

# Microservices Management with the Unicorn Cloud Framework

George Feuerlicht<sup>1,2</sup><sup>a</sup>, Marek Beranek<sup>1</sup><sup>b</sup> and Vladimir Kovar<sup>1</sup>

<sup>1</sup>Unicorn College, V Kapslovně 2767/2, Prague 3, 130 00, Czech Republic

<sup>2</sup>Prague University of Economics, W. Churchill Square. 4, 130 67 Prague 3, Czech Republic


**Keywords:** Cloud Computing, Frameworks, Microservices.


**Abstract:** The recent transition towards cloud computing from traditional on-premises systems and the extensive use of mobile devices has created a situation where traditional architectures and software development frameworks no longer support the requirements of modern enterprise applications. This rapidly evolving situation is creating a demand for new frameworks that support the DevOps approach and facilitate continuous delivery of cloud-based applications using microservices. In this paper, we first discuss the challenges that the microservices architecture presents and then describe the Unicorn Cloud Framework designed specifically to address the challenges of modern cloud-based applications.

## 1 INTRODUCTION

We are now in the midst of a major transformation driven by wide adoption of cloud computing and extensive use of mobile and IoT (Internet of Things) devices. Public cloud platforms such as AWS (Amazon.com, 2017), Microsoft Azure (2017), etc. offer ever increasing range of elastic services, and practically unlimited compute power and storage capacity, allowing more flexible acquisition of IT resources and avoiding most of the limitations of traditional on-premises IT solutions. This new technology environment is creating opportunities for innovative solutions at a fraction of the cost of traditional on-premises enterprise applications. To take full advantage of these developments, organizations involved in the production of enterprise applications must adopt a suitable enterprise architecture and application development frameworks. Two key architectural requirements emerge as a result of recent technological advances: 1) the requirement to develop and deploy server-side application components on a cloud infrastructure in the form of microservices, and 2) the need to support various types of mobile and IoT devices using cross-platform client-side application components. Deploying application components on a cloud infrastructure enables applications to be shared by

very large user populations. The architecture needs to facilitate rapid incremental development of application components, ensure secure access to information and support rapid cloud deployment. There is increasing empirical evidence that to effectively address such requirements, the architecture needs to support microservices and container-based virtualization (Balalaie et al., 2016). However, it is also becoming clear that the management of large-scale clusters of containers has its own challenges and requires automation of application deployment, auto-scaling and effective control of resource usage. The need for continuous delivery and monitoring of application components impacts on the composition and skills profile of IT teams, favoring small cross-functional teams and leading to convergence of development and operations (DevOps). The separation of code development and declarative methods of environment configuration play an important role in increasing the productivity of the software development process. Furthermore, developers of enterprise applications are increasingly turning towards open source solutions that allow full control over the entire software stack, avoiding costly proprietary solutions. Also, while the use of public cloud platforms is economically compelling, an important function of the architecture is to ensure independence from

<sup>a</sup> <https://orcid.org/0000-0001-9333-5050>

<sup>b</sup> <https://orcid.org/0000-0003-0491-4275>

individual cloud providers, avoiding a provider lock-in. Finally, the architecture should reduce the complexity of application development and maintenance process and facilitate effective reuse of application components and infrastructure services. These requirements demand a revision of existing architectural principles and application development methods.

In this paper, we describe the Unicorn Cloud Framework (uuCloud). uuCloud is a part of the Unicorn Application Framework (UAF), a suite of closely integrated frameworks described in previous publications (Beranek et al., 2017), (Beránek et al., 2017), (Beranek et al., 2018). UAF was developed by Unicorn (<https://unicorn.com/cz>) - a provider of IT solutions based in Prague, Czech Republic and is used to develop solutions for organizations across a range of industry domains, including banking, insurance, energy and utilities, telecommunications, manufacturing and trade. The main objective of uuCloud framework is to document and automate the development and deployment of microservices applications and to facilitate reuse of skills and software components across multiple projects.

The paper is structured as follows. In the next section (Section 2) we review recent research publications dealing with frameworks for container-based microservices. The following section (Section 3) describes the uuCloud framework and Section 4 are our conclusions.

## 2 RELATED WORK

In this section we review recent research publications dealing with frameworks for the management of container-based microservices. Cloud-based application development frameworks and architectures have been the subject of intense recent interest by both industry practitioners and academic researchers, in particular in the context of microservices and DevOps (Mahmood and Saeed, 2013), (Thönes, 2015). According to Rimal et al. (Rimal et al., 2011) the most important current challenge is the lack of a standard architectural approach for cloud computing. The authors explore and classify architectural characteristics of cloud computing and identify several architectural features that play a major role in the adoption of cloud computing. The paper provides guidelines for software architects for developing cloud architectures.

While container technologies and microservices have revolutionized application development and

deployment, it is also evident that the use of these technologies has its challenges. The management of large-scale container-based environments requires automation to ensure fast and predictable application deployment, auto-scaling and control of resource usage. At the same time, there is a requirement for portability across different public and private clouds. To address such issues a number of open source projects aiming to support the development and deployment of cloud-based applications have been recently initiated. Prominent examples include Cloud Foundry (CloudFoundry, 2017), OpenShift (OpenShift, 2020), Docker Swarm (Naik, 2016) and Kubernetes (Burns et al., 2016b). A key idea of these open source platforms is to abstract the complexity of the underlying cloud infrastructure and present a well-designed API (Application Programming Interface) that simplifies the management of container-based cloud environments. Brewer (Brewer, 2015) in his keynote “Kubernetes The Path to Cloud Native” argues that we are in middle of a great transition toward *cloud native* applications characterized by highly available unlimited “ethereal” cloud resources. This environment consists of co-designed, but independent microservices and APIs, abstracting away from machines and operating systems. The Kubernetes project initiated by Google in 2014 as an open source cluster manager for Docker containers has its origins in an earlier Google container management systems called Borg (Burns et al., 2016b) and Omega (Schwarzkopf et al., 2013). Kubernetes objective is to facilitate *cloud native* systems that run applications and processes in isolated units of application deployment (i.e. software containers). Containers implement microservices which are dynamically managed to maximize resource utilization and minimize the cost associated with maintenance and operations. The state of objects in Kubernetes is accessed through a domain-specific REST API that supports versioning, validation, authentication and authorization for a diverse range of clients (Burns et al., 2016a). Other similar efforts in this area include OpenStack (Kumar et al., 2014) - a *cloud operating system* designed to control large pools of compute, storage, and networking resources, managed using a dashboard used by administrators to control cloud resources and to provision resources through a web interface. In (Feller et al., 2012), the authors argue that many existing frameworks have a high degree of centralization and do not tolerate system component failures, and present the design, implementation and evaluation of a scalable and autonomic virtual machine (VM) management framework called *Snooze*. Truyen et al. (Truyen et al.,

2019) evaluate the performance overheads introduced by Docker engine and two container orchestration frameworks: Docker Swarm and Kubernetes, when running CPU-bound workloads in OpenStack. The authors report that Docker engine deployments running in host mode exhibit negligible performance overheads in comparison to native OpenStack deployments, but that virtual IP networking introduces a substantial overhead in Docker Swarm and Kubernetes. They conclude that the solution involves service networking approaches that run in true host mode and offer support for network isolation between containers. Another finding discussed in the paper was that volume plugins for persistent storage have a large impact on the overall resource model of a database workload. The paper provides experimental confirmation for this assertion showing that a CPU-bound Cassandra workload changes into an I/O-bound workload in both Docker Swarm and Kubernetes. This implies that placement decisions for native or Docker engine deployments need to be recomputed when using container orchestration frameworks such as Docker Swarm or Kubernetes. The authors conclude that while container orchestration frameworks provide various advantages for automating SLO-aware placement of multi-tenant applications, to gain full benefits of this approach container orchestration frameworks need further development.

While Kubernetes is gaining momentum as a platform for the management of cloud resources with support from major public cloud platforms including Google Cloud Platform, Microsoft Azure and most recently Amazon AWS, there is a number of alternative projects that aim to address the need for *universal* framework for the development and deployment of cloud applications. However, some proposals lack empirical verification using large-scale *real-life* applications. Complicating this situation is the rapid rate of innovation in this area characterized by the current trend toward finer resource granularity with corresponding impact on the complexity of cloud resource management frameworks (Baldini et al., 2017). According to Ben-Yehuda (Agmon Ben-Yehuda et al., 2014), the trend toward increasingly finer resource granularity is set to continue resulting in improved flexibility and efficiency of cloud-based solutions. The authors assert that resources such as compute, memory, I/O, storage, etc. will be charged for in dynamically changing amounts, not in fixed bundles. They conclude that the RaaS (Resource as a Service) cloud requires new mechanisms for allocating, metering, charging for, reclaiming, and redistributing CPU,

memory, and I/O resources among multiple clients every few seconds.

### 3 uuCLOUD FRAMEWORK

As noted in section 2, currently there is a lack of agreement about a standard framework for the management of cloud-based microservices. Our solution is the uuCloud cloud management platform that facilitates provisioning, monitoring and management of elastic cloud services in the form of virtualized Unicorn Applications (uuApps). uuCloud supports hybrid cloud operation combining on-premises cloud infrastructure with public cloud infrastructures such as Microsoft Azure and AWS cloud platforms.

Importantly, uuCloud enables the development of cloud-native applications using the DevOps approach. As illustrated in Figure 1, uuCloud incorporates a comprehensive range of infrastructure services that allow rapid deployment of applications without the need to re-implement basic functionality (i.e. authentication, authorization, etc.) separately for every application.

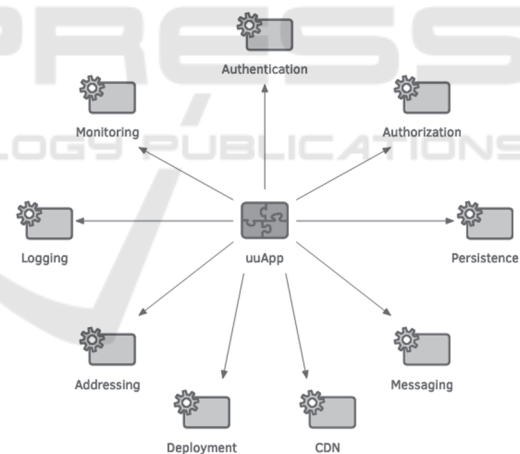


Figure 1: uuCloud framework services.

Logging and monitoring services are used by systems administrators at runtime to ensure that the system operates as specified by the SLA (Service Level Agreement).

The uuCloud framework consist of three basic components: 1) uuCloud Operation Registry – a database that maintains active information about uuCloud objects, 2) uuCloud Control Centre (uuC3) - a tool that is used to automate deployment and operation of container-based microservices, and 3) Operation Runtime that provides facilities for

monitoring runtime services. To ensure portability and to reduce overheads uuCloud uses Docker container-based virtualization (Docker, 2015). Docker containers can be deployed either to a public cloud infrastructure (e.g. AWS or Microsoft Azure) or to a private (on-premises) infrastructure (e.g. Plus4U – Unicorn cloud infrastructure).

uuCloud supports multi-tenancy and can accommodate a large number of *Tenants* using scalable resources across multiple cloud platforms. Each Tenant typically represents a separate organization (e.g. a publisher of online learning courses). Tenants are assigned resources from *Resource Pools* using the mechanism of *Resource Lease* that specifies usage constraints such as the maximum number of vCPUs (virtual CPUs), RAM size and size of storage that are available for the operation of a specific Tenant. A Tenant can be allocated several Resource Pools, for example separate Resource Pools for production and development. Each Tenant consumes its own resources and is monitored and billed separately. Applications can be shared among multiple Tenants with the Tenant in whose Resource Pool the application is deployed being responsible for the consumed resources. Each *Host* (physical or virtual server) is allocated to a logical aggregation of computing resources called a *Resource Group*. Resource groups can have additional *Resources* (e.g. MongoDB (MongoDB, 2020), Oracle DBMS,

Gateways, etc.). Each Resource Group belongs to a *Region* – a separate IT infrastructure sourced from a single cloud provider with co-located compute, storage and network resources characterized by low latency and high bandwidth connectivity. Regions are grouped into *Clouds* that can be composed of multiple public and private cloud platforms.

A *Node* is a unit of deployment with hardware characteristics that include virtual CPU count (vCPU), RAM size and ephemeral storage. Nodes are classified according to *NodeSize*, e.g. M (Medium size: 1xvCPU, 1 GB of RAM, 1 GB of ephemeral storage) or L (Large size: 2xvCPU, 2 GB of RAM, 1 GB of ephemeral storage). Nodes are further classified as *synchronous* or *asynchronous* depending on the behaviour of the application that the node virtualizes. Nodes are grouped into sets of nodes with identical functionality called *NodeSets*.

Figure 2. illustrates the mapping of uuCloud objects to Microsoft Azure and Docker Swarm (Naik, 2016). uuCloud constitutes a middleware layer between the Microsoft Azure cloud platform and the Docker cluster orchestration platform Docker Swarm. uuCloudInstance represents a private or public cloud instance. There is one to one mapping between Azure Regions, Resource Groups and virtual Machines, and the corresponding uuCloud objects: uuRegions, uuResourceGroups and uuHosts, and Swarm Clusters and Swarm Nodes that map into uuResourceGroups and uuHosts.

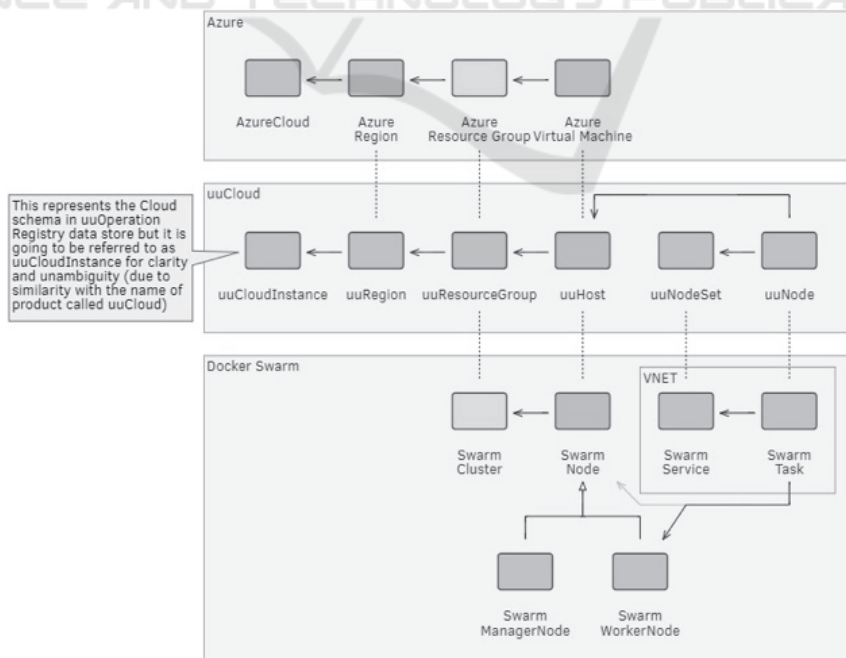


Figure 2: Mapping between uuCloud, Microsoft Azure and Docker Swarm.

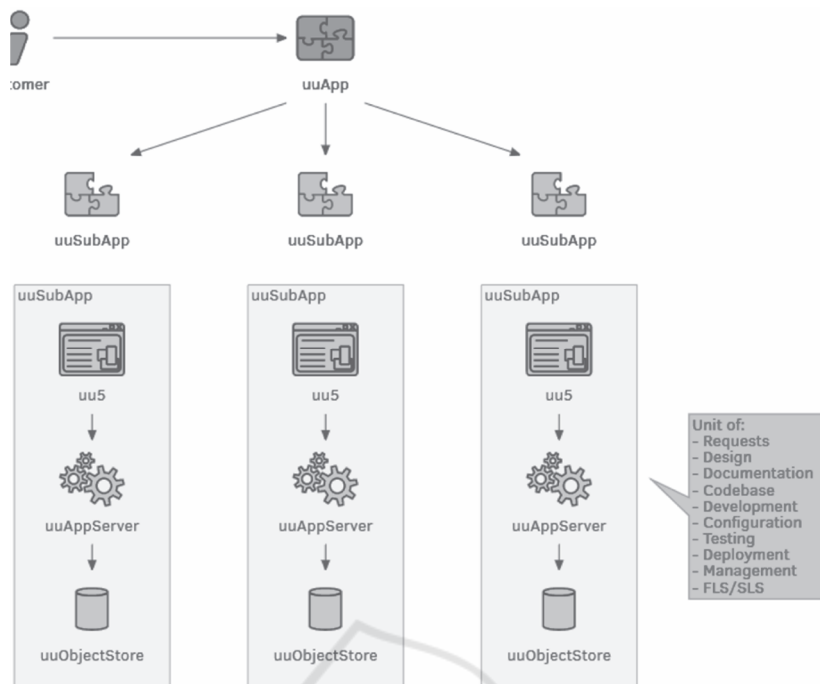


Figure 3: Mapping of applications (uuApps) to microservices.

The function of the Unicorn Cloud Control Center (uuC3) is to automate the management and operation of container-based microservices via a REST API. uuCloud API includes deployment commands (deploy, redeploy, undeploy), commands to increase or decrease the number nodes in a nodeset (scale), commands that control sharing of applications (share, unshare) and commands that are used to monitor the status of nodes and applications (getStatus, getAttributes, etc.).

uuApp application is composed of sub-applications (uuSubApps) – independent units of functionality that implement a particular set of related business functions (e.g. administration of online learning courses). As illustrated in Figure 3, each uuSubApp is implemented as a containerized microservice (uuAppServer) and is associated with its own structured or binary storage (uuObjectStore). We made an architectural decision to implement each sub-application as a single containerized microservice to ensure fast access to persistent data and to maintain security and consistency of the underlying data sources. In general, each microservice has its own life-cycle that includes requirements specifications, design, development, testing, deployment and monitoring phases. To allow finer control over computing resource individual application modules (Separated Performing Parts) can be directly addressed. This allows selected

modules of the uuSubApp to be deployed on separate computing nodes improving application scalability and reducing resource utilization.

Before a microservice can be deployed to a particular node (or NodeSet) the developer creates a *Node Image* from a uuSubApp and a *Runtime Stack* that contains all the related archives and components needed to run the application (i.e. system tools, system libraries, etc.). The resulting Node Image constitutes a unit of deployment, its metadata is recorded in the Operation Registry and the Node Image is published to the private Docker Image Registry. During deployment, the uuC3 Deploy command reads the Deployment Descriptor (JSON file that contains deployment attributes) and searches the Operation Registry for an existing application deployment object.

At runtime, client applications send requests to a router via a gateway (uuGateway) that passes each request to a load balancer. The load balancer selects a Node from a NodeSet of functionally identical nodes, optimizing the use of the hardware infrastructure and providing a failover capability (i.e. if a particular Node is not responsive the request is re-directed to an alternative Node).



## 4 CONCLUSIONS

While the use of microservices and container-based virtualization brings many benefits, the highly distributed nature of the resulting applications and the short software release cycles that characterize the DevOps approach present many challenges to organizations involved in the development of cloud-based applications. The management of large-scale container-based microservices environments requires automation to ensure fast and predictable application deployment, auto-scaling and control of resource usage. Furthermore, there is a need to support the ever-expanding range of various types of mobile and IoT devices using cross-platform client-side application components. Suitable application development frameworks and associated methods and tools are an essential pre-requisite for achieving successful project results on a repeatable basis.

The uuCloud framework described in this paper is an integral part of the Unicorn Application Framework that was developed specifically to address the requirements of modern mobile cloud-based applications. The UAF is currently used for the development of large-scale enterprise applications at Unicorn. An important difference between UAF and other similar frameworks (e.g. Kubernetes) is that UAF is not limited to the management of container-based environments (container orchestration) but it is a suite of closely integrated frameworks that address a comprehensive range of requirements of modern mobile cloud-based applications over the entire systems development lifecycle. An important benefit of the uuCloud framework is its ability to manage complex multi-tenant environments deployed across multiple (hybrid) cloud platforms, hiding the heterogeneity of the underlying cloud infrastructures. uuCloud manages cloud metadata, supports automatic application deployment and autonomic scaling of applications. uuCloud multi-tenancy derives benefits from the economies of scale. With a large number of tenants deployed on a public cloud infrastructure such as AWS or Microsoft Azure, resources can be purchased in bulk and then allocated in smaller *packages* to individual tenants who benefit from the reduced overall cost. Furthermore, uuCloud includes a detail lifecycle methodology that captures the knowledge and expertise gained across numerous projects and describes the entire design and development process starting with the identification of business use cases and ending with the implementation of well-engineered, component-based software applications.

Developing cloud frameworks such as uuCloud has its challenges, in particular the need to keep up with the rapid development and ever-increasing range of cloud services available from public cloud providers. Incorporating new types of services, for example AWS Lambda serverless compute services, and AWS Machine Learning services, requires continuous evolution of the framework and constitutes the focus of our current efforts.

## REFERENCES

2017. *Microsoft Azure: Cloud Computing Platform & Services* [Online]. Available: <https://azure.microsoft.com/en-au/> [Accessed 22 August 2017 2017].
- Agmon Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A. & Tsafir, D. 2014. The rise of RaaS: the resource-as-a-service cloud. *Communications of the ACM*, 57, 76-84.
- Amazon.Com. 2017. <http://aws.amazon.com/> [Online]. Available: <http://aws.amazon.com/> [Accessed 7 July, 2017 2017].
- Balalaie, A., Heydarnoori, A. & Jamshidi, P. 2016. Microservices architecture enables DevOps: migration to a cloud-native architecture. *IEEE Software*, 33, 42-52.
- Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R. & Slominski, A. 2017. Serverless computing: Current trends and open problems. *Research Advances in Cloud Computing*. Springer.
- Beránek, M., Feuerlicht, G. & Kovář, V. Developing enterprise applications for cloud: the unicorn application framework. *International Conference on Grid, Cloud and Cluster Computing, GCC, 2017*.
- Beranek, M., Kovar, V. & Feuerlicht, G. Framework for Management of Multi-tenant Cloud Environments. 2018 Seattle, USA. Springer International Publishing, 309-322.
- Beranek, M., Stastny, M., Kovar, V. & Feuerlicht, G. Architecting Enterprise Applications for the Cloud: The Unicorn Universe Cloud Framework. *International Conference on Service-Oriented Computing, 2017*. Springer, 258-269.
- Brewer, E. A. Kubernetes and the path to cloud native. *Proceedings of the Sixth ACM Symposium on Cloud Computing, 2015*. ACM, 167-167.
- Burns, B., Grant, B., Oppenheimer, D., Brewer, E. & Wilkes, J. 2016a. Borg, omega, and kubernetes. *Queue*, 14, 10.
- Burns, B., Grant, B., Oppenheimer, D., Brewer, E. & Wilkes, J. 2016b. Borg, omega, and kubernetes. *Queue*, 14, 70-93.
- Cloudfoundry. 2017. *Cloud Application Platform - Devops Platform, Cloud Foundry* [Online]. @cloudfoundry. Available: <https://www.cloudfoundry.org/> [Accessed 28 September 2017].

- Docker. 2015. *What is Docker* [Online]. @docker. Available: <https://www.docker.com/what-docker> [Accessed 21 August 2017].
- Feller, E., Rilling, L. & Morin, C. Snooze: A scalable and autonomic virtual machine management framework for private clouds. *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, 2012. *IEEE Computer Society*, 482-489.
- Kumar, R., Gupta, N., Charu, S., Jain, K. & Jangir, S. K. 2014. Open source solution for cloud computing platform using OpenStack. *International Journal of Computer Science and Mobile Computing*, 3, 89-98.
- Mahmood, Z. & Saeed, S. 2013. *Software engineering frameworks for the cloud computing paradigm*, Springer.
- Mongodb. 2020. *MongoDb* [Online]. Available: <https://www.mongodb.com/> [Accessed 20 January 2020].
- Naik, N. Building a virtual system of systems using Docker Swarm in multiple clouds. *2016 IEEE International Symposium on Systems Engineering (ISSE)*, 2016. *IEEE*, 1-3.
- Openshift 2020. OpenShift Container Platform by Red Hat, Built on Kubernetes.
- Rimal, B. P., Jukan, A., Katsaros, D. & Goeleven, Y. 2011. Architectural requirements for cloud computing systems: an enterprise cloud approach. *Journal of Grid Computing*, 9, 3-26.
- Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M. & Wilkes, J. Omega: flexible, scalable schedulers for large compute clusters. *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013. *ACM*, 351-364.
- Thönes, J. 2015. Microservices. *IEEE Software*, 32, 116-116.
- Truyen, E., Van Landuyt, D., Lagaisse, B. & Joosen, W. Performance overhead of container orchestration frameworks for management of multi-tenant database deployments. *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 2019. *ACM*, 156-159.