beSharp.
we make IT run.

# STATEFUL VS. STATELESS: THE GOOD, THE BAD AND THE UGLY.

Software as a Service (SaaS) | Statefull | Stateless

beSharp | 30 December 2019

---

# Stateful vs. Stateless: the good, the bad and the ugly

In the era of **SaaS services**, building **stateless microservices** seems to be the only way to achieve success. But is that true? And is it always possible to design a stateless service?

In this article, we will cover **the fundamental differences between stateful and stateless microservices.**

The key difference between stateful and stateless microservices is that **stateless** microservices **don't store data on the host**, whereas **stateful** microservices **require some kind of storage** on the host who serves the requests.

Keeping the state is critical for a stateful service. On the other hand, a stateless service can work using only pieces of information available in the request payload, or can acquire the required pieces of information from a dedicated stateful service, like a database.

Here a rapid overview of the main differences between a stateless and stateful service.

## Stateless

- The server processes requests based only on information relayed with each request and doesn't rely on information from earlier requests – this means that the server doesn't need to hold onto state information between requests (or the state can be held into an external service, like a database)
- Different requests can be processed by different servers

- The fact that any service instance can retrieve al service state necessary to execute a behavior from elsewhere enables resiliency, elasticity, and the ability for any available service instance to execute any task at all

## Stateful

- Stateful services are either a database or based on an internet protocol that needs a tight state handling on a single host
- The server processes requests based on the information relayed with each request and information stored from earlier requests
- The same server must be used to process all requests linked to the same state information, or the state information needs to be shared with all servers that need it

## The Challenges of running stateful workloads

There are multiple challenges related to running a stateful workload:

- Resource isolation – Many of the market's current container orchestration solutions still involve only a best-effort approach to resource allocation such as CPU, memory, and storage. This may work for stateless microservices, but when it comes to stateful ones, it can be a disastrous approach in which customer transactions or data are lost due to unreliable performance
- Persistent storage – Each stateful data service may need or support a different kind of storage type (for example block devices or distributed filesystems), and determining the type of backing storage for a stateful service can be challenging

These challenges are in part because many **stateful microservices were built for a legacy environment**, and are probably monolithic. Organizations may begin by attempting to containerize their stateful services, but then they need to develop highly specific tooling to coordinate numerous related instances for high availability or employ other sophisticated strategies to deploy, manage or operate these services. This can lead to manual overhead requirements, which can become time-consuming and costly and/or the need for the development of customized operation for every single service, which can bring with it considerable operational risk.

## What about SaaS?

SaaS is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted. It is sometimes referred to as "on-demand software". SaaS microservices are also known as Web-based software, on-demand software and hosted software.

The term "Software as a Service" (SaaS) is considered to be part of the nomenclature of cloud computing.

**Not all processes can be made stateless,** therefore, you can build a successful SaaS service either stateless or stateful.

Remember that a monolithic stateful service will probably be more expensive and difficult to maintain, and will make scaling much more difficult. Also, it will need special handling for backups and high availability.

## Conclusion

Sometimes you have to build a stateful service, this will not automatically harm your SaaS readiness. However, you will need to ensure some kind of scaling for your stateful services, and also plan for backups and rapid disaster recovery. While this is almost always possible, the effort may be much more than what is required to obtain better results on a stateless microservice.

Therefore, design your microservices stateless has a lot of advantages, especially when you need to scale automatically and when your usebase is big and geographically distributed across the globe.


## beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

### Get in touch

beSharp.it
proud2becloud@besharp.it