Guest Editors' Introduction

Microservices and Containers

Fred Douglis
Perspecta Labs

Jason Nieh
Columbia University

Microservices, which allow an application to be comprised of many independently operating and scalable components, have become a common service paradigm. The ability to construct an application by provisioning these interoperating components has various advantages, including the isolation and independent development of tools such as key-value stores, authentication, logging, and many others.

Containers are one type of system infrastructure that is commonly used to support microservices. With container management systems like Docker and orchestration systems like Kubernetes to control applications and dynamically provision their resources, cloud services can be extremely scalable, reliable, and reactive. However, other systems beyond containers can be used to support microservices, and many applications other than microservices benefit from containerization.

Containers should be contrasted with another virtualization technique, traditional virtual machines. Back in March 2013, *IEEE Internet Computing* published a special issue on virtualization. By then, virtual machines had become a popular way to isolate applications, as a specialized server could run in its own virtual machine while sharing a pool of available resources (such as a VMware ESXi server). Virtual machines provide a convenient way to encapsulate state (so that machines can be migrated among servers) and to deploy a service in a predictable fashion. For instance, a service can be wrapped in a deployable template that can be installed into a virtualization environment with just a few configuration steps.

Containers, by comparison, provide a way to virtualize and isolate the operating system, allowing multiple applications to run in a single operating system; i.e., it is the software rather than the hardware that is virtualized. Without the need to run multiple operating system instances, containers can be more lightweight and potentially easier to manage. But like virtual machine templates, containers can have specifications that define exactly what software environment an application is to be run within. For instance, a container might be created with a specific version of Ubuntu, in which a specific version of python and python libraries would execute.

Given the ability to create services with various interacting containers, which may execute on one or many nodes, complex applications can be synthesized by combining these services in interesting ways. In particular, as demand varies, the individual microservices can be replicated to scale with demand or reduced to fit current requirements. To support this adaptivity, the overall service must be architected to handle parallelism of individual microservices and to perform appropriate selection of the available instances to maximize performance.

Digital Object Identifier 10.1109/MIC.2019.2955784
Date of current version 17 January 2020.

IN THIS ISSUE

There are three articles in this theme issue on microservices and containers. The first two focus on how microservices should best be used, whereas the third provides a case study of containerization in a specific application context.

The first article reports on work from the Standard Performance Evaluation Corporation (SPEC) Research Group on Cloud Computing (https://research.spec.org/working-groups/rgcloud.html). In "The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms," Van Eyk *et al.* describe best practices for Functions-as-a-Service. FaaS is a key example of the elastic scalability of microservices described above: instantiate functions when they are needed, and eliminate them when not. The authors evaluate numerous existing examples of serverless computing, which support the FaaS model, and describe how it is best deployed. They compare a number of platforms, such as Kubernetes, services from Amazon Web Services, Apache, and others; and many more.

Akbulut and Perros in "Performance Analysis of Microservice Design Patterns," focus more specifically on the performance of microservices. They study several metrics (query response time, efficient hardware usage, hosting costs, and packet loss rate) as applied to three design patterns.

1) An API gateway, which acts as a load balancer and a guard against overload (note that this gateway bears some resemblance to the application described in the third paper, below).
2) A chain of microservices that pass data from stage to stage for different types of processing.
3) Asynchronous messaging, using RabbitMQ.

Finally, Amirante and Romano authored "Container NATs and Session-Oriented Standards: Friends or Foe?" This article takes a very different view of the areas covered by this special issue, with the focus on containers and in particular the network isolation that is common in container environments. In particular, they observe that Docker isolates applications by providing them one Internet Protocol (IP) address and then translating that using "network address translation" for use outside the container. Some applications need to know the IP address by which they are reached from the outside, and forcing NATs on such applications raise a level of complexity. Amirante and Romano explain the issues and propose some possible work-arounds.

Fred Douglis received the Ph.D. degree in computer science from U.C. Berkeley, Berkeley, CA, USA. He is currently a Chief Research Scientist with Perspecta Labs, Basking Ridge, NJ, USA, where he works on applied research in the areas of blockchain, edge computing, and security. He was previously with companies including Matsushita, AT&T, IBM, and (Dell) EMC. His research interests included storage, distributed systems, web tools and performance, and mobile computing. He is a member of the IEEE Computer Society Board of Governors and a fellow of the IEEE. He served as EIC of *IEEE Internet Computing* from 2007 to 2010 and has been on its editorial board since 1999. Contact him at: f.douglis@computer.org.

Jason Nieh received the BS degree from Massachusetts Institute of Technology, Cambridge, MA, USA, and the MS and Ph.D. degrees from Stanford University, Stanford, CA, USA, all in electrical engineering. He is currently a Professor of Computer Science and the Co-Director of the Software Systems Laboratory, Columbia University, New York, NY, USA. He has served as a consultant to both government and industry, including as the technical advisor to nine States on the Microsoft Antitrust Settlement, and as an expert witness before the US International Trade Commission. He was previously the Chief Scientist of Cellrox and Desktone, acquired by VMware. He has made research contributions across a broad range of areas, including operating systems, virtualization, computer architecture, thin-client computing, cloud computing, mobile computing, multimedia, web technologies, and performance evaluation. Technologies he developed are now widely used in major operating system platforms, including Android and Linux, and are built into ARM processors, billions of which ship each year. He is an IEEE Fellow. Contact him at: nieh@cs.columbia.edu