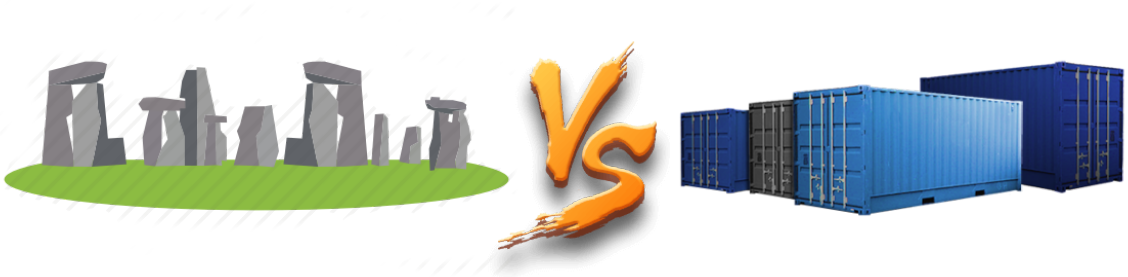
Microservices vs Monoliths: a new approach

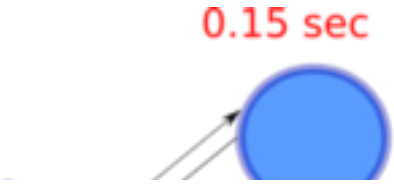
Alex Mosso Follow
Aug 10 · 4 min read ★

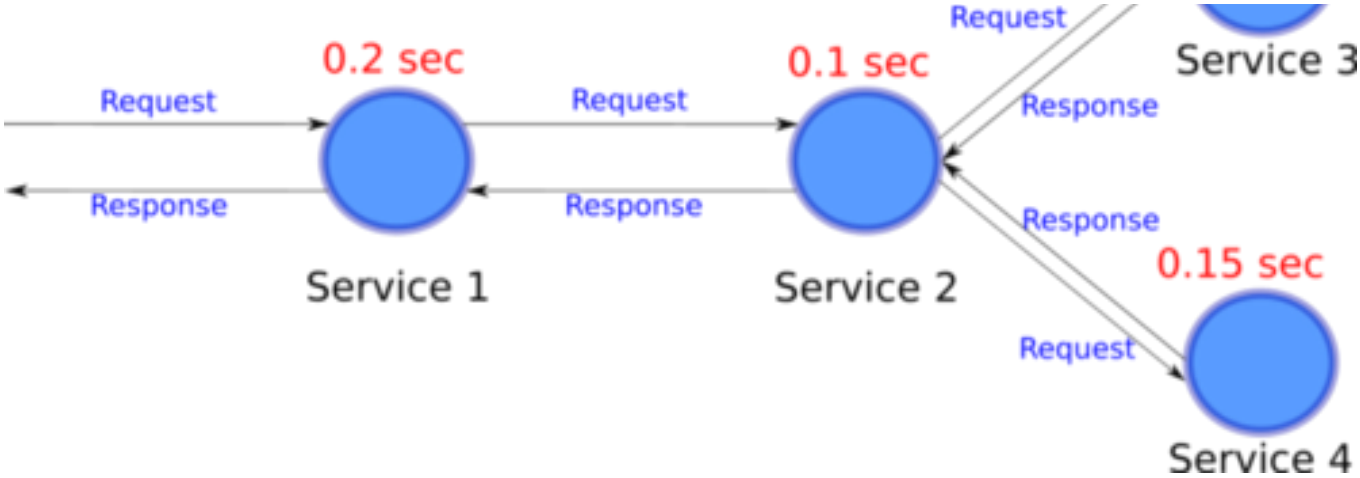


Many people who has developed SOA services think that microservices is the evolution of what they know about architecture and design patterns. But actually, this is a huge and very common mistake.

Actually, microservices theory states that traditional services have some issues by nature, therefore, it is required a new and better approach, and to make a distinction between microservices and the other approaches, we introduced the concept of monolith. The problem is that many developers think that the essential difference between traditional services and microservices is the size, because well, ones are known as a services and the others as micro-services.

Traditional services come with a problem, they are not well suited to sharing data. When we implement a request/response approach, the service will be locked until all nested requests are fulfill for a single message, and your CPU will be idle during the time the service is waiting for a response.





Apparently, this is not a big deal, but as you can see, Service 1 will be locked until all requests and responses are handled and dispatched, end even when we have several services to attend a request, the whole system is behaving as if it were a unit, in other words — as if it were a monolith.

Based on previous assumptions, now let's do a simple exercise — let's suppose the first Service in our system takes 0.2 seconds to fulfill, Service 2 takes 0.1 seconds, Service 3 and 4 take 0.15 seconds each one. When we trigger a call to Service 2, it will not really take 0.1 seconds, because it will have to wait for Service 3 and 4 to finish, therefor, Service 2 will take 0.4 seconds. It also means that Service 1 will take 0.6 seconds to return a response.

From my point of view, would be better if we just put all functionality in sequence inside a single service, because following this "distributed" approach you are injecting performance issues, also maintenance and deployment will be more complex.

Under this perspective, there are no relevant differences between a centralized multi-layer implementation and a service oriented architecture implementation — **both are monoliths**, because a single request to this kind of systems behaves as a unit.

Now the question is — do we really need monoliths? The short answer is — no — but at a first glance, many developers will think that there are some circumstances, or scenarios under which we will prefare to implement monoliths. If you really think like that, it is because you still don't make a mindset switch from monolith to microservices, because this is a change of paradigm. Once you have adopted microservices mindset,

you will realize that you can solve any kind of problem in any domain following microservices approach.

The essential difference

But, I still haven't said what is this microservice approach all about. First of all, you need to see a microservice system as a network, or as a DAG (Directed Acyclic Graph). **Actually, changing the way you see, enables you to do things in a different way too**. So to take all advantage of microservices, you should stop seeing them as the evolution of SOA, and start seeing them as a network system.

Once you make the switch, you will have new tools to design microservices. And under this new approach we can introduce new concepts like:

- Throughput: The amount of data we can get through a pipeline in a certain time.
- Pipeline: Is a data flow that continuously processes information.
- Pipeline parallelism: This is a side effect we get when we process data without interruptions nor locks in a pipeline.

The pipeline concept in microservices is the opposite to request-response in traditional services. When we approach microservices in this new way, we also have to face the throughput issues that come with this kind of implementations, and to solve throughput issues we need a broker to handle microservices communications, instead of sending direct messages through HTTP from one microservice to another.

A very good broker to handle microservices communication under this approach is Apache Kafka.

If you are not familiar with all these concepts, I will recommend below lecture:

Services come with a problem: they're not well suited to sharing data

Traditionally, developers use HTTP protocol to communicate services or micro-services. Nevertheless, there are some...

medium.com

Software Development Cloud Computing Microservices Cloud