Load Balancing

Overview

Notes

scale up | scale out

_ Load Balancer (Network / Application)

/ \

software hardware

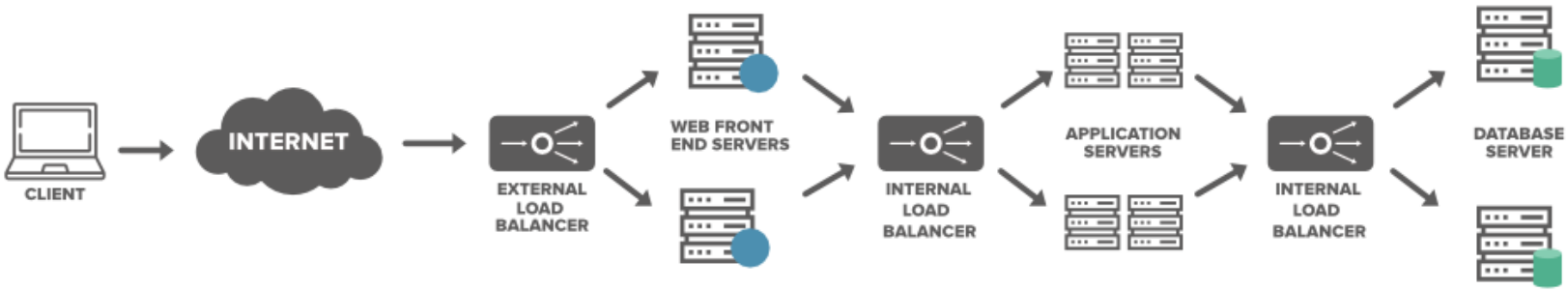
/ \

client server

Goal: scale, availability and security
Algorithms: dynamic, static
In cloud

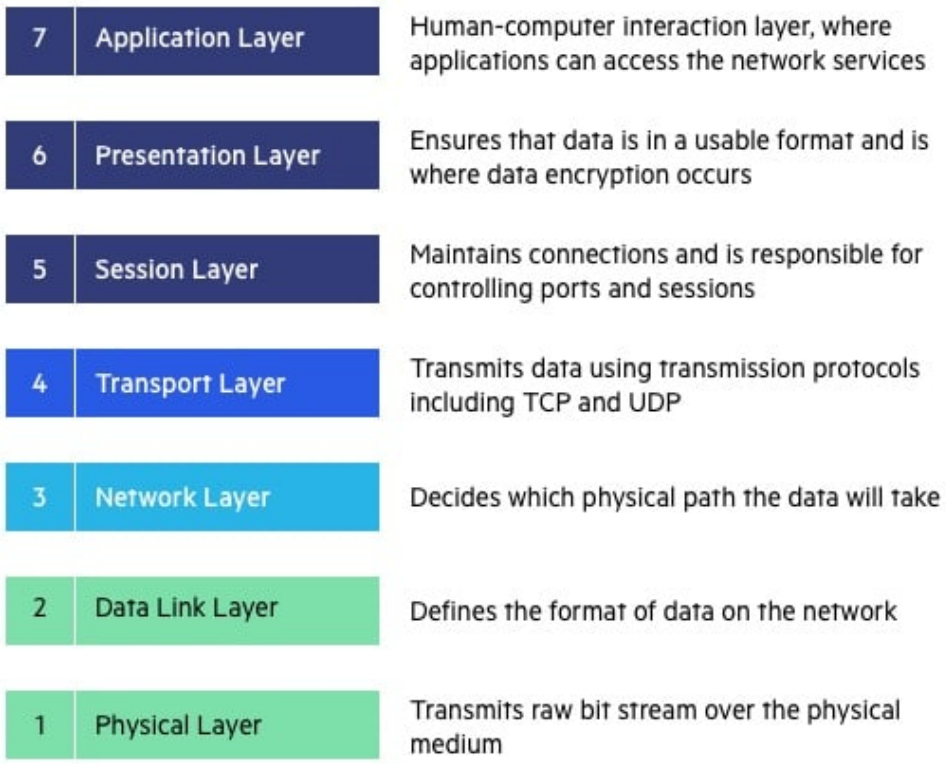
Load balancing is the process of distributing a set of **tasks** over a set of **resources**, with the aim of making their overall processing more **efficient**.

A load balancer can be placed:



- In between the **client** application/user and the **server**
- In between the **server** and the **application**/job servers
- In between the **application** servers and the **cache** servers
- In between the **cache** servers the **database** servers

Load balancers can be software-based or hardware-based. They are generally grouped into two categories: Layer 4 and Layer 7.



Load Balancers Categories

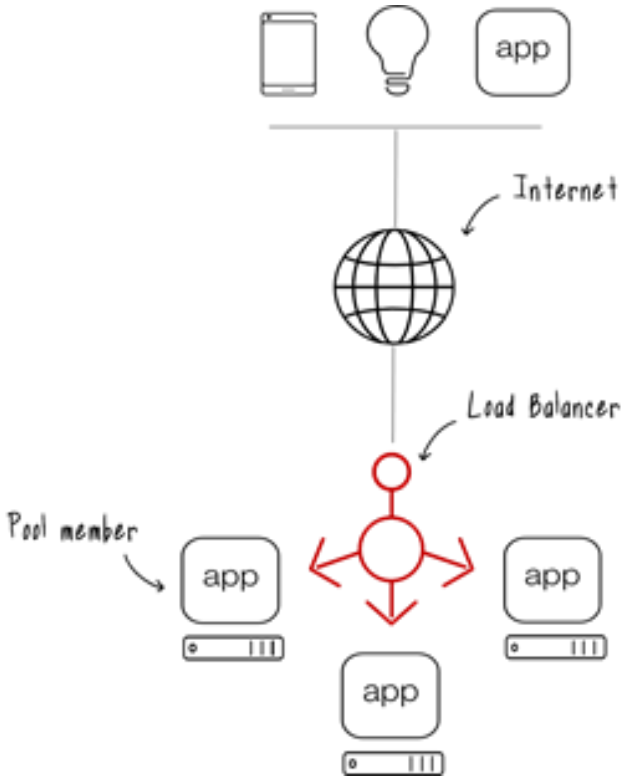
Layer 4 / Network Load Balancers

Layer 4 load balancers act upon data found in network and transport layer protocols such as IP, TCP and UDP. a Layer 4 load balancer bases the load-balancing decision on the **source and destination IP addresses and ports** recorded in the packet header, without considering the contents of the packet.

When the Layer 4 load balancer receives a request and makes the load balancing decision, it also performs **Network Address Translation (NAT)** on the **request** packet, changing the recorded destination IP address from its own to that of the **content server** it has chosen on the internal network. Similarly, before forwarding server **responses** to clients, the load balancer changes the source address recorded in the packet header from the server's IP address to its own.

Layer 7 / Application Load Balancers

Layer 7 load balancers operate at the highest level in the OSI model, the application layer (on the Internet, HTTP is the dominant protocol at this layer). Layer 7 load balancers base their routing decisions on various characteristics of the **HTTP header** and on the actual **contents** of the message, such as the URL, the type of data (text, video, graphics), or information in a cookie.



Secure Sockets Layer (SSL) is the standard security technology for establishing **an encrypted link** between a web server and a browser. SSL traffic is often **decrypted at the load balancer**. When a load balancer decrypts traffic before passing the request on, it is called **SSL termination**. The load balancer saves the web servers from having to expend the extra CPU cycles required for decryption. This improves application **performance**.

However, SSL termination comes with a **security concern**. The traffic between the load balancers and the web servers is no longer encrypted. This can expose the application to possible attacks. However, the risk is lessened when the load balancer is **within the same data center** as the web servers.

Another solution is the **SSL pass-through**. The load balancer merely passes an encrypted request to the web server. Then the web **server** does the **decryption**. This uses more CPU power on the web server. But organizations that require extra **security** may find the extra overhead worthwhile.

Load Balancing Algorithm

Dynamic

According to the current state of each server

- (Weighted) Least Connection Method
  directs traffic to the server with the fewest active connections
- (Weighted) Least Response Time Method
  directs traffic to the server with the fewest active connections and the lowest average response time.
- Resource-based
  based on what resources each server has available at the time.
  Specialized software (called an "**agent**") running **on each server** measures that server's **available** CPU and memory, and the **load balancer queries the agent** before distributing traffic to that server.

Static

Not dynamic

- (Weighted) Round Robin
  rotates servers by directing traffic to the first available server and then moves that server to the bottom of the queue (sequentially)
- Hashing

A set of information, e.g. the IP address of the client, is used to determine which server receives the request.

Session Affinity

Session affinity, or sticky sessions, are a load balancer feature for **stateful** services. With sticky sessions, the load balancer sends all **requests from the same client** to the same **service** instance. This enables the service to maintain in-memory state about each specific client session.

Load imbalance occurs because client sessions last for varying amounts of time. Even if sessions are evenly distributed initially, some will terminate quickly while others will persist. To help alleviate load imbalance, load balancers usually provide policies such as sending new sessions to instances with the least connections or fastest response times. These help direct new sessions away from heavily loaded services.

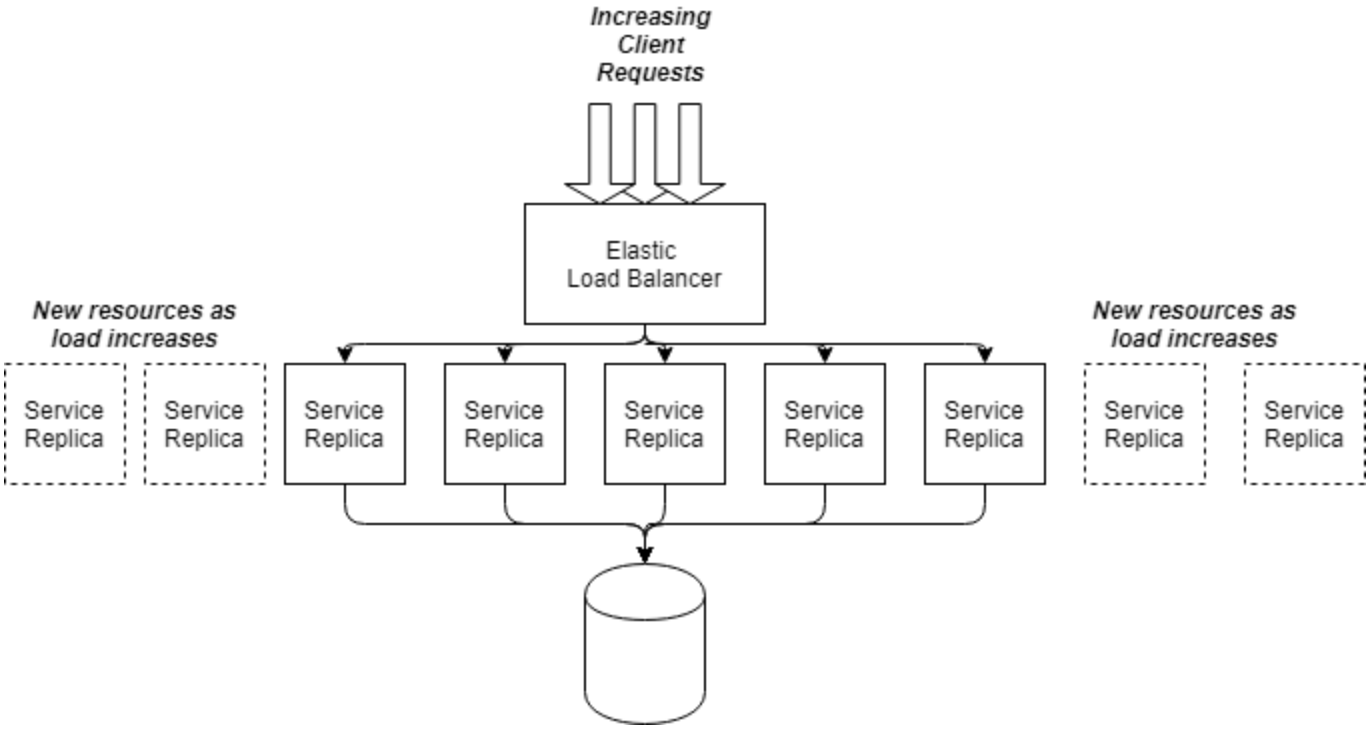
Cloud Considerations for Load Balancing

- Scale Infrastructure

To take full advantage of the cloud your application infrastructure should **increase and decrease capacity automatically** to match the demand of your users. Your load balancer must be able to **register and deregister nodes** from load balancing pools through **automation**.

That requires a load balancer to be tightly integrated with application **monitoring**, so that scaling policies can be defined to determine when to scale up and down. Policies may specify for example that capacity for a service should be increased when the average service CPU utilization across all instances is over 70%, and decreased when average CPU utilization is below 40%.

An example of that is the Amazon Web Services (AWS) Auto-Scaling groups. An Auto Scaling group is a collection of service instances available to a load balancer that is defined with a minimum and maximum size. The load balancer will ensure the group always has the minimum numbers of services available, and the group will never exceed the maximum number.



- Scale Load Balancers

Load balancers are often the front door of your application—they need to be highly available and always accepting connections. We should auto scale them to match the capacity you need when you need it, and release that capacity when you don't.

The load balancing algorithm being used by your solution should also be considered. It's important to understand how adding or removing a node from the pool will redistribute load.

- Health Check

Your application stack will experience issues, employing your load balancer to health check the application and only pass traffic to healthy nodes ensures that your end users see as little interruption as possible.

Health checks should be **configurable** for the request type, response, and timeout. With these settings you can ensure that your application is responding correctly, and in a reasonable amount of time.

*References

https://en.wikipedia.org/wiki/Load_balancing_(computing)
https://www.nginx.com/resources/glossary/
https://avinetworks.com/what-is-load-balancing/
https://docs.aws.amazon.com/elasticloadbalancing/index.html
https://www.f5.com/services/resources/glossary/load-balancer
https://www.imperva.com/learn/application-security/osi-model/
https://www.cloudflare.com/learning/performance/types-of-load-balancing-algorithms/
https://learning.oreilly.com/library/view/load-balancing-in/9781492053934/
https://www.geeksforgeeks.org/load-balancer-system-design-interview-question/
Load Balancing in the Cloud By Derek DeJonghe
Foundations of Scalable Systems By Ian Gorton