grained scaling is particularly efficient for event-centric systems, as for example in the Internet of Things (IoT), where actions need to promptly take effect in response to events triggered by software, data or the physical world.

2) Productivity gains. These arise from storing IT know-how as reusable serverless functions, which can be automatically orchestrated onto either private or public cloud infrastructures. Contrary to web services, serverless platforms automatically take care of the underneath operations in a portable and reproducible way. Thus, every organization can be digitally transformed through investment in archiving core functions to be later used as building blocks for IT-driven business processes. Further to this idea, any function may possibly be downloadable from public marketplaces or repositories. Such a development can considerably accelerate the delivery of new offerings to the market, even for companies with limited IT skills and no operations teams.

Despite the above promises, today there is still a lack of methodologies to reap the benefits of serverless FaaS in industry. Our vision is that open-source tools can greatly assist companies, especially SMEs, in building proof-of-concept and business cases with serverless FaaS at low costs, in order for them to responsively and progressively acquire technical capacity needed to incorporate this innovation in existing and new products. From an engineering point of view, a DevOps framework integrated with holistic management solutions that cover the entire life cycle of FaaS-based applications is still an open research challenge that deserves attention from the software engineering community.

The RADON research agenda proposes to develop an advanced DevOps framework to prototype, design, deploy, test, verify and evolve complex applications building on (i) serverless FaaS, defining events, triggers and actions (handling functions); (ii) services of various granularities, typically microservices, implementing business logic; (iii) data pipelines, i.e., combination of specialized microservices, resources and communication mechanisms that manage the life cycle of data (ingestion, filtering, transformation, analysis,

transfer and storage). The framework we aim for would allow support for a continuous spectrum of service granularities, a fine-grained control of autoscaling at the function level, and introduce the ability to deploy on infrastructures with heterogeneous capacities and at different locations mediated by models.

II. Major Research Challenges

In this section, we identify and discuss the major challenges in attaining the objective of RADON, highlighting research and innovation problems that need to be solved.

A. FaaS-Oriented Modeling

Each element in the microservices architecture constitutes a single, independent, functional and self-managing building block, e.g., a business logic module [6]. Serverless computing, particularly its instantiation into FaaS, is realized as a suitable programming paradigm for microservices, making the management of computing resources completely transparent and enabling the services to be designed in an event-centric manner [7]. Specific languages are emerging to natively program architectural design of applications composed of microservices, e.g., Jolie [8]. Such languages are a positive development but also pose new challenges, e.g., they can rely on custom interpreters that potentially carry a risk of vendor lock-in. Moreover, they do not support the specification of dependencies at the function level nor the definition of data pipelines needed for business logic, where events are essentially triggered by data. To counteract vendor lock-in, we argue that a model-driven approach can be combined with DevOps to deliver FaaS-based application. One major challenge identified by the RADON research agenda is then to propose a *FaaS-oriented modeling language* with novel constructs for specifying function dependencies and defining data pipelines carrying the events that trigger the function calls.

B. Requirements Formalization

Common features of modeling languages include abstract syntax, i.e., the definition of concepts and how they relate to each other, and semantics, i.e., machine-interpretable mappings from abstract concepts to concrete elements in a given domain. The semantics of cloud modeling languages often reflects English prose [9]. This results in a shortage of formal solutions for synthesizing requirements across different layers of cloud applications, creating a gap with respect to advanced tools for automated synthesis of requirements arising from business goals and user scenarios outside the cloud context [10], [11]. DevOps and continuous delivery tools also call for extension to existing formalization methods to simplify decomposition, synthesis and refinement of requirements up to the orchestration layer. In particular, there is a strong expectation for requirement formalisms that, while remaining close to concepts and abstractions typical of human thinking, they are also easy to translate into standard models for orchestration. To address these challenges, RADON will introduce a constraint definition language (CDL) for formally specifying both functional and non-functional requirements, increasing automation in designing FaaS-based applications.

C. Continuous Integration and Continuous Delivery

Continuous integration and continuous delivery (CI/CD) are concepts that underlie agile approaches for software development and release [12], [13]. CI/CD define full pipelines that involve compiling, functional testing, packaging, artifact generation and storage, and intalling test environments as well as delivering complete artifacts. At the bottom level, they include configuration management tools, such as Chef [14], Ansible [15] and Puppet [16], that use infrastructure-as-code (IaC) languages to install and configure application components on infrastructures. Operating at the next level are orchestrators that take the deployment blueprint of a whole application, provision compute resources needed for running the application and invoke configuration management tools to populate the resources. OpenTOSCA [17], Cloudify [18] and Apache ARIA TOSCA [19] are examples of orchestrators that consume variations of OASIS TOSCA [20], while Apache Brooklyn [21] and Juju [22] are based on other languages. At the top level are servers like Hudson [23] and Jenkins [24], which are open-source, and commercial ones like TeamCity [25], Bamboo [26], Travis CI [27] and CircleCI [28]. These offerings have in common that users can set up CI or CD jobs to execute a given script upon certain events. Within the RADON agenda, the challenge in this aspect is to extend existing solutions with capabilities to orchestrate serverless functions, microservices and data pipelines in a native, reusable and portable fashion.

D. Quality Assurance

Tools for functional testing are already an integral part of today's CD pipelines, e.g., automated unit and integration tests. However, testing non-functional properties, e.g., performance, has always been difficult [29], [30] and additional challenges arise in the context of microservices and CD [31]. An often-cited challenge is the trade-off between rapid delivery and extensive testing. One promising approach for this is to use monitoring data recorded in production to automatically create, select and evolve test cases to be executed through CD pipelines [32]. Among existing efforts, ITEA3 TESTOMAT [33] focuses on testing automation in agile development, but does not cover the microservices architecture. ContinuITy [34] aims to improve the quality of load testing for microservices by evolving test specifications using monitoring data. DICE QT [35] automates load testing of data pipelines by generating workloads that are stochastically similar to input traces. A set of quality

assurance tools is envisioned by RADON as a way for seeking latent defects in FaaS-based solutions and verifying their satisfaction of service-level agreements (SLAs) and other specified requirements.

As in the DevOps quality assurance paradigm [36], IaC enables developers and operators to jointly create, configure and manage infrastructures by means of executable code (e.g., using a combination of the TOSCA language and configuration management tools like Chef, Ansible and Puppet). At present, the quality of IaC code is still largely dependent on the experience of developers and operators. A great deal of research work has been devoted in traditional software engineering to devising quality measures for technical debt of code in recognition of for example bad smells [37], [38], which are suboptimal design decisions made by developers that can affect the overall maintainability of a software system and make it more defect-prone [39]. However, none of technical debt management techniques, such as bad smell detection or defect prediction, has been applied to IaC languages. Within RADON we see that this gap needs also to be addressed by DevOps quality assurance tools, broadening the spectrum of research challenges of the agenda.

III. A Proposal for an Integrated Framework

Based on the previous analysis of the state-of-the-art, we now propose a vision and research agenda for a framework that tackles the identified challenges. The authors of this paper are involved in a research initiative, the RADON project (http://radon-h2020.eu, 2019-2021, funded by Horizon 2020) that will concretely deliver on this vision and release the results as open source software.

The RADON framework is envisioned to comprise three environments, the modeling environment, the runtime environment and the coding environment (IDE). It will be coupled with a DevOps methodology to coordinate the development and release of FaaS-based applications, and with quality assurance tools to validate a particular solution. Figure 1 illustrates the architecture of the framework.

A. Modeling Environment

RADON will rely on a model-driven approach for managing cloud applications that typically apply the serviceless FaaS paradigm and the microservices architecture. We see OASIS TOSCA as being in the best position to act as the baseline modeling language to describe the topology and orchestration of such applications, on top of which we will define a novel family of RADON models. At present, no native support is provided by the TOSCA language for serverless FaaS or data flows, which however are both critical to RADON's significance due to the fact that serverless functions are often used to handle events triggered by data (e.g., real-time streams and diagnostic logs). However, there is already preliminary work available focusing on the

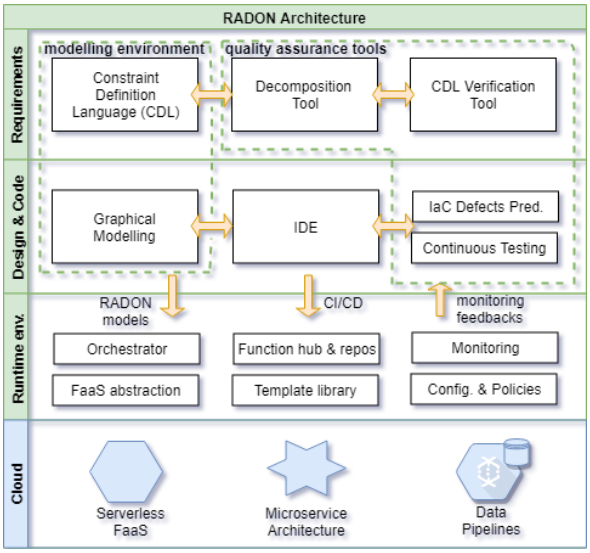

Figure 1. The architecture of the RADON framework.

modeling and automated deployment of FaaS-based applications using TOSCA [40], which will be used as a basis for our further research in RADON. One conceptual difference between RADON and TOSCA models lies in incorporating the behavior specification of FaaS-based applications. For example, RADON tends to simplify the description of a temporal sequence of actions triggered by a certain chain of events via graphical annotations, while in cases where the graphical complexity may be a limiting factor, users will have the possibility to annotate temporal predicates through the CDL. This is an intuitive logic-based language that will be proposed for specifying temporal behavior and formal requirements (for performance, cost, security and privacy) at different development stages of the application. The CDL will enable users to relate entities as well as events in RADON models, including those pertaining to data that transit on a pipeline. With CDL annotations, RADON models will retain two core advantages of TOSCA: (i) graphical and textual modeling, thanks to the Eclipse Winery project [41] and the TOSCA YAML specification [42]; (ii) automated orchestration of the application at runtime.

B. Runtime Environment

The RADON runtime will package tools that bridge the modeling environment and IDE of the framework with the cloud, relying on model-driven orchestration and IaC to enact the deployment of FaaS-based applications and data pipelines on different target cloud platforms. Particularly, development in the absence of an operations team will also be covered by this methodology. As such, the development team will be able to manage the runtime life cycle of the application in a self-service fashion. Model-to-text transformation between RADON models and TOSCA YAML

files will be enabled by Eclipse Winery to directly feed the RADON orchestrator.

The runtime environment will feature a template library that encapsulates at the proper levels of abstraction all the necessary elements of the microservices architecture. This will extend the DICE TOSCA library [43], a baseline that encodes many TOSCA templates for Big Data processing. For FaaS-based applications, the extension includes the node representations of individual functions and relationship representations that express dependencies as well as annotations for events foreseen to trigger a function. Each template in effect defines a directed acyclic graph (DAG) of the application components, inclusive of those pertaining to serverless computation. Purpose-built microservices, called event gateways, will serve as mediators between event producers and consumers across target FaaS platforms, realizing an abstraction layer underlying the application. Developers will be able to implement functions at their discretion and store the implementations in a function hub provisioned within both RADON runtime and graphical modeling tool, allowing the reuse of business logic according to a function life-cycle model devised in RADON.

A special focus will be put on enabling data engineers to control the data life-cycle (ingestion, buffering, transfer, scheduling, storage) and processing (filtering, transformation, analysis) of data across the FaaS-based applications by using data pipelines, which are composed of specialized microservices (or FaaS functions), resources and communication mechanisms. Data pipeline templates will combine these building blocks into reusable TOSCA components that can be injected into TOSCA models and deployed using the RADON orchestrator.

Upon deployment, the RADON orchestrator will leverage IaC at the configuration management level to set up and wire up all the components of the application. It will automatically configure monitoring services and connect metric collectors with user-specified storage. This will primarily make passive monitoring useful for the quality assurance of the application but will also enable the orchestrator to adhere to performance requirements defined in the template. The orchestrator will be able to dynamically reconfigure the application components and scale the number of node instances according to scaling policies, while the auto-scaling of the functions is handled by the FaaS platform.

The security and privacy policies (e.g., data access control, encryption) will be implemented by the RADON orchestrator to control access to the application components at runtime and their initialization process as well in order to limit the exposure of critical functionalities, ensuring that both business logic and sensitive data are protected even before the application starts running. For enacting the security and privacy policies of FaaS-based applications (consisting of services, serverless FaaS and data pipelines) RADON orchestrator will utilize service meshes, where intelligent proxies (deployed as sidecars to microservices) mediate and route traffic between microservices. A service mesh defines a separate control plane for managing and configuring the intelligent proxies.

C. IDE & DevOps Methodology

RADON models will be exposed via a web-based graphical IDE (Eclipse Che [44]). By clicking corresponding graphical elements, users will be able to (i) define serverless functions and microservices (for developers); (ii) design IaC recipes such as TOSCA YAML files (for runtime operators). Data engineers will also rely on the IDE to implement data transformations as serverless functions or microservices, and combine these building blocks into reusable data pipeline templates. Under teamwork circumstances, users with different roles will be granted appropriate levels of access.

A DevOps methodology will be investigated, identifying stakeholders, the socio-technical system, barriers to adoption, customization methods, recurrent architectural patterns and the roles of users interacting with the RADON framework. An important feature of the methodology will be the conceptualization of different life cycles at play within an application based on serverless FaaS. This is a richer setting compared to that of a traditional methodology, as one can envision different life cycles for serverless functions, microservices and data pipelines. The identified life cycles will be archived in both user documentation and online help to provide guidance on how to decide tool workflows and make the whole framework easy to use.

The DevOps methodology will contain CI/CD methods to continuously release updated versions of the application. a baseline in this respect comes from the DICE project, a delivery tool that connects Eclipse with the Cloudify orchestrator through TOSCA. However, this baseline needs to be retrofit so that it becomes compatible with the new Eclipse Che platform. One novelty that will be introduced into the baseline is the support of advanced delivery modes, for example dark launches, whereby new features of an application can be deployed and tested at runtime while remaining invisible to all the users but the developer.

D. Quality Assurance Tools

A particular concern is raised by serverless FaaS for security and privacy. The increased granularity and expressiveness of RADON models will prompt the need to consider security and privacy with high priority in the architectural design. To delivery the business logic through serverless functions or microservices, developers have to make different trade-offs in terms of other non-functional requirements such as performance, cost, security and privacy. The decision of the optimal attack surface exposed by an architecture, as well as the implication of the fine-grained decomposition on non-functional requirements, ask for a rigorous methodology to help engineers select the right level of granularity.

RADON models will combine CDL annotations and generalized TOSCA models that support serverless FaaS and data pipelines. Developers will be able to analyze a RADON model using a hierarchy of logic programming and simulation techniques to determine whether the decomposition or aggregation of certain functions or services produces an improvement in satisfying the specified requirements. With this mechanism, it will be possible to enact progressive decisions on the model, resulting in a solution with the optimal decomposition and the satisfaction of security and privacy policies in addition to usual non-functional requirements for performance and cost. Monitoring feedbacks will also be visible on a dashboard for users to diagnose the runtime behavior of serverless functions or microservices, and then identify in a semi-automated manner what needs to be prioritized in the design.

The RADON quality assurance tools will be used to validate (i) IaC recipes, (ii) business logic encoded in serverless functions or microservices, (iii) data pipelines.

- For IaC recipes, a defect-prediction tool will be developed, combining anti-pattern detection and recent techniques from machine learning. It will address the intrinsic polyglot nature of infrastructure code, being either agnostic of IaC technologies or specific to certain IaC defects and anti-patterns.
- For business logic, a continuous testing approach will be employed, through execution at the development stage immediately preceding the actual deployment of the application, to help detect unexpected issues before they are manifested in production.
- For data pipelines, users will be required to model data flows by customizing data generation profiles, which are needed to automatically produce test data with desired characteristics (e.g., burstiness) for verifying responsiveness and scalability annotated though CDL.

IV. Industrial Use Cases

Part of the challenge of investing in FaaS is to recognize the industrial use cases where the new technology is necessary to overcome technical and cost barrier. We here illustrate three industrial use cases that demonstrate industrial domains in which the FaaS paradigm can strike a contribution and there is a potential advantage in adopting a model-driven DevOps framework such as the one we have described. The cases studies describe the commercial needs of units within three European companies: Athens Technology Center (ATC), Engineering Ingegneria Informatica (ENG) and Praqma (PRQ).

A. Tourism Promotion

ADAMO is an Android mobile application that combines the individual's tourist profile, a city's mobility network and points of interest (POIs) to share personalized experiences for travelers and residents, who can then discover the most exciting city routes customized to their preferences. The portfolio of news portals and touristic agencies will be able to suggest alternative routes for their clients and promote relevant data and services during route development. ADAMO may also play the role of a fair advertisement hub for local enterprises, while allowing city authorities to gain from the visibility of their profile. This will help showcase the touristic values of the city and connect preferable business offerings to stakeholders.

The current implementation of ADAMO is based on microservices and coarse-grained web services, which are developed in Java for the back end and in mobile Java for the Android app. The former hosts a set of components for collecting information about POIs in a city, analyzing data regarding the mobility and transportation support to move from one place to another and maintaining user personalization features. ATC exploits RESTful web services to coordinate communication among different components. These will be migrated to Docker [45] containers, applying the FIWARE [46] generic enabling (GE) technologies for (i) authentication and authorization (KeyRock, AuthZForce, Wilma); (ii) the management of POIs and routes (POI Data Provider, Object Storage, POI Proxy Swagger).

ADAMO is a general use case of RADON where novel data pipelines and data processing microservices, several FaaS-based, will be added to an existing architecture. With the help of the RADON framework, ATC will enrich ADAMO with a story-building environment, in which journalists will harness online content collected from multiple sources (e.g., social media, blogs, news reports) to compose stories about POIs in a city. This development will facilitate the needs of news portals and tourist agencies for stories about specific routes. Specifically, data processing and AI technologies will be applied for the real-time analysis of online content and their connection to POIs and routes. The resulting stories will be displayed on alternative routes to introduce POIs along them.

B. Ambient Assisted Living

In recent years, eHealth technologies has been increasingly applied to assisted living in the home environment. ENG's use case falls within this application domain. It couples a horizontal platform, the CLOE-IoT middleware, on top of which runs a vertical application for ambient assisted living called AREAS SARA. CLOE-IoT is one of the ENG cloud offerings (CLOE), which aims to simplify the development of complex and robust IoT solutions across a wide variety of IoT devices and networks. SARA is an example of a CLOE-IoT distributed application envisioned to be part of AREAS, ENG's health ERP solution. It integrates robotic and IoT technologies to preserve quality of life for elders with mild cognitive impairment or Alzheimer's disease.

Building on the CLOE-IoT platform, SARA coordinates the following nodes available in the use case: Smart Phone, acting as the hub of a body area network of wearables for mobility detection and risk assessment; Robotic Rollator, a powered, wheeled walking frame, primarily used for physical support but also capable of identifying the patient, monitoring his mobility and navigating autonomously; Robotic Assistant, a robotic component connected to a (possibly dynamic) network of embedded devices and services for monitoring the patient's activities, health status and treatment/training progress as well as for supporting cognitive skills training and notifying the patient of upcoming treatments and visits; Environment Gateway, acting as the hub of smart devices embedded in the local physical environment and providing a variety of services ranging from security monitoring over control of appliances to patient tracking.

SARA is relevant to RADON as a prototypical use case where augmenting the capacity of the middleware can accelerate the redeployment, configuration and evolution of the vertical application consisting of robotic and IoT devices. Concretely, the use case will involve the use of the RADON model to capture the existing IoT/edge landscape. This will show in the process how to code new IaC recipes to setup the robotic and embedded devices at the edge, allowing ENG to develop a reusable library of deployment templates and configuration recipes for SARA. Serverless functions to capture events and triggering of actions will then be coded. Through integration of an open-source FaaS platform (e.g., OpenFaaS [47]) with CLOE-IoT, ENG will show how to increase the capacity of this industrial middleware to give access to SARA to serverless FaaS technology. Serverless functions will be then used to detect and react to specific events relatively to the environment.

C. Managed DevOps

Embracing DevOps in industrial setting requires accepting agile development and delivery principles throughout an organization, intersecting with the socio-technical systems that revolve around the software production process. PRQ is building a novel framework, called Orbit, that consolidates the company expertise in CI/CD methods and automation to implement DevOps-style governance in organizations. Managed DevOps, the focus of this use case, is conceptually similar to an outsourcing contract that a customer stipulates with an external company, like PRQ, to restructure the business with DevOps-style agile processes and adopt and maintain the related IT tools. This use case offers the opportunity to explore the interface between serverless, DevOps, socio-technical systems, and ensure that RADON is not perceived itself as a form of lock-in by adopters. As PRQ operates mainly in consulting, Orbit is envisioned as consolidating and expanding their market offering for digital transformation and will be offered to PRQ's customers across various markets, considerably enhancing the business potential of the company.

PRQ has noted a potential to further expand the use of FaaS through enrichment of DevOps governance with automatic FaaS-based triggers and actions that react to fine-grained events generated by developers, operators and all the human actors in the socio-technical system, while they work together in developing and delivering software. For example, a certain comment in the source code monitored by serverless functions may trigger a chain of actions to alert other developers in the team or be automatically picked-up by the testing pipeline to reconfigure itself for an atypical test. Moreover, a large market has been identified towards integrating within Orbit a general-purpose function hub that consolidate the customer organization assets into libraries of functions that can be systematically reuse to rapidly create new business processes and IT-driven products. This will allow the consortium to further reason on the function lifecycle and long-term storage of technical and business process functions.

The primary goals of this use case are twofold: (i) to explore the use of adopt DevOps through externally management services. This will be achieved through automation of IT actions that are needed to ensure that the different teams working on a software product can collaborate smoothly and efficiently, while ensuring the correct setup of security and privacy rules; (ii) to explore the capabilities of the function hub component to archive complex process templates that are in use across industries relevant to this market segment, in order to enhance the capacity of the target organizations to start from day-1 to use serverless functions and build new processes on top of such stored business logic.

V. Conclusion

We have proposed a research agenda for developing an innovative DevOps framework centered around serverless computing, unlocking the advantages of the serverless FaaS paradigm to industry. Applications created with RADON will be faster, easier and cheaper to develop and operate than today, thanks to function-level scaling and billing, automated orchestration, and reuse of serverless functions, microservices and data pipelines. Technically, RADON will enable developers and operators to combine pre-packaged functions and microservices into architectural topologies that are readily deployable, automating design, deployment, testing, verification and evolution of FaaS-based applications under specified functional and non-functional requirements. A broad range of users can potentially benefit from the innovations foreseen by RADON as illustrated in our review of possible industrial use cases.

Acknowledgment

This paper has been partially supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825040 (RADON).

REFERENCES

[1] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski *et al.*, "Serverless Computing: Current Trends and Open Problems," in *Research Advances in Cloud Computing*. Springer, 2017, pp. 1–20.

[2] G. C. Fox, V. Ishakian, V. Muthusamy, and A. Slominski, "Status of Serverless Computing and Function-as-a-Service (FaaS) in Industry and Research," *arXiv preprint arXiv:1708.08028*, 2017.

[3] AWS Lambda. [Online]. Available: https://aws.amazon.com/lambda

[4] Google Cloud Functions. [Online]. Available: https://cloud.google.com/functions

[5] Microsoft Azure Functions. [Online]. Available: https://azure.microsoft.com/services/functions

[6] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley, 2012.

[7] D. Gannon, R. Barga, and N. Sundaresan, "Cloud-Native Applications," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 16–21, 2017.

[8] Jolie. [Online]. Available: https://www.jolie-lang.org

[9] M. P. Papazoglou and L. M. Vaquero, "Knowledge-Intensive Cloud Services: Transforming the Cloud Delivery Stack," in *Knowledge Service Engineering Handbook*. CRC Press, 2016, pp. 472–517.

[10] D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel, "Elaborating Requirements Using Model Checking and Inductive Learning," *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp. 361–383, 2013.

[11] D. Alrajeh, J. Kramer, A. Russo, and S. Uchitel, "Automated Support for Diagnosis and Repair," *Communications of the ACM*, vol. 58, no. 2, pp. 65–72, 2015.

[12] P. M. Duvall, S. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*. Pearson, 2007.

[13] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson, 2010.

[14] Chef. [Online]. Available: https://www.chef.io

[15] Ansible. [Online]. Available: https://www.ansible.com

[16] Puppet. [Online]. Available: https://puppet.com

[17] OpenTOSCA. [Online]. Available: https://www.opentosca.org

[18] Cloudify. [Online]. Available: https://cloudify.co

[19] Apache ARIA TOSCA. [Online]. Available: https://ariatosca.incubator.apache.org

[20] OASIS TOSCA. [Online]. Available: https://www.oasis-open.org/committees/tosca

[21] Apache Brooklyn. [Online]. Available: https://brooklyn.apache.org

[22] Juju. [Online]. Available: https://jujucharms.com

[23] Hudson. [Online]. Available: http://hudson-ci.org

[24] Jenkins. [Online]. Available: https://jenkins.io

[25] TeamCity. [Online]. Available: https://www.jetbrains.com/teamcity

[26] Bamboo. [Online]. Available: https://www.atlassian.com/software/bamboo

[27] Travis CI. [Online]. Available: https://travis-ci.org

[28] CircleCI. [Online]. Available: https://circleci.com

[29] Z. M. Jiang and A. E. Hassan, "A Survey on Load Testing of Large-Scale Software Systems," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1091–1118, 2015.

[30] P. Leitner and C.-P. Bezemer, "An Exploratory Study of the State of Practice of Performance Testing in Java-Based Open Source Projects," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ACM, 2017, pp. 373–384.

[31] R. Heinrich, A. van Hoorn, H. Knoche, F. Li, L. E. Lwakatare, C. Pahl, S. Schulte, and J. Wettinger, "Performance Engineering for Microservices: Research Challenges and Directions," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ACM, 2017, pp. 223–226.

[32] H. Schulz, T. Angerstein, and A. van Hoorn, "Towards Automating Representative Load Testing in Continuous Software Engineering," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ACM, 2018, pp. 123–126.

[33] ITEA3 TESTOMAT. [Online]. Available: https://www.testomatproject.eu

[34] ContinuITy. [Online]. Available: https://continuity-project.github.io

[35] DICE QT. [Online]. Available: https://github.com/dice-project/DICE-Quality-Testing

[36] C. A. Cois, J. Yankel, and A. Connell, "Modern DevOps: Optimizing Software Development through Effective System Interactions," in *Proceedings of the 2014 IEEE International Professional Communication Conference*. IEEE, 2014, pp. 1–7.

[37] F. Shull, D. Falessi, C. Seaman, M. Diep, and L. Layman, "Technical Debt: Showing the Way for Better Transfer of Empirical Results," in *Perspectives on the Future of Software Engineering*. Springer, 2013, pp. 179–190.

[38] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, and D. Poshyvanyk, "When and Why Your Code Starts to Smell Bad," in *Proceedings of the 37th International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 403–414.

[39] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.

[40] M. Wurster, U. Breitenbücher, K. Képes, F. Leymann, and V. Yussupov, "Modeling and Automated Deployment of Serverless Applications using TOSCA," in *Proceedings of the IEEE 11th International Conference on Service-Oriented Computing and Applications*. IEEE, 2018, pp. 73–80.

[41] Eclipse Winery. [Online]. Available: https://projects.eclipse.org/projects/soa.winery

[42] TOSCA YAML. [Online]. Available: https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML

[43] DICE TOSCA Library. [Online]. Available: http://dice-project.github.io/DICE-Deployment-Cloudify

[44] Eclipse Che. [Online]. Available: https://www.eclipse.org/che

[45] Docker. [Online]. Available: https://www.docker.com

[46] FIWARE. [Online]. Available: https://www.fiware.org

[47] OpenFaaS. [Online]. Available: https://www.openfaas.com