The Goldilocks Calculation

How to Build Right Sized Application Services

By: Lee Atchison
Cloud Strategist
CEO, Atchison Technology LLC

Introduction

In the world of applications, services are standalone components that, when connected and working together, create an application that performs some business purpose. But services come in a wide variety of sizes, from tiny, super-specialized microservices up to services big and complete enough to form their own monolithic applications. The sizes vary significantly based on the application, and where the application came from and how it developed over time.

But, what's the ***right*** size for a service? If you are building an application from scratch, what size *should* you make your services?

What's the ***right*** size for a service?

Just as Goldilocks was looking for the perfect fit it's not always easy to determine the right size for the services that you need to build applications for your company and achieve your business objectives.

What is a service?

To be a service, a component must meet certain conditions. In my book, Architecting for Scale, High Availability for Your Growing Applications, I define a service by five criteria:

#1. **Maintains its own code base**.

A service has its own code base that is distinct from the other parts of your code base.

#2. **Manages its own data**.

A service manages its own distinct and separate datastore. It does not share data with other services or systems, other than through published APIs.

#3. **Provides capabilities to others**.

A service has a well-defined set of capabilities and it provides these capabilities to other services in your applications. In other words, it provides an API.

#4. **Consumes capabilities from others**.

A service uses well-defined sets of capabilities provided by other services and systems, and uses them in a standard, supported manner. In other words, it consumes other services' APIs.

#5. **Has a single owner**.

A service is owned and maintained by a single development team within your organization. This ownership is clear and well understood. When the service has a problem, the organization knows who needs to solve the problem.

How big should a service be?

Services can, realistically, be any size. Historically, services tended to be rather large and could form "mini applications" on their own. In fact, even a massive, monolithic application can be, to some extent, a service for other systems.

Nevertheless, there has been a trend in recent years towards making services smaller and smaller. The idea is that smaller services are more comprehensible and easier to manage, monitor and upgrade. A smaller group of individuals can own and manage a service, and those individuals' knowledge doesn't have to be as extensive. In fact, many of those smaller services can be grouped together and handled by a single development team.

While in theory this is a good idea, there is a fundamental problem with this "smaller is better" approach in practice. As services get smaller, each individual service tends to do less and less of what the overall application requires. This can result in a significant increase in the number of services necessary to build your applications. So even as the services are getting smaller, there are more and more services in play. Dealing with significantly more services often leads to a huge increase in the number of interactions between those services.

The net result? Using smaller services mean the interactions between services becomes significantly more complicated and harder to manage. Even though the individual services are easier to debug, the overall application itself can become harder to debug.

What service size is 'just right'?

Clearly, there's a point where the savings in complexity for individual services is outweighed by the increased complexity of the overall system architecture. Where this balancing point lies varies for every application, and depends on the maturity of three critical areas:

#1. The maturity of your service infrastructure.

Your services run on an infrastructure that your organization must provide. Generally speaking, the more mature the service infrastructure, the greater the number of services your system can maintain before becoming overwhelmed with complexity. This includes individual service infrastructure components such as deployment pipelines and service management tooling, along with shared infrastructure components such as inter-service communications channels, pipelines, and mechanisms.

#2. **The maturity of your individual development teams**.

The greater the maturity of your development teams, the larger and more complex the services they can maintain. The lower the development teams' maturity, the smaller you want your services to be.

#3. **The maturity of your overall application architecture team**.

The greater the maturity of your overall application architecture team, the greater the inter-service complexity that can be managed and maintained (and hence the larger number of smaller services that can be tolerated).

Smaller isn't necessarily better

Building services smaller and smaller has become a popular trend in modern application construction. In fact, it's so popular that they have even coined a name for this type of system: microservice architecture.

This trend to smaller services has gained significant momentum, and many people now advocate for making services as small as possible.

But smaller is ***not*** always better. The smaller the service, the more complicated the overall application typically becomes. This can be counterproductive to the simplification goal of service-oriented architectures.

Instead of building towards smaller and smaller services, your goal should be to build ***right-sized services***. Build services that are of an appropriate size consistent with the maturity of your service infrastructure, your development teams, and—most important—the maturity and sophistication of your application architecture team.

Otherwise, you risk ending up with microservice-based applications that are just as complex and difficult to manage and debug as those big monolith apps were.

Way Too Small

I once heard a service-based architecture that called for creating very small services as simple as a "TrueService" (a service that returns the answer "true" all the time) and a "FalseService" (a service that returns the answer "false" all the time), and utilizing these simple services in their overall system architecture. Other services used these, along with various logic services to perform actions, creating a needlessly very complex application.

About Atchison Technology LLC

Many corporate leaders are facing stagnant revenue and unhappy staff. They are facing aging technology and infrastructures. They are facing loss of customers.

Life is too short to be irrelevant

Competitive disruptors are taking business away from you. You are afraid of failing, of becoming irrelevant in your industry. You risk becoming irrelevant in your field.

It shouldn't be so hard to keep your competitive edge…or get it back again. I understand how hard transformation is to remain competitive, which is why I created Atchison Technology LLC.

I have 33 years experience working in this industry. I've led major transformation efforts at disruptors such as Amazon, AWS, and New Relic. I can help you with your transformation effort as well. I can help you stay relevant in your industry. I can help you *become* the disruptor.

Schedule a Call

If you want to find out more, schedule a call with me. We can discuss how I can help you in your modernization journey.

Schedule a Call

Or learn more by visiting me at:

www.atchisontechnology.com

For more application modernization ideas, please subscribe to our newsletter at:

atchisontechnology.com/subscribe