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.

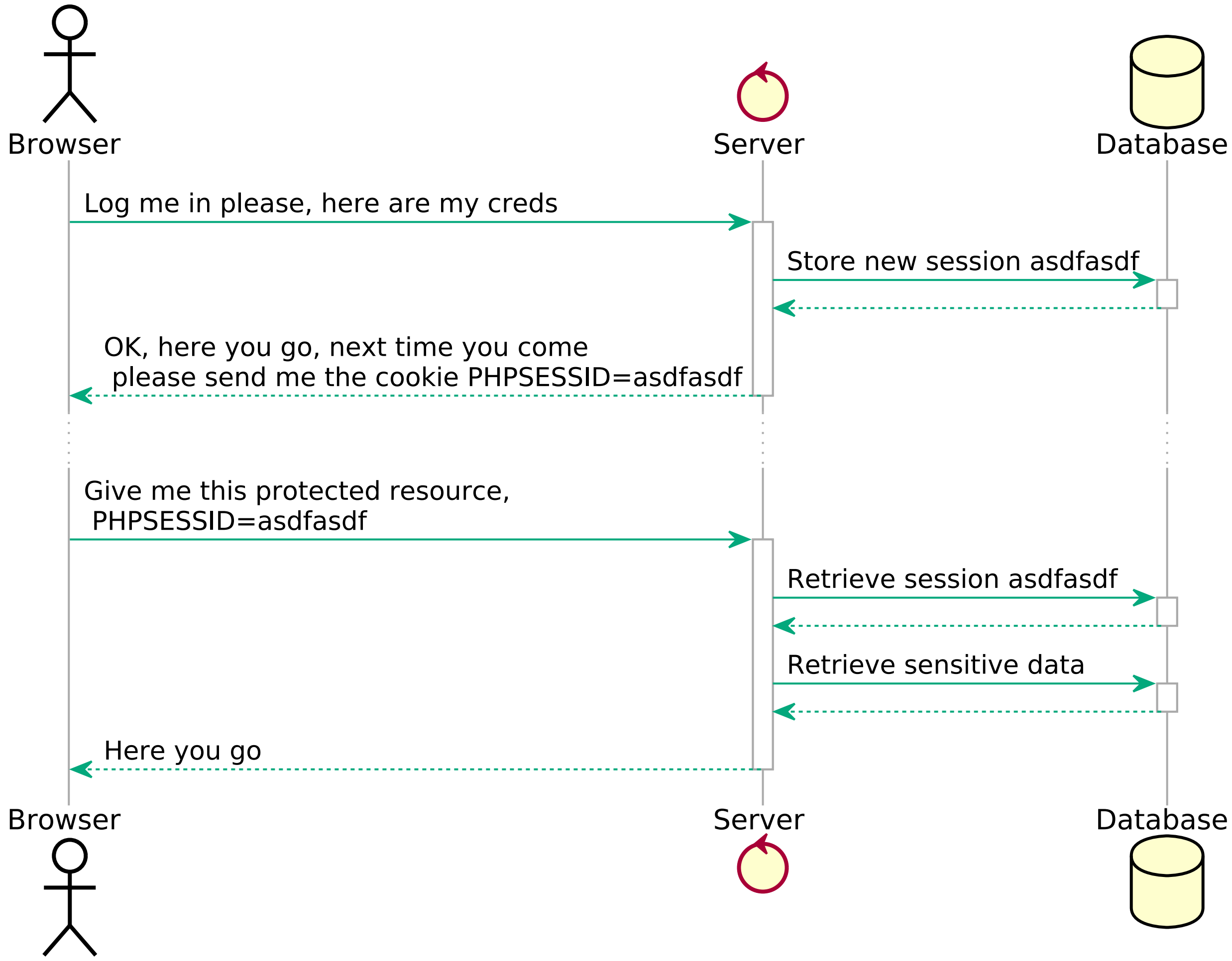
Similarly, building a redundant database system that can perform an automatic failover requires a high skill level in managing that specific database engine. This skill level may not be available in small teams, or a small team may not want to spend time on managing the database instead of focusing on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

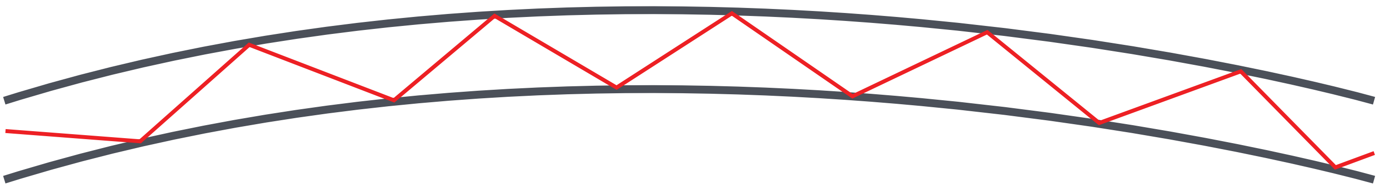
This has an adverse effect on user experience which is why newer, so-called “cloud native” applications don't use sticky sessions. Instead, cloud native applications put client-specific data (e.g. session data) in database systems with redundancy. Sessions themselves have their own race condition problems,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

CDNs

While it seems the Internet is blazing fast nowadays delivering content to the other side of the planet is still an issue. As you may know most of the Internet is comprised of fiber optic cabling. Data is transmitted by turning a laser on and off in rapid succession.

Let's do a little mental exercise: the speed of light is 299.792.458 m/s. The radius of our planet is 6.371 km. A ray of light should be able to round the planet in `6.371.000 / 299.792.458 =`

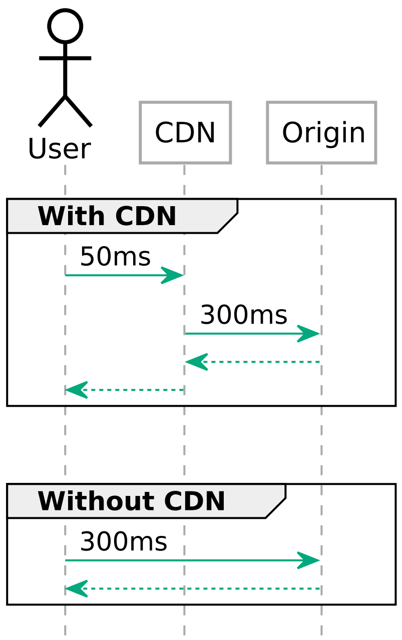
`0.021` seconds. In other words any data transmitted should be able to travel around the world in ~21 milliseconds. Yet, in practice we see latencies upwards of 100 ms when transmitting data from Europe to India.



The issue is that light does not go “around the corner”. Fiber optic cables work by having reflective walls so the light bounces off the walls repeatedly to reach the end. This multiplies the way light has to travel to reach the other side of the planet. This is further compounded by the fact that light can only travel so far in a fiber optic cable, repeaters and routers in between also need time to process the data.

The problem is further compounded by how HTTP works. When a website is loaded there are several elements that have to wait for each other. The website may reference a style sheet (CSS file), the CSS file may reference an image and so on. This means several round trips have to be done to build a page. HTTP/2 server push attempts to mitigate this.

CDN's work around this problem by replicating content to servers closer to the users. When a user requests a certain file that request lands on the CDN's *edge node* first. If the CDN has the file in the cache the delivery is very fast. If, however, the CDN does not have the content in cache the delivery will take longer.



In essence, CDN's help with latency issues *if the content can be cached*. in other words, this helps mostly with static content, but in combination with functions as a service (see below) dynamic content can also be delivered from edge nodes to some extent.

Did you know?

Low latency connections are important not just for delivering content. Stock exchanges benefit from very low latencies. For example, there is a private microwave network between Frankfurt and London that has twice the speed of the public internet.

Did you know?

SpaceX is building the Starlink network to provide lower latency connectivity across the globe.

Object Storage

In the previous lecture we briefly mentioned object storages. As a reminder, traditional block storage devices and the filesystems implemented on top of them have several features you may consider advanced:

- Multiple programs can open the same file in parallel,
- Files can be read and written partially,
- Locks can be placed on files preventing other processes from accessing them.

Object storages are different. The most popular Amazon S3 protocol offers the ability to up- and download, list and delete files. However, files can only be uploaded as a whole, partial reads or writes are not possible. Consistency is also not guaranteed when multiple programs are accessing the same files in parallel.

However, due to the limited featureset object storages have a few unique abilities not afforded by traditional block storage:

- They are redundant over multiple servers by design. The loss of a single physical machine does not mean a data loss.
- The S3 protocol offers the ability to place ACL's (access control lists) on the objects uploaded to the object storage. This allows making files publicly accessible over the web without needing to maintain a server.
- Some object storage implementations offer the ability to keep *multiple versions* of files. Clients can be prevented from deleting older versions making versioning an effective data loss prevention mechanism.
- Some object storage implementations offer the ability to lock files from being modified in the future. This is especially important when adhering to corporate or government data retention requirements.

Cold storage

Some providers offer an extension to their object storage system that puts data in cold storage (e.g. on tape). Data can be uploaded directly via the API, or in the case of very large data amounts shipped to the provider on hard drives.

> **Did you know?**
>
> "Never underestimate the bandwidth of a truck full of hard drives." — is an industry saying that kept its validity to this day.

Since the data is stored on offline (cold) storage the data retrieval is not as immediate as with the object storage. To retrieve data from cold storage you often need to wait several hours until the data becomes available. Therefore, an effective backup strategy to the cloud often involves moving data to the object storage first and only older backups to cold storage. Amazon, for example, allows for automating this process using S3 lifecycle rules.

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