SOFTWARE ARCHITECTURES ARE CONSTANTLY EVOLVING. We initially had, and still have, monolithic software architectures, where the deployed software stack is one long string of code that does all, including functional and nonfunctional application requirements. This is sometimes referred to as tightly coupled code. Such an approach might suffice for small applications developed by small teams as small projects. However, as complexity (in terms of more involved functional and nonfunctional requirements) increases, such an approach becomes inefficient, costly, and time consuming. Making a change is terrifying because you’ll need to retest the whole application. Starting time can be slow because you need to upload large amounts of code. The whole thing can become a nightmare. Another interesting twist here is that because the code is so tightly coupled, it’s very difficult to incorporate new technology trends.

Modular software architectures evolved mainly to ease the complexities around monolithic architectures. They include loosely coupled modules, à la distributed applications. Functions, services, and microservices are other examples of modular software architectures. They differ in their size and complexity, the mechanisms for interconnecting them, the scope of integration, and whether they deliver single or multiple tasks. For example, service-oriented architectures (SOAs) bring services together through an enterprise service bus (ESB). An ESB isn’t as simple as it might seem; it must include the glue for all the connectivity and integration among all the services.

The key architectural advantage of modular architectures is that they tackle the complexity of monolithic architectures. There are other advantages, however, mainly as side benefits to breaking the code into smaller pieces:

- It’s easier to make changes, update, and test.
- There are fewer barriers to introducing new technology trends.
- They’re likely faster to start.
- It’s easier to mix and match modules with different profiles in terms of processor and memory needs, resulting in much better resource utilization.
- It’s easier to construct applications by bringing together modules with different functions.

The market has been positioning microservices as the hottest new trend in software development, and it might be. But the confusion around microservices is everywhere. When you ask N people to define microservices or what the typical size of a microservice is, you’ll likely get N + M different definitions. Moreover, many mistakenly equate microservices with containers.

So what are microservices? They’re programs with a single task (or unit of work) that also include all the connectivity to the outside world as well as the runtime requirements to run the task. (Note that the word “task” is generic and refers to the smallest function possible, but no smaller.) Regardless, microservices inherit all the benefits of a modular architecture. Also increasing the developer community’s interest in microservices are containers and DevOps, which evolved around the same time that microservices did. Containers tailor themselves nicely to microservices because they can be deployed with much less overhead than virtual machines. DevOps represents an approach to developing, testing, and running code with tighter collaboration between developers, testers, and operators. Read more about this in David Linthicum’s “Cloud Tidbits” column.

Microservices

FROM THE EDITOR IN CHIEF COLUMN

Mazin Yousif
T-Systems International
mazin@computer.org

IEEE Cloud Computing Published by The IEEE Computer Society 2325-6095/16/$33.00 © 2016 IEEE
Microservices (or modular architectures in general) are better suited for the many complex applications we’re building these days. This includes enterprise applications (that is, confined within the enterprise) as well as Web-scale applications, where companies need to scale to reach consumers worldwide. Microservices, specifically, work well for new types of applications such as the Internet of Things, where single-function sensors and actuators are deployed in the field.

Given the complexities of our business environments, technology’s role in the social fabric, and the flat-world view of things, I’m encouraged that technological evolutions have brought us microservices, containers, and DevOps.

The columns in this issue address various topics related to microservices. As mentioned earlier, David Linthicum provides a generic overview of microservices and relates them to containers and DevOps. “Cloud Economics” guest author Andy Singleton addresses the costs and benefits associated with microservices. In the “Cloud and the Law” column, Christian Esposito, Aniello Castiglione, and Kim-Kwang Raymond Choo look at the possible security challenges around microservices and related mitigation topics requiring more research. In “Blue Skies,” Maria Fazio, Antonio Celesti, Rajiv Ranjan, Lydia Chen, Chang Liu, and Massimo Villari look at scheduling and efficient resource management for microservices. Finally, in “Standards Now,” Alan Sill explores how microservices exploit both modern and historical standards and looks at the future of microservices development.

One last item of news is the addition of Christine Miyachi to IEEE Cloud Computing’s editorial board (see the sidebar for a brief biography). She currently chairs the IEEE Special Technical Community of Cloud Computing and has worked diligently to expand its reach to more than 10,000 subscribers—an outstanding accomplishment!

MAZIN YOUSIF is the editor in chief of IEEE Cloud Computing. He’s the chief technology officer and vice president of architecture for the Royal Dutch Shell Global account at T-Systems International. Yousif has a PhD in computer engineering from Pennsylvania State University. Contact him at mazin@computer.org.

Welcome to Christine Miyachi, the newest member of the IEEE Cloud Computing editorial board. Christine Miyachi has almost 30 years of experience working for startups and large corporations. She’s currently a systems engineer and architect at Xerox Corporation and holds several patents. She works on Xerox’s Extensible Interface Platform, which is a software platform upon which developers can use standard Web-based tools to create server-based applications that can be configured for the multifunction peripheral’s touchscreen user interface. Miyachi graduated from the University of Rochester with a BS in electrical engineering. She holds two degrees from the Massachusetts Institute of Technology: an MS in technology and policy/electrical engineering and computer science and an MS in engineering and management. Miyachi writes a blog about software architecture (http://abstractsoftware.blogspot.com). For more, see www.christinemiyachi.com.