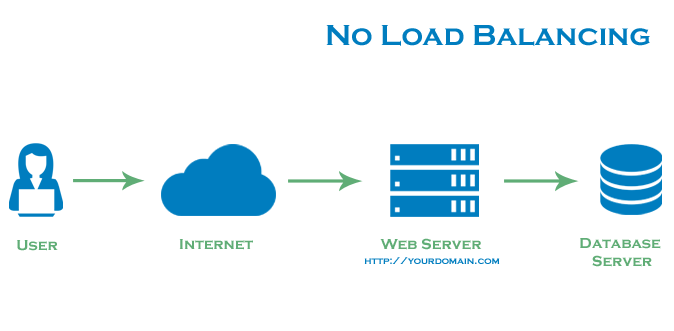
# Load Balancing

### **No Load Balancing**

The name says it all; without load balancing it’s just a simple web application environment. The following diagram will help you to understand it in a right way:

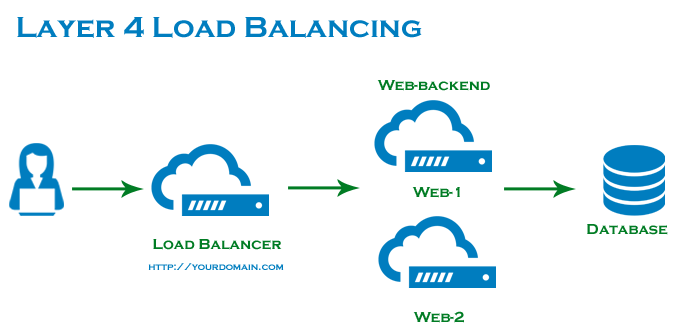


The above diagram denotes, the user connects straight to the web server i.e. domain.com and there is no presence of load balancing. This mean, in case the web server goes offline due to some reasons; the end user will not be able to access it. In another scenario, if there are multiple users trying to access the web server simultaneously and web server shows its limitation to handle the load, the end users generally experience the slower response or might not able to connect to the server at all.

### **Layer 4 Load Balancing**

Layer 4 load balancing (which is also called as transport layer load balancing) is widely acknowledged for its simple way to load balance the network traffic through multiple servers. This type is based on IP range/domain and port i.e. if user request comes in for domain.com/blog, the traffic will be sent to the backend that manages all the user requests for domain.com on port 80.

Check the below diagram to see a simple example of layer 4 load balancing:

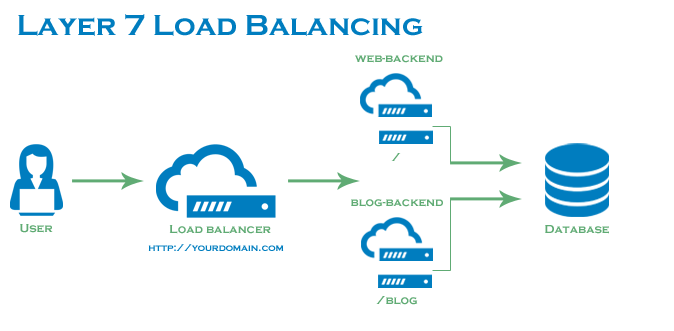


In this architecture, the user first connects to the load balancer and then the user request is sent to the web servers. The selected web server responds directly to the user request immediately. Usually, all the web servers contain the similar data which avoid sending back inconsistent content to the user. Remember, all the web servers connect to the similar database server.

### **Layer 7 Load Balancing**

Also known as application layer load balancing is more refined and sophisticated way of network traffic load balancing than Layer 4. This mode is based on the content of the user’s request in which load balancer send user request to the web servers according to the content of a request. This is the very advantageous way because users can run multiple web servers on the same domain and port.

Check the below diagram to see a simple example of layer 7 load balancing:



**The Frontend configuration example:**

frontend http

bind \*:80

mode http

acl url\_blog path\_beg /blog

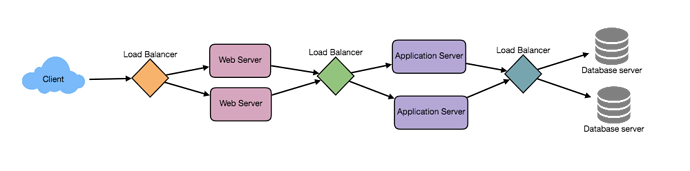
use\_backend blog-backend if url\_blog

default\_backend web-backend

1. “Acl url\_blog path\_beg/blog” meet a request if the path of the user’s request starts with /blog
2. “Use\_backend blog-backend if url\_blog” uses the ACL to proxy the traffic to blog-backend
3. “Default\_backend web-backend” identifies that all other traffic will be forwarded to web-backend

To utilize full scalability and redundancy, we can try to balance the load at each layer of the system. We can add LBs at three places:

* Between the user and the web server
* Between web servers and an internal platform layer, like application servers or cache servers
* Between internal platform layer and database.



**Load Balancing Algorithms**

The algorithm plays a key role to define which server will take place during load balancing. And HAProxy offers plenty of options for algorithm. Furthermore, servers can be allocated with a *weight parameter* to handle how often the server is chosen as compared to other servers.

HAProxy offers numerous algorithm options, but we will cover a few of them only, as describing all of them will make this post like a Ph.D. thesis :). You should visit [http://cbonte.github.io/haproxy-dconv/index.html](https://cbonte.github.io/haproxy-dconv/index.html) to see the configuration manual for a complete list of algorithms.

**Anyway, let’s see the regularly used algorithms:**

* **Algorithms**
  + **Weights parameter**
  + **Round Robin**
    - This is the default algorithm that selects the servers in a rotational way.
  + **Leastconn**
    - Minimum numbers of connections are considered here for the server selection. For longer sessions, this algorithm is highly recommended. In the same way, backend servers turn in a round-robin way.
  + **Source**
    - The server selection is based on the hash of the source IP address i.e. user’s IP. This makes sure that the user will connect to the matching server.

### **Sticky Sessions**

Certain applications demand that a user continues to connect to the similar backend server. This tenacity is accomplished via sticky sessions, using the appsession parameter in the backend that requires it.

### **Health Check**

The health check checks whether backend is vacant to route requests or not, this means there is no manual intervention to remove an unavailable backend. The default health check is to try to launch TCP connection to check the availability of the backend servers.

If in case a server fails a health check, and for that reason, it is unable to respond to the server requests, it is by default gets disabled in the backend. This mean, traffic will not be sent to backend server till it turns out to be healthy again. If we consider the worst situation where all backend servers fail, then the service will become unavailable till at least one of them turns back up again…

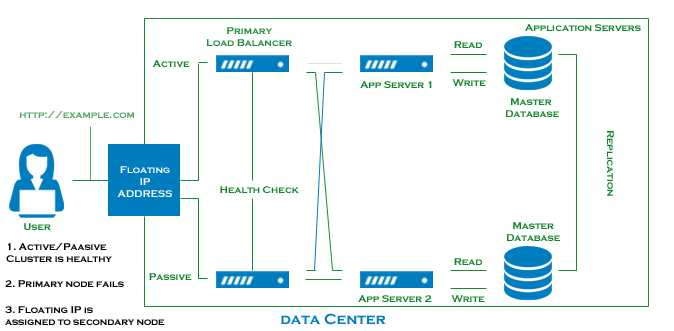
However, the default health check is not enough in some types of backends such as database servers in certain conditions to figure out whether the server is still healthy or not.

### **High Availability**

As mentioned above the layer 4 and layer 7 load balancing works based on the use of a load balancer to direct traffic to one of many backend servers. In both these types of load balancers is a single point of failure so if main load balancer goes down due to some reasons or it gets overcrowded with requests, the end user experience the high latency or downtime for the service.

To overcome this issue a high-availability configuration is mostly preferred because it eliminates a single point of failure. By adding redundancy to each layer of your architecture, HA configuration prevents a single server failure point. Basically, load balancer enables redundancy to the backend servers, but to achieve a real high availability there is a need of redundant load balancers as well.

The following diagram will help you to understand a basic high availability setup:



The above diagram clearly describes the functionality of multiple load balancers. In this diagram, 2 load balancers are in the action; one is active and second is in passive mode (you can add more load balancers in passive mode) behind a static IP address, which can be remapped from one server to another. In this kind of setup, if one load balancer fails, the failover mechanism immediately identify it and automatically reallocate the IP address to one of the servers in passive mode. This is just one of the ways to implement active/passive high availability setup; however, there are many different techniques.

### **implementation**

There are many ways to implement load balancing.

1. Smart Clients

One way to implement load-balancing is through the client applications. Developers can add the load balancing algorithm to the application or the database client. Such a client will take a pool of service hosts and balances load across them. It also detects hosts that are not responding to avoid sending requests their way. Smart clients also have to discover recovered hosts, deal with adding new hosts, etc. Smart clients look easy to implement and manage especially when the system is not large, but as the system grows, LBs need to be evolved into standalone servers.

1. Hardware Load Balancers

The most expensive–but very high performance–solution to load balancing is to buy a dedicated hardware load balancer (like a [Citrix NetScaler 2](https://en.wikipedia.org/wiki/NetScaler)). While they can solve a remarkable range of problems, hardware solutions are costly, and they are not trivial to configure.

As such, even large companies with large budgets will often avoid using dedicated hardware for all their load-balancing needs. Instead, they use them only as the first point of contact for user requests to their infrastructure and use other mechanisms (smart clients or the hybrid approach discussed in the next section) for load-balancing for traffic within their network.

1. Software Load Balancers

If we want to avoid the pain of creating a smart client, and since purchasing dedicated hardware is expensive, we can adopt a hybrid approach, called software load-balancers.

[HAProxy 5](https://en.wikipedia.org/wiki/HAProxy) is one of the popular open source software LB. The load balancer can be placed between the client and the server or between two server-side layers. If we can control the machine where the client is running, HAProxy could be running on the same machine. Each service we want to load balance can have a locally bound port (e.g., localhost:9000) on that machine, and the client will use this port to connect to the server. This port is, actually, managed by HAProxy; every client request on this port will be received by the proxy and then passed to the backend service in an efficient way (distributing load). If we can’t manage the client’s machine, HAProxy can run on an intermediate server. Similarly, we can have proxies running between different server-side components. HAProxy manages health checks and will remove or add servers to those pools. It also balances requests across all the servers in those pools.

For most systems, we should start with a software load balancer and move to smart clients or hardware load balancing as the need arises.

### **Conclusion**

These two articles depict the fundamentals of load balancing and HAProxy, which will help you to understand how to increase the performance and reliability of your crucial server environment. If in case you have any further questions regarding HAProxy and Load Balancing, don’t hesitate to ask them in the comment section.