Caching

What is Caching?

In computing, a cache is a **high-speed data storage** layer that stores a **subset** of data.
Caching allows you to efficiently reuse previously retrieved or computed data.

A cache's primary purpose is to increase **data retrieval** performance by reducing the need to access the underlying slower storage layer.

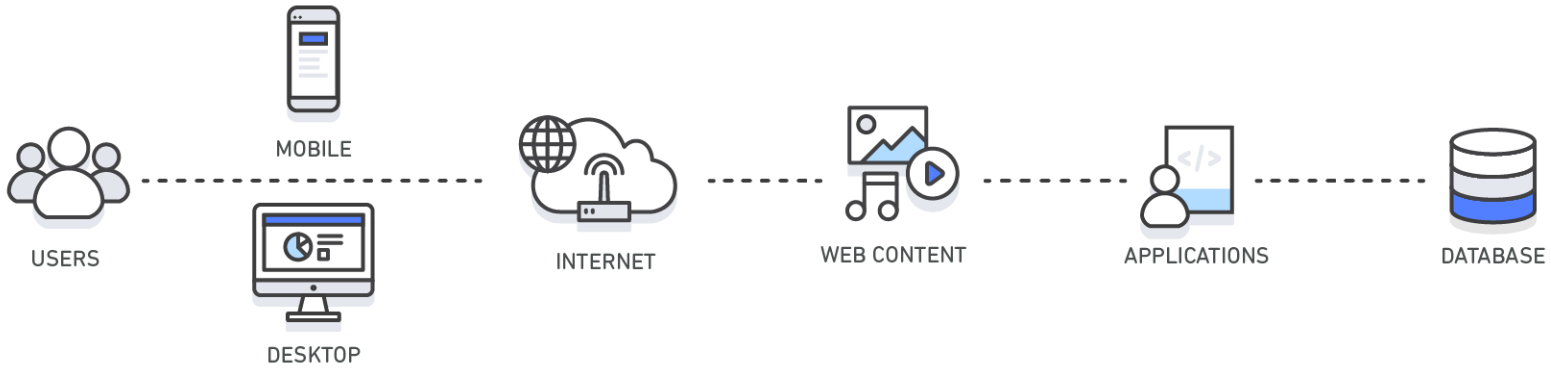
Trading off capacity for speed, a cache typically stores a subset of data temporarily, in contrast to databases whose data is usually complete and durable.

Caching Best Practices

When implementing a cache layer, it's important to understand the **validity** of the data being cached. A successful cache results in a **high hit rate** which means the data was present when fetched. A **cache miss** occurs when the data fetched was not present in the cache. Controls such as **TTLs (Time to live)** can be applied to expire the data accordingly.

Another consideration may be whether or not the cache environment needs to be **Highly Available**. In some cases, a cache environment can be used as a **standalone** data storage layer in contrast to caching data from a primary location. In this scenario, it's important to define an appropriate **RTO** (Recovery Time Objective--**the time it takes to recover** from an outage) and **RPO** (Recovery Point Objective--the **last point or transaction** captured **in the recovery**) on the data resident in the cache environment to determine whether or not this is suitable.

Caching Use Cases



Web Caching

Web caching is performed by retaining **HTTP responses and web resources** in the cache for the purpose of fulfilling future requests from cache rather than from the origin servers.

Various web caching techniques can be employed both on the **server** and on the **client** side.

Reverse proxy* cache can be placed in front of application and web servers in order to serve a cached version of the **HTTP responses** retained from them. Another form of server side web caching includes utilizing **key/value stores** such as Memcached and Redis, they can be used to cache **any web conten**t desired.

Client side web caching can include **browser based caching** which retains a cached version of the previously visited web content.

Database Caching

Caching can be applied to any type of database including **relational** databases or **NoSQL** databases. The best part of caching is that it's minimally invasive to implement and by doing so, your application performance regarding both **scale** and **speed** is dramatically improved.

The cache itself can live in a number of areas including your database, application or as a standalone layer:

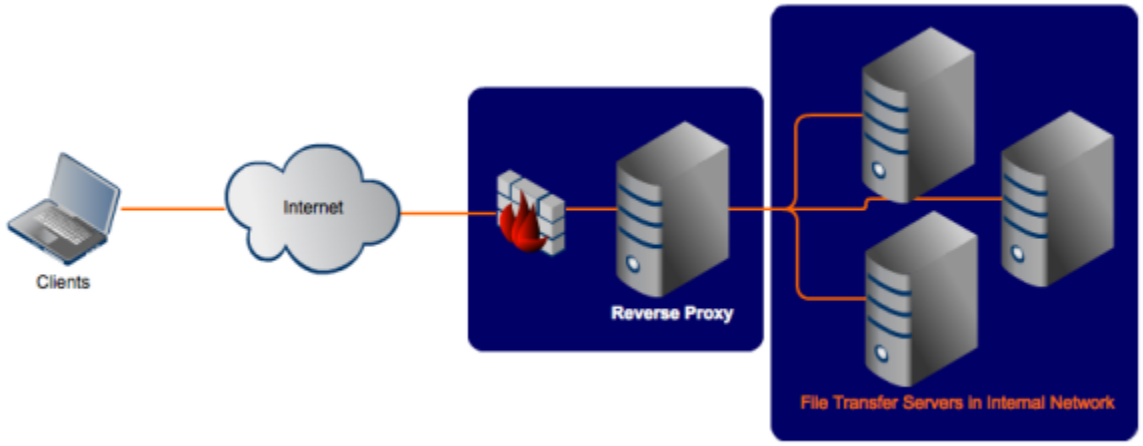
- **Database Integrated Caches**
  Some databases offer an integrated cache that is managed within the database engine and has built-in write-through* capabilities.
- **Local Caches**
  A local cache stores your frequently used data within your application.
- **Remote caches**
  Remote caches are stored on dedicated servers and typically built upon key/value NoSQL stores such as Redis and Memcached.

Application Programming Interfaces (APIs) Caching

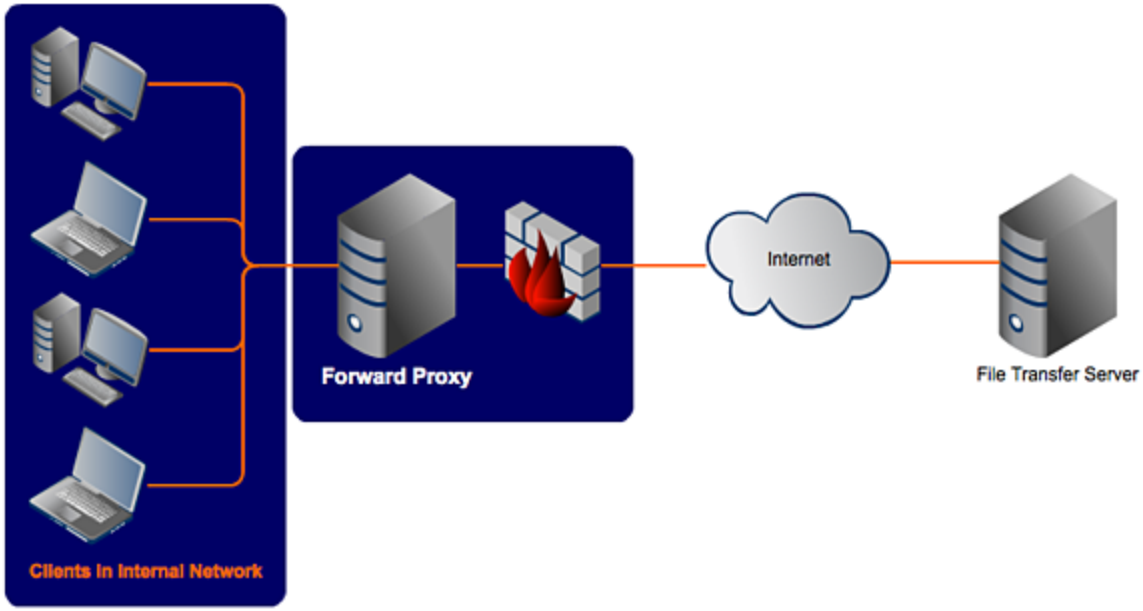
For example, you exposed a product listing API to your users and your product categories only change once per day. Given that the response to a product category request will be identical throughout the day every time a call to your API is made, it would be sufficient to cache your API response for the day.

Terms

- Reverse Proxy: A reverse proxy is the application that sits in front of back-end applications and forwards client (e.g. browser) requests to those applications. Reverse proxies help balance the load between internal **servers**, keep a cache, and add features such as compression or TLS encryption.



- Forward Proxy: While a reverse proxy proxies on behalf of servers, a forward proxy proxies on behalf of clients (or requesting hosts),.



- Write-through Caching: When data is updated, it is written to both the cache and the back-end storage. This mode is easy for operation but is slow in data writing because data has to be written to both the cache and the storage.

- Write-back Caching: When data is updated, it is written only to the cache. The modified data is written to the back-end storage only when data is removed from the cache. This mode has fast data write speed but data will be lost if a power failure occurs before the updated data is written to the storage.

References*

https://aws.amazon.com/caching/
https://aws.amazon.com/caching/database-caching/
https://stackoverflow.com/questions/27087912/write-back-vs-write-through-caching

https://en.wikipedia.org/wiki/Reverse_proxy
https://www.jscape.com/blog/bid/87783/forward-proxy-vs-reverse-proxy