WHY YOU SHOULD BE TALKING ABOUT MICROSERVICES

By Neveen Awad, Stuart Scantlebury, and Firas Sleiman

On their leaner-faster-better to-do lists, many companies have already checked off practices like agile and DevOps, which foster collaboration and responsiveness in meeting customers' needs. But they shouldn't put down their pens just yet. Microservices also help businesses create capabilities and value—at digital speed. They're proven and potent. They also tend to be poorly understood.

Yet the idea behind microservices is actually simple. Instead of writing an application as one large "monolithic" block of code, developers link small, independent, easily reusable pieces. Each piece—a microservice—contains all of the code, interfaces, and data (or links to the data) necessary to perform a particular service or small set of services, such as updating a customer's address or deleting customer information (the list is virtually boundless). Self-contained and ready to go, microservices can be plugged into a wide array of applications. They can be shared, and leveraged, throughout an organization.

Microservices aren't a secret. Companies like Netflix and Uber have been using them for years, creating extensive catalogs that facilitate fast-paced software development. But the paradigm isn't just for digital natives. By understanding how microservices work, their requirements, and a caveat or two, companies of all backgrounds can benefit from the flexibility and efficiency these bite-sized programs promise.

Microservices Put Capabilities and Value Front and Center

Microservices give you a lot of reasons to like them. For one thing, they play well with others. Teams looking to use a microservice don't need to know how it works—what programming language it is built on, where its data is stored, or what the internal software logic looks like. They only need to "talk" to that microservice, sending a request—typically through an application programming interface (API)—and receiving an answer. This means that developers can easily and quickly tap into whatever microservices fit their needs.

While microservices can be used to write or rewrite an application, their real value becomes apparent when they work as reusable components in multiple applications. A life insurance company, for example, might build a "suggest next best action" microservice (prompting an agent or advisor to make a suggestion to a customer) for its fixed-annuities business, and then reuse it in its term-life business.

Meanwhile, the decoupled nature of microservices allows teams to create and improve them without worrying about complex integrations. Microservices may work with applications, and with each other, yet they are built and updated as independent units. This lets developers create and improve a microservice quickly. A team can focus on one small standalone piece of code—sometimes less than a dozen lines long, sometimes more than a thousand—and avoid complex testing (necessary under the monolithic approach) to ensure that the changes don't "break" an application.

Indeed, even if a microservice does go down, it won't take everything else down with it. In a well-designed architecture, if a microservice fails, another version of that microservice (or perhaps an earlier one that worked well) will come up in its place, avoiding the domino effect that can crash a monolithic application. In effect, the microservices architecture—which includes an ecosystem of automation tools—functions like the watertight compartments on a ship, where flooding is contained and disaster averted while repairs are made.

Moreover, the team responsible for a microservice has great flexibility in how it does its work. Since microservices are self-contained, developers are free to use whatever programming language, tools, and databases best suit their purpose. And they can change those languages, tools, and databases over time. As long as the APIs keep working, people outside the team never need concern themselves with the microservice's pedigree.

It gets better still. The automation tools integral to the microservices architecture enable what's known as a continuous deployment pipeline. In effect, all of the tedious, labor-intensive activities associated with a monolithic application world—such as testing, code integration and compiling, and infrastructure provisioning—are automated.

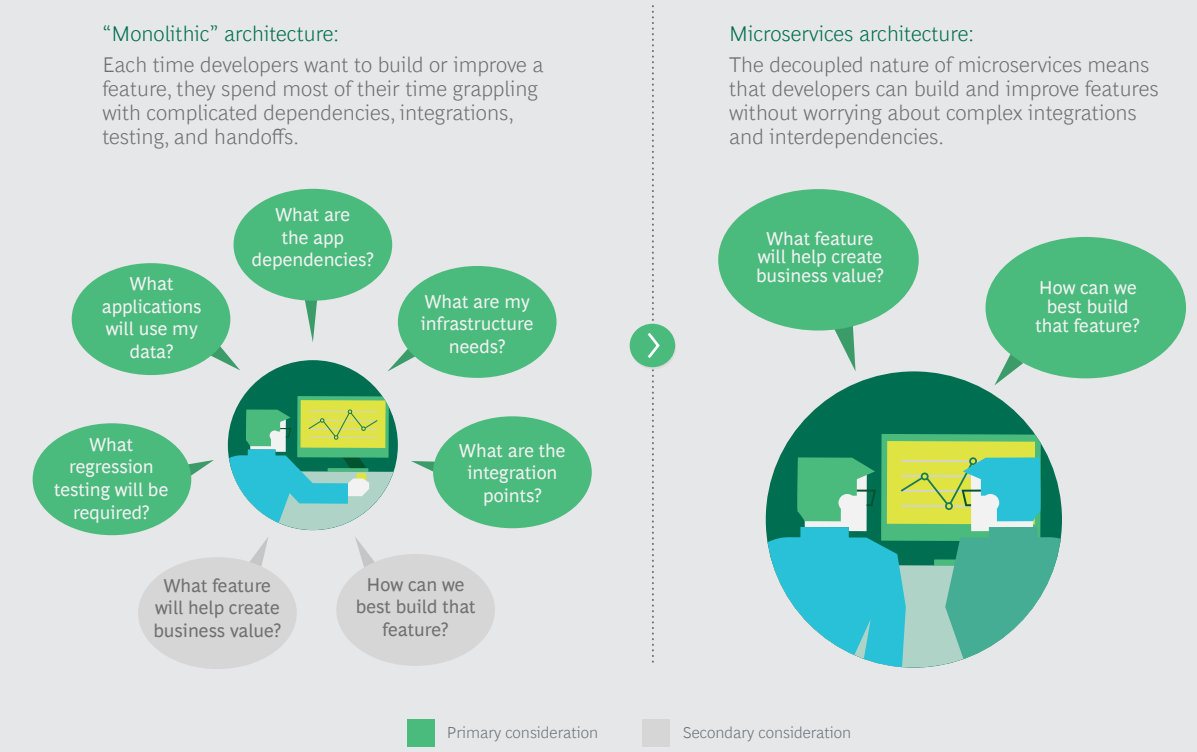
The upshot is that software teams can focus more on creating and rapidly delivering business value. With monolithic applications, developers often get so bogged down in their checklists that the crux of their mission—identifying how best to serve and satisfy customers—often winds up a secondary consideration. Microservices flip that dynamic, pushing the "side" issues to the sidelines where they belong. (See the exhibit.)

When—and When Not—to Use Microservices

Given all the benefits of microservices, why not use them for everything? To be sure, the companies at the forefront of the architecture tend to be prolific microservice coders. Uber, for instance, uses thousands of microservices to support its mobile apps, internal and infrastructure services, and products.

Yet in some cases, a company might want to stick with its existing architecture. That's because of the loosely coupled nature of microservices. Because they are self-contained, microservices rely on links—generally APIs—to communicate with applications or other microservices. This arrangement introduces a certain amount of latency, or delay, while the communications are in process (in effect, you're hopping from one microservice to another, instead of having everything hardwired together). In most cases, the latency is inconsequential. But in certain scenarios—for example, high-volume, mission-critical transactions where speed is imperative—latency could cause real problems. Consider a high-frequency trading desk, where prices change quickly. A delay of even a fraction of a second could cause disarray (traders think they are buying or selling at a certain price, but in reality, another price applies).

Focus More on Business Value with Microservices



Source: BCG analysis.

Keep in mind, too, that moving to microservices means laying a fair amount of groundwork and contending with certain complexities. For example, companies need to create—and manage—the teams that will build and continually update the microservices. And there may be dozens of teams, each with a handful of engineers. Companies will also have to decide on the appropriate data architecture: will each microservice have its own repository for the data it uses or will it pull in data from a shared repository? Microservices might perform faster and more autonomously when data is "local" but be easier to design and deploy when data is shared. There are the APIs and pipeline automation tools to manage, as well.

But perhaps the biggest task—and challenge—is determining which existing applications are best suited for a microservices makeover. In effect, companies need to triage their portfolio, prioritizing applications where a transition to microservices will bring the greatest ROI. To this end, we recommend a multi-pronged approach:

- **Evaluate.** Companies should assess existing applications according to three criteria: their strategic alignment with business priorities; the degree of change they require; and real-time performance requirements. The best candidates for microservices will be highly strategic applications that will likely need a lot of new and updated features and that can tolerate some latency. This evaluation process also brings another benefit: it helps companies consolidate the portfolio, identifying applications that are redundant or no longer needed to drive strategic goals or keep the lights on.

- **Map.** Next, companies should map applications into one of four buckets: replace, leave as is, outsource, or retire. Applications with high strategic alignment, a high degree of change, and tolerance for some latency will typically go in the "replace" group, tagged for a transition to microservices. Applications with lower strategic alignment can be left as is—or, if they require minimal or

no changes, they can be outsourced to reduce costs and focus internal development teams on areas of key business value and change. Finally, applications that are no longer required can be retired.

- **Prioritize.** For the replace and outsource buckets (and, less crucially, the retire bucket), companies should determine which applications should take precedence. While prioritization is a bit of an art, the general strategy is to factor in the effort required (so as to identify a few quick wins) and the business impact.

Savvy companies will add a couple of footnotes to these steps. First, it's good practice to re-evaluate applications on a regular basis. Business conditions—and strategic goals—change, so applications may need to be reclassified over time. The "retire" bucket is particularly likely to see rolling admissions. Second, it's important to look holistically at applications. A key characteristic of microservices is that they lend themselves to reuse. And reuse is a boon to efficiency, reducing development costs, effort, and time. So instead of looking at each application in a vacuum and deciding how it can be migrated to microservices architecture, companies should be looking across their portfolio, identifying opportunities to reuse microservices and avoid duplication of efforts.

Creating a Microservices-Ready Organization

The fast-moving, distributed nature of microservices requires companies to make organizational, technical, and cultural shifts. The good news is that many companies are already well down that road, thanks to their embrace of agile methodologies and DevOps. Indeed, agile, DevOps, and microservices form a sort of next-gen trifecta in how companies develop, deploy, and update applications.

We've found that organizations that succeed in creating—and prospering from—microservices embrace certain core principles and practices:

- **Agile Methodologies.** Many companies that develop microservices do so with cross-functional teams. Culled from both the business and technology sides of the company, these teams are responsible for solving specific business problems. This cross-functional structure is a core tenet of agile. Indeed, agile and microservices can go particularly well together. Agile stresses developing and testing small increments of code and incorporating feedback in quick iterations. Small and self-contained, microservices are quite amenable to frequent and fast development cycles.

- **Continuous Improvement and Deployment.** Since microservices can be updated and tested independently, developers can, and should, make continuous improvements. DevOps—which emphasizes automation in the software lifecycle—can greatly speed up testing and deployment. This enables teams to continually adapt their microservices to changing requirements and to deliver those updates quickly.

- **Thinking Like a Performance Engineer.** With microservices, latency—and its impact on a business outcome—becomes a key consideration. So developers need to think more like performance engineers. For example, perhaps a potentially detrimental delay could be avoided or reduced by employing techniques like precomputing, instant rendering, or caching.

- **New Tools.** Companies can call on an array of tools to help them create and manage microservices. Open-source, cloud-based platforms like Cloud Foundry and WSO2 let developers build and deploy microservices in containers. Each container holds all of the components needed to run the microservice: code, libraries, settings, and so on. Companies should keep in mind, though, that containers are not the only way to deploy microservices. Also an option: virtualization. While many developers view containers as the more

"modern" approach (losing the added layer of virtualization and making microservices more programming-language and technology agnostic), some companies feel more comfortable taking an intermediate step. For them, virtualization fits the bill. Another handy tool is a service mesh. This is a software component—sitting above the microservices—that facilitates communications among services. In effect, developers can offload many network functions to the service mesh—and focus more on business features and capabilities.

- **New Talent.** Of course, none of these shifts and tools will mean much if companies don't have the right talent. Microservices require teams with strong skills in areas like DevOps, data architecture, security, testing automation, and performance engineering. At the outset, few companies are likely to have a deep bench of such talent. But there are ways to fill the roster. One approach is internal incubation, through training and hands-on experience. Another is looking outside the company, outsourcing work to third parties or recruiting new in-house talent (or even acquiring new companies). Savvy companies also know to get the word out that they're developing and deploying microservices—which informs prospective employees and partners that there's innovative work afoot, and more to come.

- **Leading from the Top.** For the microservices journey to succeed, management needs to support it every step of the way. It's not enough to create cross-functional teams and embrace a fail-and-learn-fast approach to development. Those concepts need to be ingrained into the company's DNA. That happens only when senior executives put their words—and their weight—behind them.

MICROSERVICES HELP COMPANIES get right to the point. By focusing on self-contained bits of code and data—instead of intricate integrations—developers can deliver digital capabilities at the pace customers demand and growth requires. Readily reusable, microservices can be plugged in wherever they're needed. Like sausages, you don't need to know how they're made. But instead of heartburn, you get efficiency and speed.

About the Authors

Neveen Awad is a managing director and partner in the Detroit office of Boston Consulting Group. You may contact her by email at awad.neveen@bcg.com.

Stuart Scantlebury is a senior advisor in the firm's Boston office. You may contact him by email at scantlebury.stuart@advisor.bcg.com.

Firas Sleiman is an associate director and an architecture practice lead in the Washington, DC office of BCG Platinion. You may contact him by email at sleiman.firas@bcgplatinion.com.

Boston Consulting Group partners with leaders in business and society to tackle their most important challenges and capture their greatest opportunities. BCG was the pioneer in business strategy when it was founded in 1963. Today, we help clients with total transformation—inspiring complex change, enabling organizations to grow, building competitive advantage, and driving bottom-line impact.

To succeed, organizations must blend digital and human capabilities. Our diverse, global teams bring deep industry and functional expertise and a range of perspectives to spark change. BCG delivers solutions through leading-edge management consulting along with technology and design, corporate and digital ventures—and business purpose. We work in a uniquely collaborative model across the firm and throughout all levels of the client organization, generating results that allow our clients to thrive.