

# Beyond IaaS

This lecture walks you through the most important services offered by cloud providers beyond the IaaS layers. Sometimes this is called "PaaS" (Platform as a Service) indicating that it is intended as a developer platform.

The easiest way to remember the difference is that IaaS offers virtual machines and connected services. In order to operate on top of an IaaS platform you need someone skilled in running an operating system. In other words, you need a system administrator.

PaaS on the other hand is intended for developers. The platform services, on the other hand, are intended for developers. The main goal of PaaS services is enabling developers to deploy applications without having to manage the operating system of the underlying virtual machine.

There is, however, nothing preventing you from mixing IaaS and PaaS services. A typical use case would be using a managed database with virtual machines. This helps smaller teams because operating a database proficiently on a small scale can be an undue burden.

This might not seem like a big deal but consider that databases store data. Every time when data storage is concerned disaster recovery becomes more complex. If an IaaS team were to operate a database themselves they would need to regularly test backups and disaster recovery procedures. Managed databases take that complexity away.

Similarly, building a redundant database system that can perform an automatic failover requires a high skill level in managing that specific database engine. This skill level may not be available in small teams, or a small team may not want to spend time on managing the database instead of focusing on their core objective.

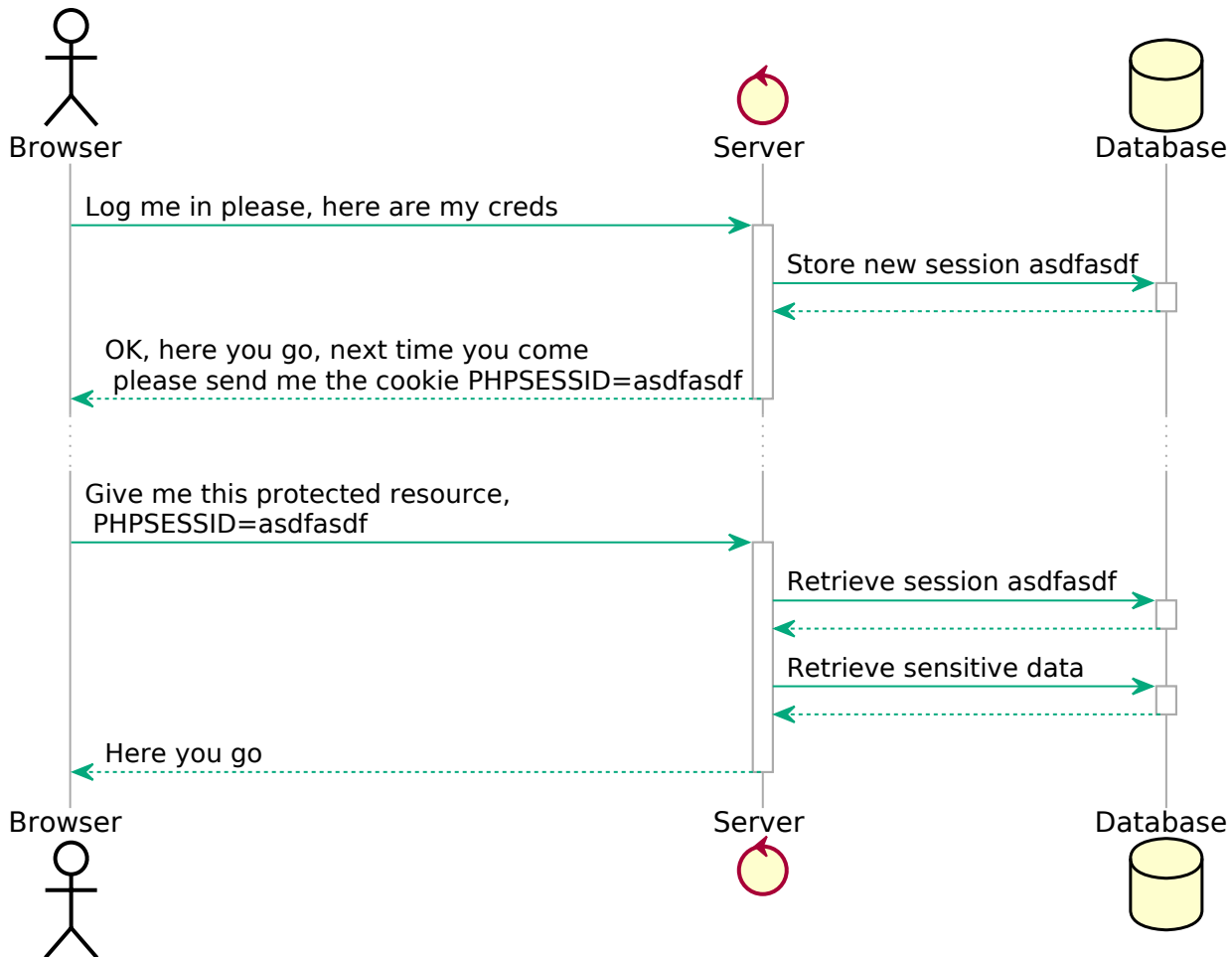
## Application load balancers

Application load balancers provide load balancing capabilities on [layer 7 of the OSI-model](#). In practice application load balancers offer support [HTTP](#) and HTTPS load balancing.

HTTP is a request-response protocol. Typically connections are short-lived but longer connections (e.g. with [Websockets](#)) are also possible. Good application load balancers allow sending traffic to a different set of backends based on the host name ( `example.com` ) as well as the path ( `/some/address` ).

Some applications also require that subsequent requests from the same client end up on the same backend. This is called session stickiness and can be achieved in a variety of ways. Less

advanced solutions route requests based on the source IP, while more advanced versions send a so-called [cookie](#) to the client and route subsequent requests based on that cookie.



Sticky sessions, however, present a problem: when a backend goes down the users of that backend will be redistributed to other backends. In practice this usually means that users will be logged out. When a cascading fault, or a rolling update occurs that takes down multiple backends in rapid succession users can be subjected to multiple involuntary logouts.

This has an adverse effect on user experience which is why newer, so-called “cloud native” applications don't use sticky sessions. Instead, cloud native applications put client-specific data (e.g. [session data](#)) in database systems with redundancy. Sessions themselves [have their own race condition problems](#), but that is not a discussion for this lecture.

CDNs

Object Storage

**Cold storage**

Databases as a Service (DBaaS)

**Relational databases (SQL)**

**Document databases**

**Time Series databases**

Functions as a Service (FaaS / Lambda)

Containers as a Service (CaaS)

Stream processing

Deployment pipelines