**Introduction to Load Balancing**

In this lesson, you will understand load balancing and the need for it in web applications.

**We'll cover the following**

* + [What is load balancing?](https://www.educative.io/module/lesson/web-application-architecture-101/R8zN8YoxD8R#What-is-load-balancing?)
  + [Performing health checks of the servers with load balancers](https://www.educative.io/module/lesson/web-application-architecture-101/R8zN8YoxD8R#Performing-health-checks-of-the-servers-with-load-balancers)

**What is load balancing?**[#](https://www.educative.io/module/lesson/web-application-architecture-101/R8zN8YoxD8R#What-is-load-balancing?)

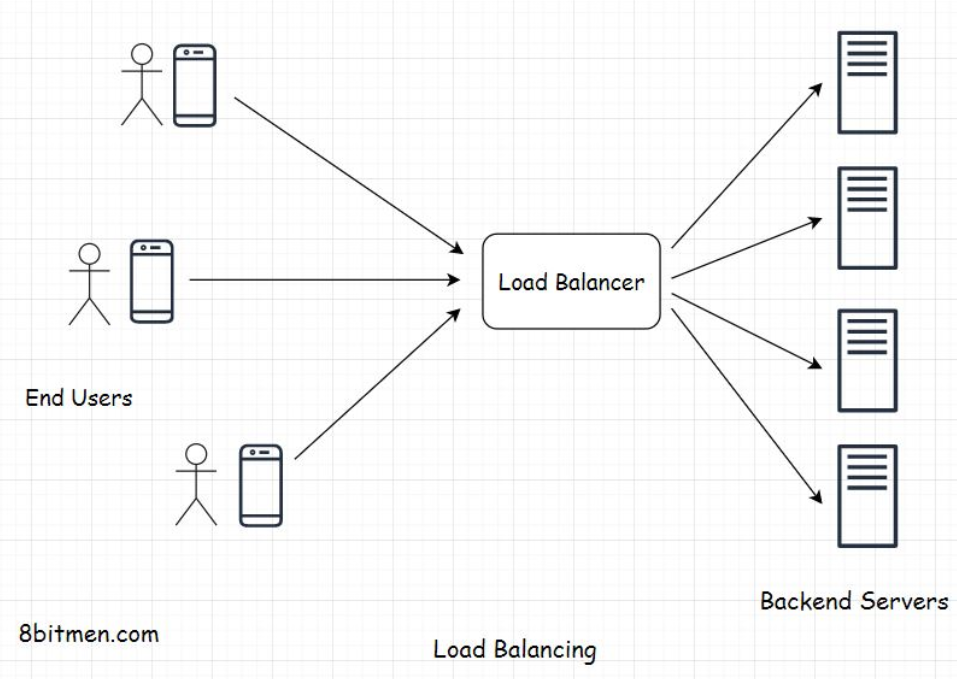
*Load balancing* enables our service to scale well and stay highly available when the traffic load increases. Load balancing is facilitated by *load balancers*, making them a key component in the web application architecture.

Load balancers distribute heavy traffic load across the servers running in the cluster based on several different algorithms. This averts the risk of all the traffic converging to a single or a few machines in the cluster.

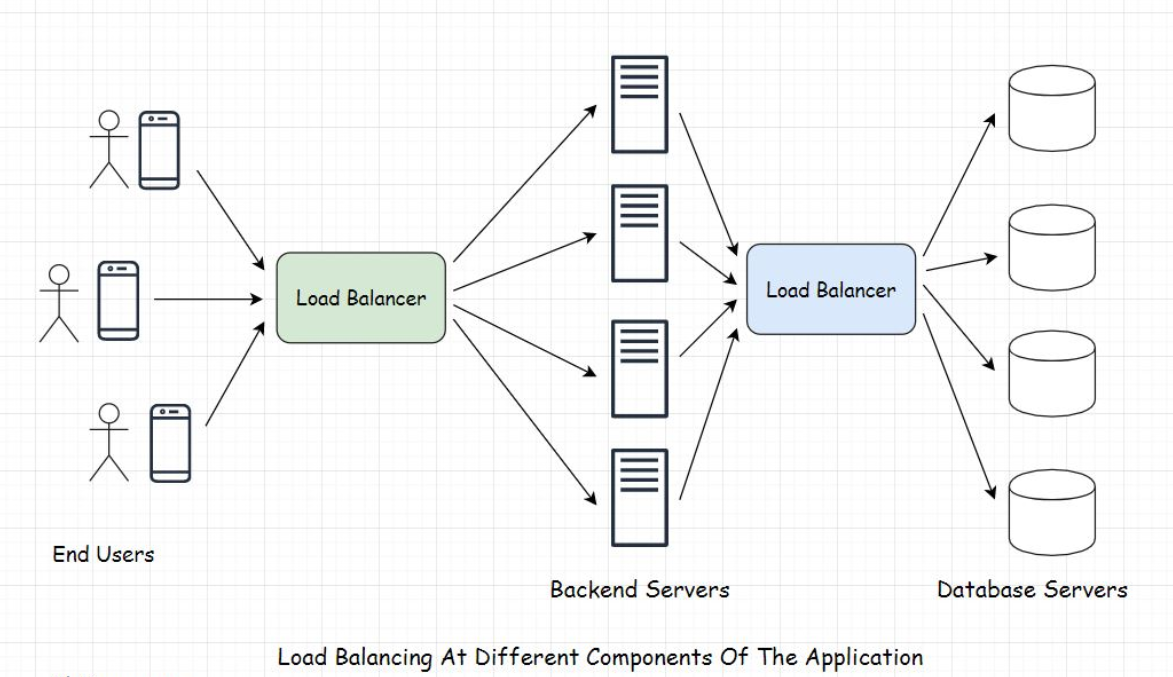
If the entire traffic on a particular service converges only to a few machines, this will cause an overload, subsequently spiking the latency, also bringing down the application’s performance. The excess traffic might also result in the server nodes going offline. Load balancing helps us avoid all this mess.

While processing a user request, if a server goes down, the load balancer automatically routes the future requests to other up and running server nodes in the cluster. This enables the service as a whole to stay available.

Load balancers act as a single point of contact for all the client requests.



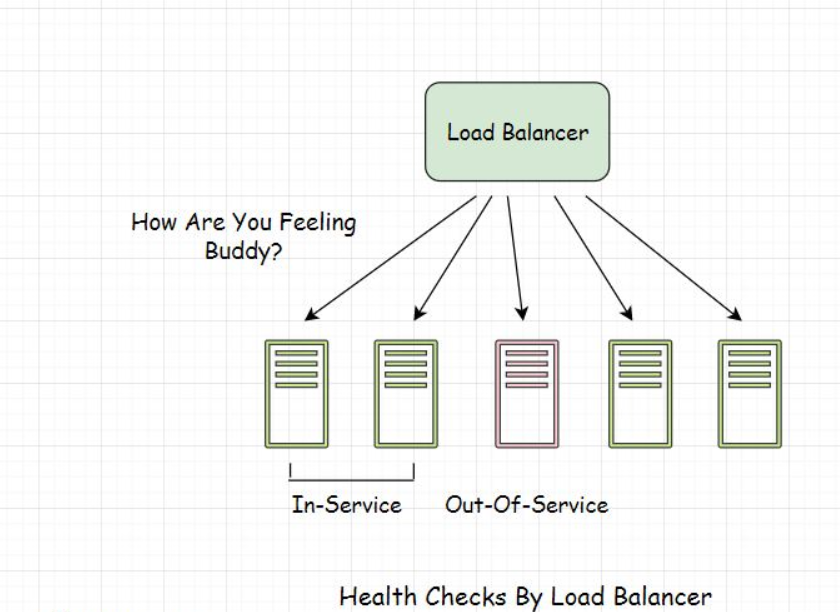
They can also be set up at the application component level to efficiently manage traffic directed towards any application component, be it the backend application server, database component, message queue, or any other. This is done to uniformly spread the request load across the machines in the cluster powering that particular application component.



**Performing health checks of the servers with load balancers**[#](https://www.educative.io/module/lesson/web-application-architecture-101/R8zN8YoxD8R#Performing-health-checks-of-the-servers-with-load-balancers)

To intelligently route all the user requests to the active nodes in the cluster, a load balancer should be well aware of their running status.

To ensure that the user request is always routed to the machine that is up and running, load balancers regularly perform health checks on the machines in the cluster.



Ideally, a load balancer maintains a list of machines that are up and running in the cluster in real-time, and the user requests are forwarded to only those machines in service. If a machine goes down, it is removed from the list.

Machines that are up and running in the cluster are known as *in-service* machines, and the servers that are down are known as *out-of-service* instances.

For the record, *Node, Server, Server Node, Instance*, and *Machine* all mean the same thing and can be used interchangeably.

After the *out-of-service* instance comes back online and becomes *in-service*, the load balancer updates its list and starts routing the future requests to that particular instance all over again.

Okay! In the next few lessons, you will discover how load balancers work. To understand that well, you need to first understand the *Domain Name System* (DNS) that we will discuss in the lesson up next.

**Understanding DNS – Part 1**

In this lesson, you will learn what the domain name system (DNS) is and how it works.

**We'll cover the following**

* + [Domain name system](https://www.educative.io/module/lesson/web-application-architecture-101/NEYnrVNMJ76#Domain-name-system)
  + [How does a domain name system work?](https://www.educative.io/module/lesson/web-application-architecture-101/NEYnrVNMJ76#How-does-a-domain-name-system-work?)

Every machine that is online and is a part of the *World Wide Web* (WWW) has a unique *IP address* that enables it to be contacted by other machines on the web using that particular *IP address*.

*IP* stands for *Internet Protocol*. It’s a protocol that facilitates the delivery of data packets from one machine to another using their IP addresses.

*2001:db8:0:1234:0:567:8:1* - This is an example of a machine’s *IP address*. The server that hosts our website will have a similar *IP address*. To fetch content from that server, a user has to type in the unique IP address of the server in their browser’s URL tab and hit enter to interact with the website’s content.

Naturally, it is not viable to type in the website’s IP address from memory every time we visit a particular website. Even if we try to, how many IP addresses do you think you can remember?

Typing in domain names, for instance, [*amazon.com*](http://amazon.com/), is a lot easier than working directly with IP addresses. I think we can all agree on this.

**Domain name system**[#](https://www.educative.io/module/lesson/web-application-architecture-101/NEYnrVNMJ76#Domain-name-system)

*Domain name system*, commonly known as *DNS*, is a system that averts the need to remember long *IP addresses* to visit a website by mapping easy-to-remember domain names to IP addresses.

[*amazon.com*](http://amazon.com/) is a domain name that is mapped to its unique *IP address* by the *DNS* so that we are not expected to type in the *IP address* of [*amazon.com*](http://amazon.com/) into our browsers every time we visit that website.

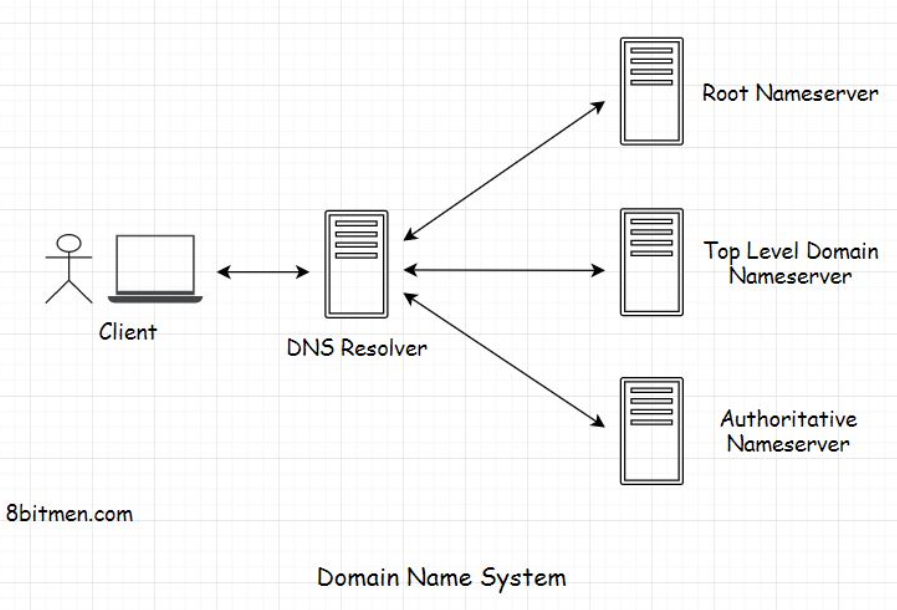
Now let’s explore how DNS works?

**How does a domain name system work?**[#](https://www.educative.io/module/lesson/web-application-architecture-101/NEYnrVNMJ76#How-does-a-domain-name-system-work?)

When a user types in the *URL* of the website in their browser and hits enter, this event is known as *DNS querying*.

Four key components, or a group of servers, make up the DNS infrastructure. These are:

* *DNS Recursive nameserver aka DNS Resolver*
* *Root nameserver*
* *Top-Level Domain nameserver*
* *Authoritative nameserver*



# Understanding DNS – Part 2

This lesson continues our discussion on the domain name system.

In this lesson, you will get an insight into the complete DNS query lookup process and understand the role of different servers in the DNS infrastructure.

Lets’ begin.

When the user hits enter after typing in the domain name into their browser, the browser sends a request to the DNS Recursive nameserver, also known as the DNS Resolver.

The role of DNS Resolver is to receive the client request and forward it to the Root nameserver to get the address of the Top-Level domain nameserver.

The DNS Recursive nameserver is generally managed by our ISP (Internet service provider). The whole DNS system is a distributed system setup in large data centers managed by internet service providers.

These data centers contain clusters of servers optimized to process DNS queries in minimal time in milliseconds.

Once the DNS Resolver forwards the request to the Root nameserver, the Root nameserver returns the address of the Top-Level domain nameserver in response. As an example, the top-level domain for [*amazon.com*](http://amazon.com/) is .com.

Once the DNS Resolver receives the address of the top-level domain nameserver, it sends a request to it to fetch the details of the domain name. Top Level domain nameservers hold the data for domains using their top-level domains.

For instance, the .com top-level domain nameserver will contain information on domains using .com. Similarly, a .edu top-level domain nameserver will hold information on domains using .edu.

Since our domain is [*amazon.com*](http://amazon.com/), the DNS Resolver will route the request to the .com top-level domain name server.

Once the top-level domain name server receives the request from the Resolver, it returns the IP address of the [*amazon.com*](http://amazon.com/) domain name server.

[*amazon.com*](http://amazon.com/) domain nameserver is the last server in the DNS query lookup process. This nameserver is responsible for the [*amazon.com*](http://amazon.com/) domain and is also known as the Authoritative nameserver. The owner of the domain name owns this nameserver.

Then, DNS Resolver fires a query to the Authoritative nameserver, and it returns the IP address of the [*amazon.com*](http://amazon.com/) website to the DNS Resolver.

DNS Resolver caches the data and forwards it to the client.

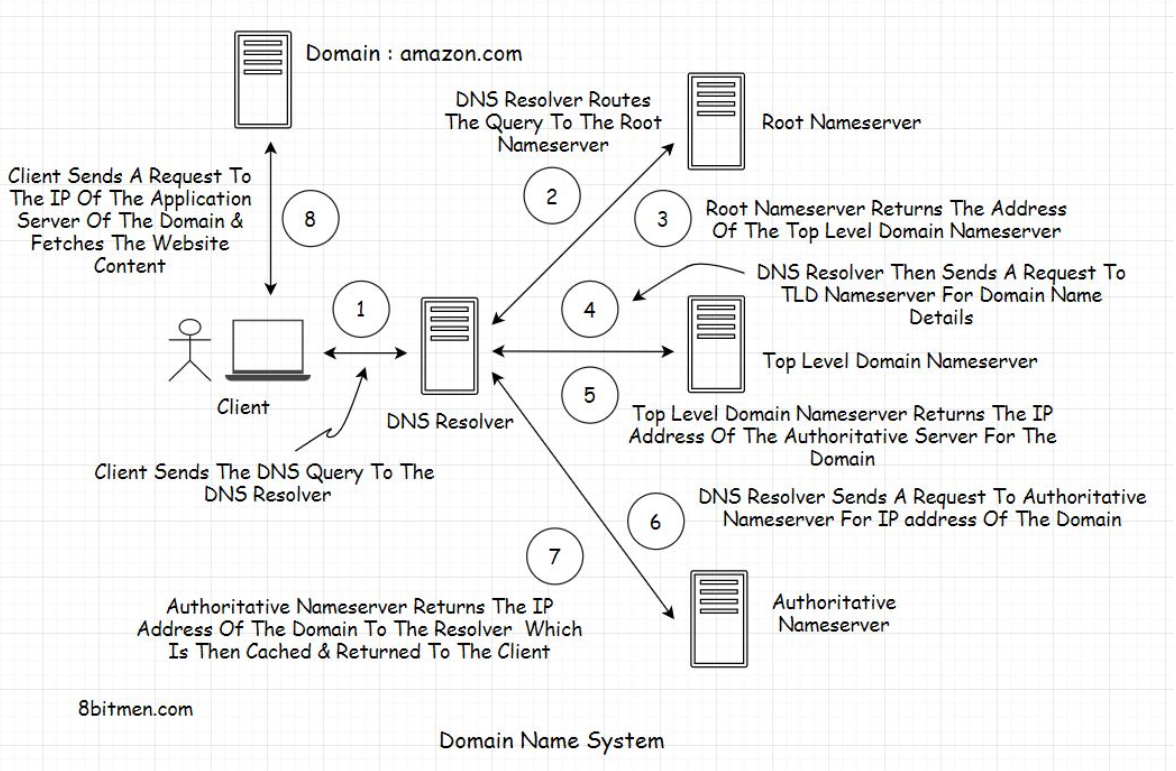
On receiving the response, the browser sends a request to the [*amazon.com*](http://amazon.com/) website’s IP address to fetch data from their servers.

Often all this DNS information is cached, and the DNS servers don’t have to do so much rerouting every time a client requests an IP of a certain website.

The DNS information of websites that we visit also gets cached in our local machines, that is, our browsing devices with a TTL (Time to live).

All modern browsers do this automatically to cut down the DNS query lookup time when revisiting a website.

This is how the entire DNS query lookup process works.



In the next lesson, let’s get an insight into DNS load balancing.

**DNS Load Balancing**

In this lesson, we will cover DNS load balancing.

**We'll cover the following**

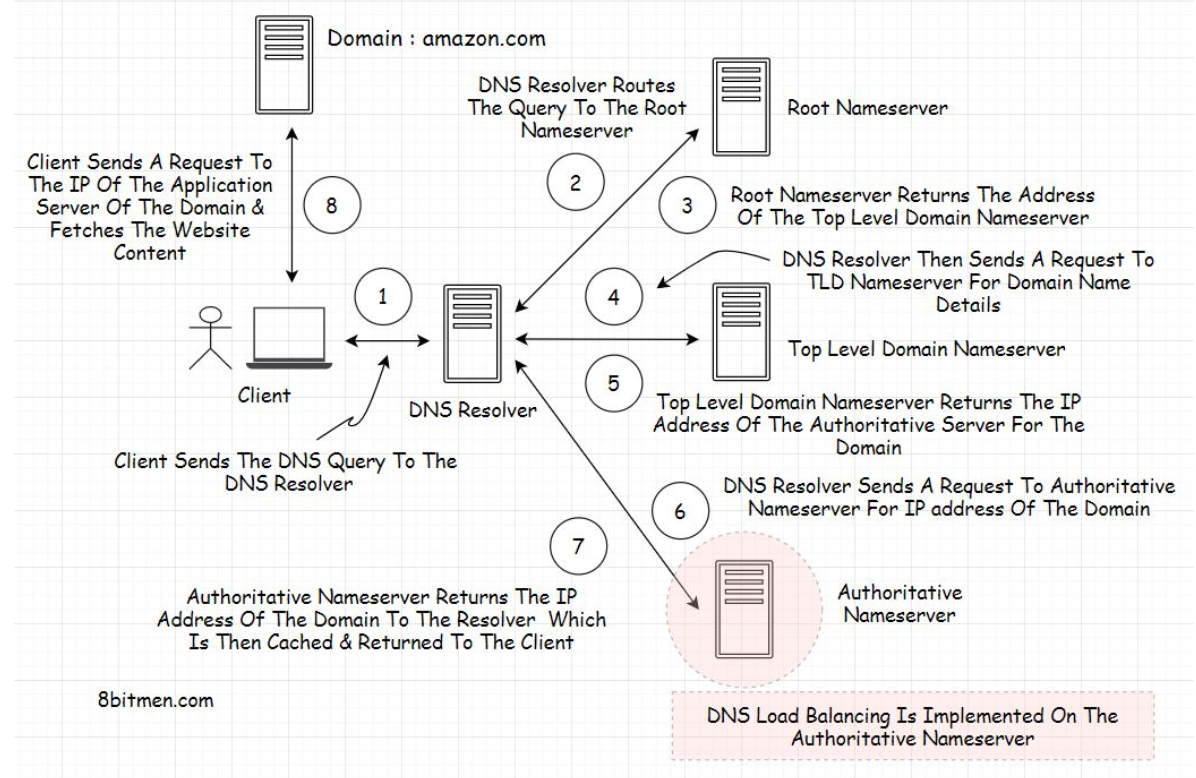
* + [DNS load balancing](https://www.educative.io/module/lesson/web-application-architecture-101/JEWqoGNDXn9#DNS-load-balancing)
  + [Limitations of DNS load balancing](https://www.educative.io/module/lesson/web-application-architecture-101/JEWqoGNDXn9#Limitations-of-DNS-load-balancing)

**DNS load balancing**[#](https://www.educative.io/module/lesson/web-application-architecture-101/JEWqoGNDXn9#DNS-load-balancing)

In the previous lesson, we discussed how the *DNS query lookup* process works and the role of different servers in the domain name system. The end server, in the lookup chain, is the *authoritative server*, which returns the *IP address* of the domain.

When a large-scale service such as [*amazon.com*](http://amazon.com/) runs, it needs way more than a single machine to run its services. A service as big as [*amazon.com*](http://amazon.com/) is deployed across multiple data centers in different geographical locations across the globe.

To spread the user traffic across different clusters in different data centers. There are various ways to set up *load balancing*. In this lesson, we will discuss *DNS load balancing*, which is set up at the *DNS* level on the *authoritative server*.



*DNS load balancing* enables the *authoritative server* to return different *IP addresses* of a particular domain to the clients. Every time it receives a query for an *IP*, it returns a list of *IP addresses* of a domain to the client.

With every request, the *authoritative server* changes the order of the *IP addresses* in the list in a *round-robin* fashion.

As the client receives the list, it sends out a request to the first *IP address* on the list to fetch the data from the website. The reason for returning a list of *IP addresses* to the client is to enable it to use other *IP addresses* in the list in case the first doesn’t return a response within a stipulated time.

When another client sends out a request for an *IP address* to the *authoritative server*, it re-orders the list and puts another *IP address* at the top of the list following the *round-robin* algorithm.

Also, when the client hits an *IP*, it may not necessarily hit an *application server*. Instead, it may hit another *load balancer* implemented at the data center level that manages the clusters of application servers.

**Limitations of DNS load balancing**[#](https://www.educative.io/module/lesson/web-application-architecture-101/JEWqoGNDXn9#Limitations-of-DNS-load-balancing)

*DNS load balancing* is largely used by companies to distribute traffic across multiple data centers that the application runs in. However, this approach has several limitations.

For instance, it does not take into account the current load on the servers, the content they hold, their request processing time, their in-service status, and so on.

Also, since these *IP addresses* are cached by the client’s machine and the *DNS Resolver*, there is always a possibility of a request being routed to a machine that is out of service.

*DNS load balancing*, despite its limitations, is preferred by companies because it is an easy and less expensive way of setting up *load balancing* on their services.

***Recommended Read*** – [Round Robin DNS](https://en.wikipedia.org/wiki/Round-robin_DNS)

# Load Balancing Methods

In this lesson, you will get an insight into hardware and software load balancing.

**We'll cover the following**

* + [Hardware load balancers](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Hardware-load-balancers)
  + [Software load balancers](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Software-load-balancers)
  + [Algorithms/Traffic routing approaches leveraged by load balancers](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Algorithms/Traffic-routing-approaches-leveraged-by-load-balancers)
    - [Round robin and weighted round robin](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Round-robin-and-weighted-round-robin)
    - [Least connections](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Least-connections)
    - [Random](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Random)
    - [Hash](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Hash)

There are primarily three modes of load balancing:

* DNS Load Balancing
* Hardware-based Load Balancing
* Software-based Load Balancing

We discussed DNS load balancing in the previous lesson. In this one, we will look into hardware and software load balancing.

Let’s get on with it.

Hardware-based and software-based load balancing are common ways of balancing traffic loads on large-scale services. Let’s begin with hardware-based load balancing.

## Hardware load balancers[#](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Hardware-load-balancers)

Hardware load balancers are highly performant physical hardware. They sit in front of the application servers and distribute the load based on the number of currently open connections to a server, compute utilization, and several other parameters.

Since these load balancers are physical hardware, they need maintenance and regular updates, just like any other server hardware would need. They are also expensive to set up compared to software load balancers, and their upkeep may require a certain skill set.

Also, the hardware load balancers have to be overprovisioned upfront to deal with peak traffic, which is not the case with software load balancers.

But when it comes to performance, hardware load balancers stand out.

If the business has network specialists and an IT team in-house, they can manage these load balancers. Otherwise, the developers are expected to wrap their heads around setting up these hardware load balancers with some assistance from the vendors. This is also the primary reason why developers prefer working with software load balancers.

Now, let’s take a look at software-based load balancing.

## Software load balancers[#](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Software-load-balancers)

Software load balancers can be installed on commodity hardware and VMs. They are more cost-effective and offer more flexibility to the developers. Software load balancers can be upgraded and provisioned easily compared to hardware load balancers.

You will also find several Load Balancers as a Service (LBaaS) services online that enable you to directly plug in load balancers into your application without you having to do any sort of setup.

Software load balancers are pretty advanced compared to DNS load balancing. They consider many parameters such as data hosted by the servers, cookies, HTTP headers, CPU and memory utilization, load on the network, etc., to route traffic across the servers.

They also continually perform health checks on the servers to keep an updated list of in-service machines.

Development teams prefer to work with software load balancers as hardware load balancers require specialists to manage them.

[HAProxy](https://www.haproxy.com/) is one example of a software load balancer widely used by the big guns in the industry, including GitHub, Reddit, Instagram, AWS, Tumblr, StackOverflow, to scale their systems.

Besides the round Robin algorithm, which DNS Load balancers use, software load balancers leverage several other algorithms to efficiently route traffic across the machines. Let’s take a look.

## Algorithms/Traffic routing approaches leveraged by load balancers[#](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Algorithms/Traffic-routing-approaches-leveraged-by-load-balancers)

### Round robin and weighted round robin[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Round-robin-and-weighted-round-robin)

We know that the round robin algorithm sends IP addresses of machines sequentially to the clients. Parameters such as the server load, CPU consumption, and so on are not considered when sending the IP addresses to the clients.

We have another approach known as the weighted round robin, where based on the server’s compute and traffic handling capacity, weights are assigned to them. And then, based on server weights, traffic is routed to them using the round robin algorithm.

With this approach, more traffic is converged to machines that can handle a higher traffic load, thus efficiently using the resources.

This approach is pretty useful when the service is deployed across multiple data centers having different compute capacities. More traffic can be directed to the larger data centers containing more machines.

### Least connections[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Least-connections)

When using this algorithm, the traffic is routed to the machine with the least open connections of all the machines in the cluster. There are two approaches to implement this.

In the first, it is assumed that all the requests will consume an equal amount of server resources, and the traffic is routed to the machine with the least open connections based on this assumption.

In this scenario, there is a possibility that the machine with the least open connections might already be processing requests demanding most of its CPU power. Routing more traffic to this machine would not be a good idea.

In the other approach, the CPU utilization and the request processing time of the chosen machine are also considered before routing the traffic to it. Machines with the shortest request processing time, most negligible CPU utilization, and the least open connections are suitable candidates to process future client requests.

The least connections approach comes in handy when the server has long opened connections like persistent connections in a gaming application.

### Random[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Random)

Following this approach, the traffic is randomly routed to the servers. The load balancer may also find similar servers in terms of existing load, request processing time, and so on. Then it randomly routes the traffic to these machines.

### Hash[**#**](https://www.educative.io/module/lesson/web-application-architecture-101/m2pORlwpwJE#Hash)

In this approach, the source IP where the request is coming from and the request URL are hashed to route the traffic to the backend servers.

Hashing the source IP ensures that a client’s request with a certain IP will always be routed to the same server.

This facilitates a better user experience as the server has already processed the initial client requests and holds the client’s data in its local memory. There is no need for it to fetch the client session data from the session memory of the cluster and process the request. This reduces latency.

Hashing the client IP also enables the client to re-establish the connection with the same server that was processing its request in case the connection drops.

Hashing a URL ensures that the request with that URL always hits a certain cache that already has data on it. This is to ensure that there is no cache miss.

This also averts the need for duplicating data in every cache and is, thus, a more efficient way to implement caching.

Well, this is pretty much it on the fundamentals of load balancing. In the next chapter, we will cover monoliths and microservices.