blog.runscope.com

5 Reasons Not to Use Microservices — Runscope Blog

7-8 minutes

Michael Churchman is a guest contributor for the Runscope blog. If you're interested in sharing your knowledge with our readers, we would love to have you! Please fill out this short form and we'll get in touch with you.

Are microservices always the right solution when it comes to software design? Probably not. Very few things in this world are always the 100% right solution, and microservices are no exception.

In this post, we'll talk about when it is reasonable to consider using microservices in designing or redesigning your application, and when it might make sense to avoid using microservices.

First, let’s take a quick look at what microservices are, and what is good about them.

The What and Why of Microservices

A microservice is, as the name implies, a discrete software service, typically defined at the smallest reasonable scale within the context of an application. ("Send document to system printer driver" may be a reasonable microservice, but "print the letter 'n'" is probably not reasonable.) When an application is made up of microservices, they are deployed and managed individually (or in pods/clusters) but work together to perform the functions of the application.

This means that individual microservices can be updated without having to redesign or update the entire application. It also means that the failure of a single microservice (or several microservices) will not necessarily bring the application down, and a compromised microservice will not necessarily leave the entire application vulnerable. For very large and complex applications, microservice architecture may make the entire system much more manageable than monolithic (traditional, non-microservice) architecture.

Complex Means Complex

Why not use them all the time, then? As it turns out, what works for large and complex programs does not always work at a smaller scale, and what makes sense when designing a new application does not always make sense when maintaining or updating an existing application.

When it comes to microservice architecture, complexity is perhaps the key factor. As Martin Fowler, one of the most influential thinkers in modern software development, states, "...don't even consider microservices unless you have a system that's too complex to manage as a monolith..." In other words, complexity, more than anything else, is the problem for which microservices are a solution. **If complexity isn't your problem, microservices aren't the solution.**

Microservice architecture also brings with it significant overhead in terms of design, interoperability of services, management, and use of system resources. It has a price, and for applications which cannot make sufficient use of its advantages, the price may be too high.

Small Team, Big Job

Consider, for example, a moderately large, moderately complex

application being maintained by a relatively small development and operations team. If it's monolithic, the interactions between individual services may be very direct, and may be optimized for specific tasks as required. For a small development team familiar with the code, maintenance may be relatively simple. Deployment (as is often the case with monolithic applications) may be a bit awkward at times, but for the most part, it is manageable.

If the same team has to manage a microservice-based version of the same application, however, their overhead in terms of time and effort may increase significantly. Communication between microservices is more likely to be generic and protocol-based, requiring more coding time to implement even a minor change in data being handed from one service to another—and small design changes may require changes to the microservice orchestration and management system. This can put a strain on the resources of the operations team, as well as the developers.

Too Small to Break Down

Not all applications are large enough to break down into microservices. Even high-end, high-ticket desktop applications, for example, tend to be at least an order of magnitude too small for microservice architecture. A suite of applications consisting of small to medium-sized discrete services is probably already broken down as much as it needs to be, even if those discrete services contain multiple subordinate services.

Do your inventory module and your accounts payable module really need to be split up into microservices, or are they doing quite well as they are? Chances are very good that the scale at which they are currently operating is appropriate for your application. Decomposition into microservices would have the effect of adding rather than reducing complexity.

Living with a Legacy

For most software developers, legacy code isn't just a fact of life — it is their basic, day-to-day reality. If you are maintaining an application consisting largely of legacy code, no matter how haphazard its original design may have been, no matter how ugly it may be now, you need to think long and hard before refactoring it into microservices. At what point is it in its lifecycle? Does it serve a mission-critical function, such as maintaining an irreplaceable legacy database? Are you likely to be able to replace it entirely in a few years? Or, conversely, will the process of updating or replacing it require a well thought-out, long-term strategy?

Microservice architecture may very well play a major role in updating or replacing legacy software, but that process may be long, and in the meantime, a poorly conceived attempt at converting it to microservices with no overall strategy may turn out to be an avoidable disaster.

Lean and Tight

Some applications by their nature require tight integration between individual components and services. This is often true, for example, of applications that must process rapid streams of real-time data. Any added layers of communication between services may slow real-time processing down. When the system needs to respond quickly to data contained in that stream (for example, input from the sensors of a self-driving vehicle), delays could be catastrophic.

In fact, embedded applications in general operate under often very tight constraints, both in terms of response time and available resources, making them unlikely candidates for microservice architecture. It is typically important to design embedded applications for simplicity of operation and optimum use of resources right from the start. Microservices, on the other hand, are

largely a way of compensating for unavoidable complexity in a system where resources are not a major constraint.

Microservices or Not?

Are microservices right for your application? If it's very large, very complex, and in serious need of being tamed, then it's very possible that they are. But if it's doing okay as it is, then microservice architecture may be one bandwagon that you don't need to jump on.