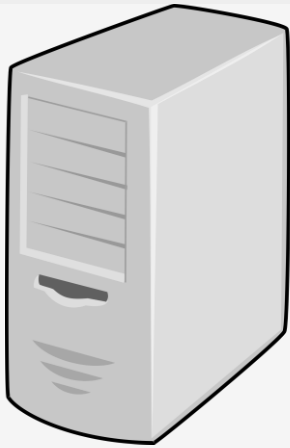
Servers

Presentation plan

1. What are servers?
2. How servers work and how they communicate?
3. Requests and responses – quick HTTP reminder
4. Two types of clients – thin and thick
5. Platform scaling – vertical and horizontal
6. Server platforms – point of failure
7. Different types of services
8. Clouds
9. Comparison of different server applications

Some theory behind servers – what are servers?

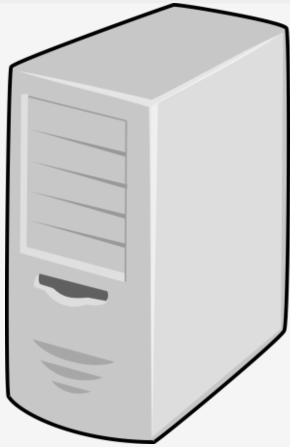


Servers can have two meanings in the context of our work.

Servers are web applications which have communication interface that can be used to send or receive messages. Basically – servers are applications that are able to communicate with other applications.

The easiest way to communicate is to create a socket or an endpoint to which we can connect.

Some theory behind servers – what are servers?

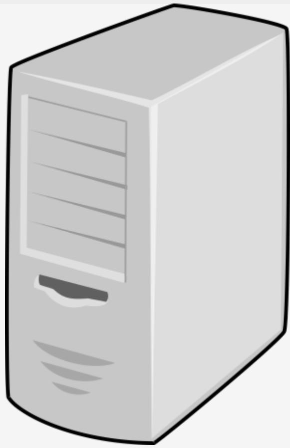


Servers can have two meanings in the context of our work.

Servers are web applications which have communication interface that can be used to send or receive messages. Basically – servers are applications that are able to communicate with other applications.

The easiest way to communicate is to create a socket or an endpoint to which we can connect.

Some theory behind servers – what are servers?

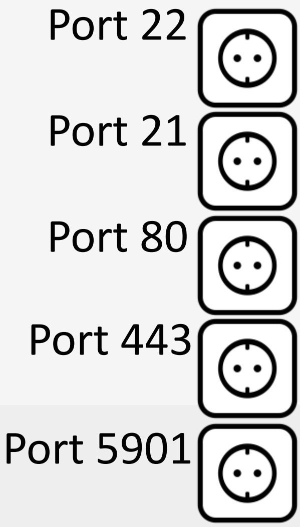


Servers can have two meanings in the context of our work.

Servers are web applications which have communication interface that can be used to send or receive messages. Basically – servers are applications that are able to communicate with other applications.

The easiest way to communicate is to create a socket or an endpoint to which we can connect.

Server applications use computer ports to communicate. Each application can allow new incomming clients to connect to a single port. Each computer has 65535 ports that can be used for communication (16 bit number). There can only be one application listening for new connections at a single port.



Some theory behind servers – what are servers?

From all network ports in our operating systems not all have to be open. Each operating has a security service responsible for controling open ports – most systems have firewall. The lowest 1024 port numbers are called well known port numbers, and identify historically most commonly used services.

For Example:

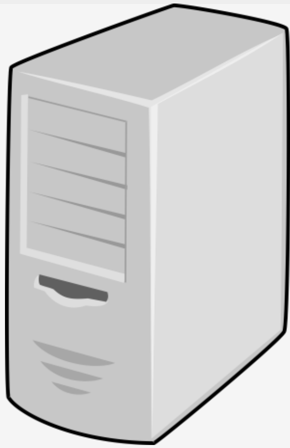
- Port 7 – icmp (ping)
- Port 22 – ssh (secure shell)
- Port 80 – http (website port)
- Port 25 – smtp (e-mail routing)
- Port 443 – https (secure website)
- Port 554 – rtsp (video transfer protocol)

Many other ports (above port 1024) are commonly known to be used by different services:

- Port 3306 – mysql port
- Port 5432 – postgresql port
- Port 5900 – vnc (remote desktop application)

...and many others.

Application servers



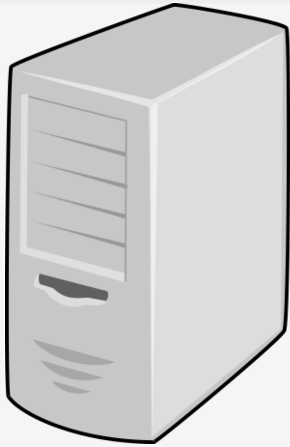
Application servers are designed to provide us some basic functionalities/mechanisms required to run web applications. They:

- Manage threads
- Dynamically load and maintain applications
- Provide authorisation and authentication (JAAS – Java Authentication and Authorization Service)
- Handle HTTP Protocol
- Handle Encryption and SSL

Other than that application servers handle much more functionalities of a web server. All of them are available using special API. For example, Tomcat functionalities :

http://tomcat.apache.org/tomcat-8.0-doc/

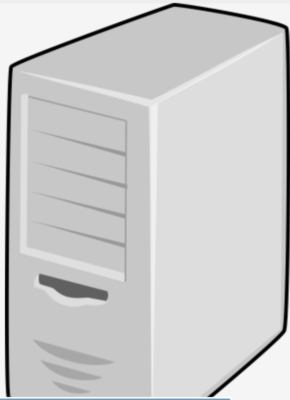
Hardware servers



Most of the time when we are talking about servers we mean hardware. By this we mean machines running our software. By this we also mean different devices than those used commonly in our homes.

Hardware servers have the same architecture as our comuters – they consist of processors, memory, motherboard, hard drives and other components. If we look closer to their parameters, we might see some differences.

Hardware servers

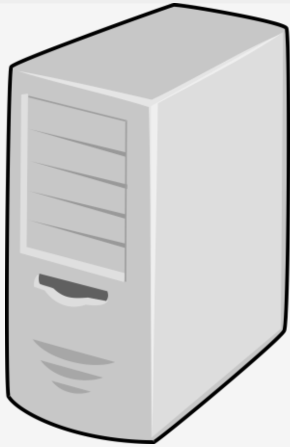


Most of the time when we are talking about servers we mean hardware. By this we mean machines running our software. By this we also mean different devices than those used commonly in our homes.

Hardware servers have the same architecture as our comuters – they consist of processors, memory, motherboard, hard drives and other components. If we look closer to their parameters, we might see some differences.

Parameter	Dekstop	Server
CPU	2-8 cores, 32-64 bit single processor (i3, i5, i7). Sometimes ARM architecture.	Minimum one 10 core processor, sometimes 2-4 or more processors. Mostly Xenon – designed for best performance.
RAM	2-8, sometimes 16 GB of RAM. Smaller devices could run even on 1GB.	At least 32 GB of ram. Up to 1 TB of RAM memory for a single machine. Built in error check mechanism.
HDD	Single or sometimes two drives with up to 2 TB of memory.	10-100 TB of HDD space on drives with sometimes another SSD drive for operating system.
LAN	WiFi + LAN, up to 1Gb networking interface for gaming PC’s	At least 4 LAN interfaces with 1Gb -10Gb networking interface.
GPU	One or two graphic cards with 1-2 GB memory and 256 bit data bus	Single integrated graphics card with 8-256 MB of memory and 64 bit data bus.

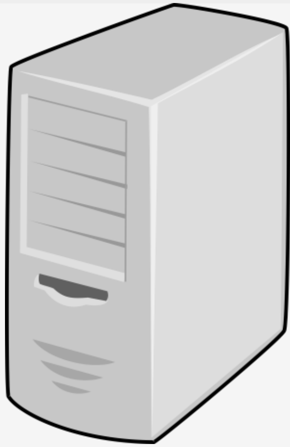
Hardware servers



Servers are not worse than desktop computers, they are meant to do different things than desktop computers.

Most desktop computers are gaming PC’s or Business laptops. They have long lasting batteries or high performance graphic cards. Servers are designed to process as many requests as possible. They can keep more sessions and connections alive to handle more clients requests.

Hardware servers

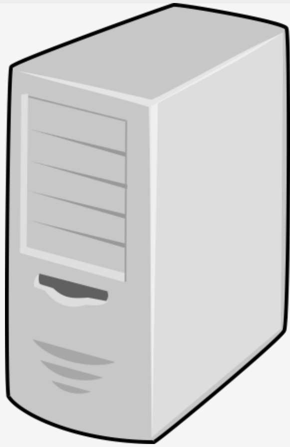


Servers are not worse than desktop computers, they are meant to do different things than desktop computers.

Most desktop computers are gaming PC’s or Business laptops. They have long lasting batteries or high performance graphic cards. Servers are designed to process as many requests as possible. They can keep more sessions and connections alive to handle more clients requests.

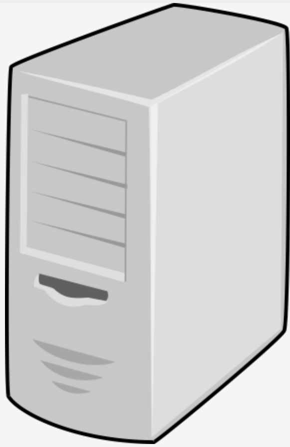
Most servers are able to process from 100 up to 4000 requests per second. To make it a little bit more understandable – if we have **1 milion users sending 5 requests per session** using a **single server for 4 hours** and their requests are spread evenly around that time, they would only make **350 requests per second**.

Application servers and their ways of communication



As mentioned before – servers use ports to communicate. We need to open port and then listen for clients to accept new requests. We can use one of two internet protocols to communicate.

Application servers and their ways of communication



As mentioned before – servers use ports to communicate. We need to open port and then listen for clients to accept new requests. We can use one of two internet protocols to communicate.

TCP communication – using this protocol is much safer from the user/programming side. First of all, each frame is confirmed to be delivered, which gives us certainty that the connection is being kept alive and both sides are taking part in communicating.

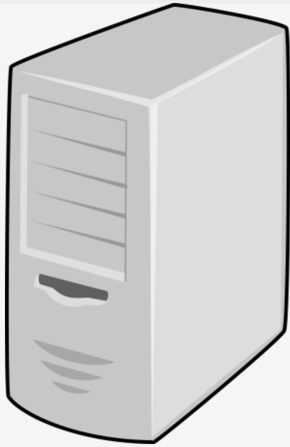
On the other hand, using this protocol requires both sides to send more data with each frame. Other than frames, we also need to send ACK which is Acknowledge packet. Each packet has to be confirmed by ACK.

After connecting we can always check if the connection is alive and we can send our data with confirmation that it has been delivered but...

... at what cost.

TCP Connections are used by : http, ssh, ftp, smtp, imap...

Application servers and their ways of communication



As mentioned before – servers use ports to communicate. We need to open port and then listen for clients to accept new requests. We can use one of two internet protocols to communicate.

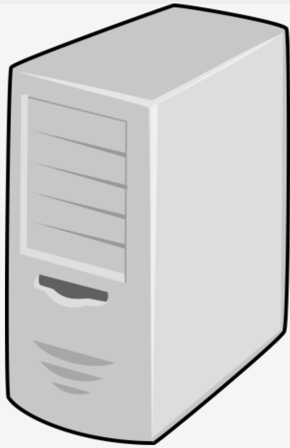
UDP communication – uses shorter frames and shorter headers (each frame has header 12 bytes shorter than TCP). UDP is faster because no error recovery is attempted and there is no flow control, so we have no guarantee that our data will be delivered.

Unfortunately without guarantee we don’t know if our data will arrive at the destination. Data integrity can only be checked if every packet is delivered. If any error occurs, than no recovery is attempted.

With UDP we don’t know if the other side is still active. The only way to know if the recipient is there is by sending a message and waiting for any response.

UDP Connections are used by: tunneling, media streaming, games, broadcasting and any other area in which packets doesn’t have to be delivered.

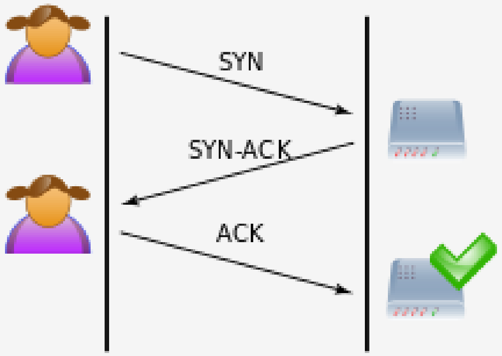
Application servers and their ways of communication



We have two ways of communicating and HTTP uses TCP connection.

In HTTP client initializes an HTTP session by openin TCP connection to the HTTP server with which he wishes to communicate. It then sends request messages to the server, each of which specifies particular type of action that user of the HTTP client would like the server to take. The server responds to the client's request and sends the data which he wants.

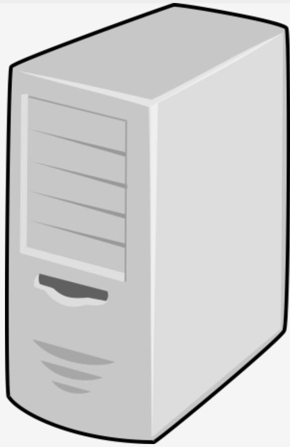
HTTP uses only one connection. Server uses the same connection to send data to the client and there is no guarantee that the data will arrive in the TTL (time to alive). In this case data is being retransmitted.



Application servers and their ways of communication

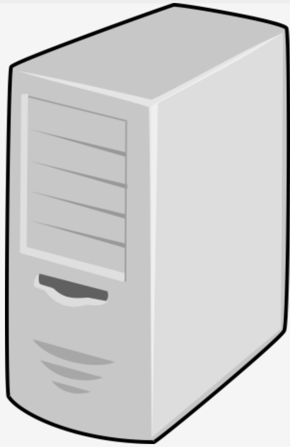
We have two ways of communicating and HTTP uses TCP connection.

In HTTP client initializes an HTTP session by openin TCP connection to the HTTP server with which he wishes to communicate. It then sends request messages to the server, each of which specifies particular type of action that user of the HTTP client would like the server to take. The server responds to the client's request and sends the data which he wants.



HTTP uses only one connection. Server uses the same connection to send data to the client and there is no guarantee that the data will arrive in the TTL (time to alive). In this case data is being retransmitted.

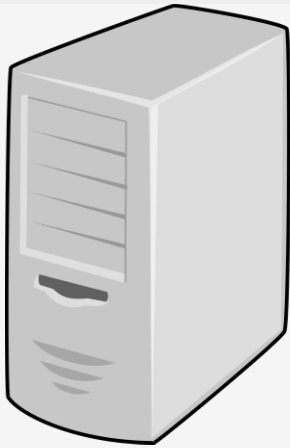
Requesting website vs requesting data – two types of client



When we communicate with servers there are two ways of getting content presented on our machines.

There are two techniques called **thin clients** and **thick clients**.

Requesting website vs requesting data – two types of client

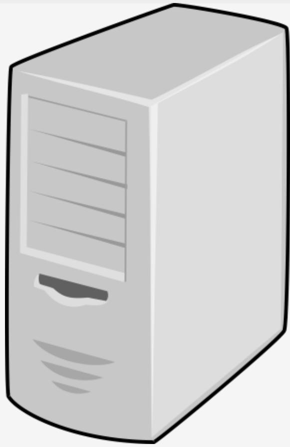


When we communicate with servers there are two ways of getting content presented on our machines.

There are two techniques called **thin clients** and **thick clients**.

The choice to develop thick or thin client is ours to make. We can create solution with in both ways if we want to. There are some differences that are very important. The difference between two solutions is how we develop our applications.

Requesting website vs requesting data – two types of client



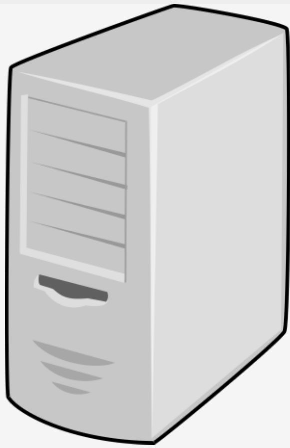
When we communicate with servers there are two ways of getting content presented on our machines.

There are two techniques called **thin clients** and **thick clients**.

The choice to develop thick or thin client is ours to make. We can create solution with in both ways if we want to. There are some differences that are very important. The difference between two solutions is how we develop our applications.

We can create thin client, and that is the simplest way of development. It means that our client will only receive the website from the server, and other than that it does absolute minimum on the view and processing side. So if we have thin client, the responsibility to present the website the way it's supposed to be is on the server side.

Thin client – thick server



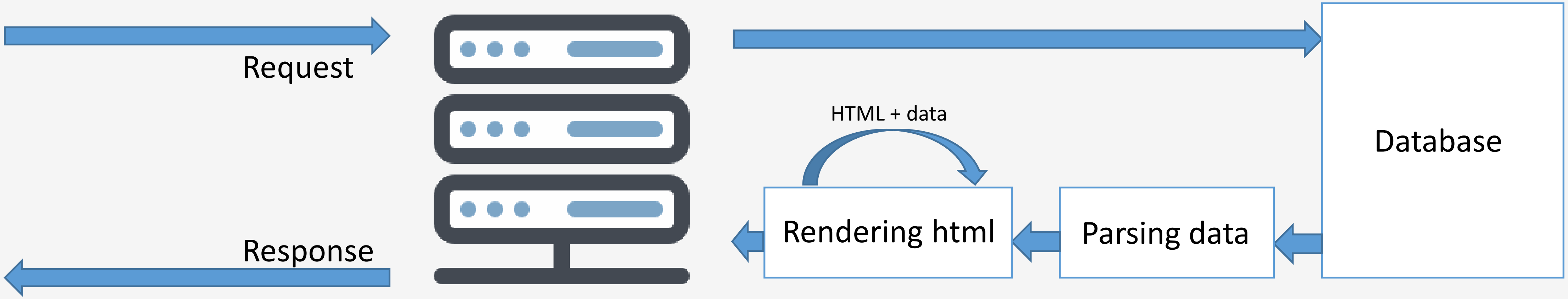
Let’s say we have an online shop and we are browsing for some products (for my purposes i will use clothes shop). I will than enter the website and go to the browsing page. There I should have a way to filter some products and i will choose t-shirts with my size. But let’s see what happens with the requests:

My computer created a new connection. After that request has been sent to the server with all filtering parameters. Than server is using it’s database connection to search through the products and retrieves (let’s say) 10 000 records. Now after that data is retrieved from the database we have to present it at our website.

With thin client that responsibility is on the server side. All records are being parsed into the website and the only thing that is being returned to the client is the view – the html result of my search.

Thin client – thick server

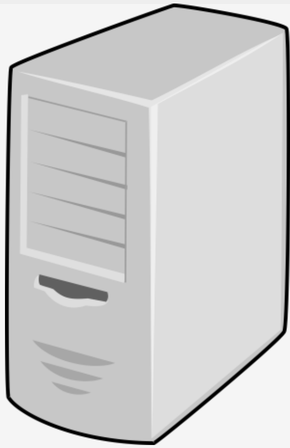
Let's say we have an online shop and we are browsing for some products (for my purposes i will use clothes shop). I will than enter the website and go to the browsing page. There I should have a way to filter some products and i will choose t-shirts with my size. But let's see what happens with the requests:



My computer created a new connection. After that request has been sent to the server with all filtering parameters. Than server is using it's database connection to search through the products and retrieves (let's say) 10 000 records. Now after that data is retrieved from the database we have to present it at our website.

With thin client that responsibility is on the server side. All records are being parsed into the website and the only thing that is being returned to the client is the view – the html result of my search.

Thick client – thin server

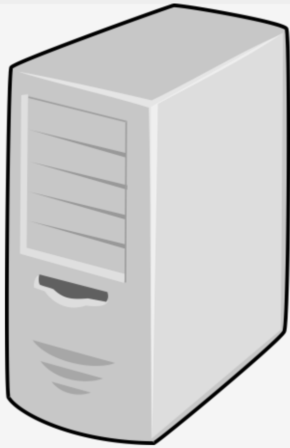


Now let's take a look at the same situation (online clothes shop) but this time we will apply thick client to the model. I am making the same request to the server, but this time the rendering part is on my side.

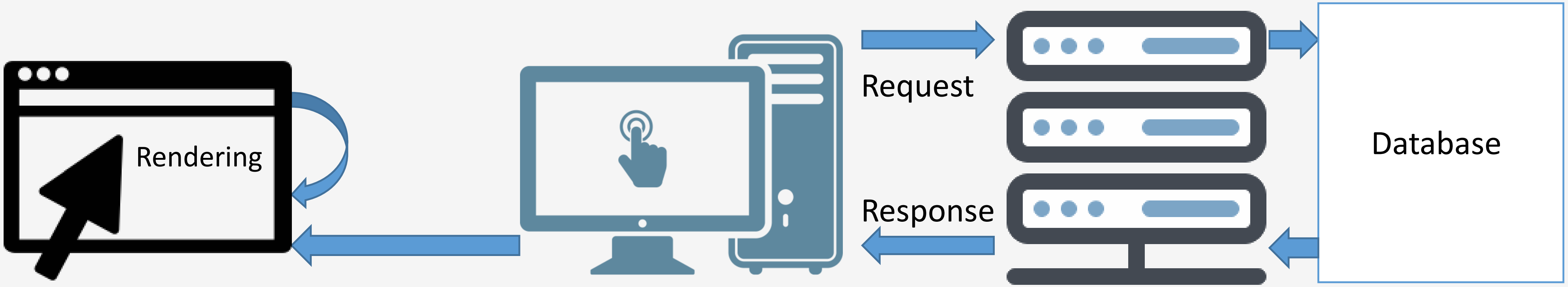
After request has been sent to the server with all filtering parameters server will search for the data and retrieve it from the database. This time instead of rendering that data and building it into html response server uses chunkes of data to deliver everything to the client.

This way server is thinner – by that we mean less processing power is used by the server, and more processing power is required from the client side. Server only delivers the data to the client. After data has been sent the rendering part takes place on the client's browser.

Thick client – thin server



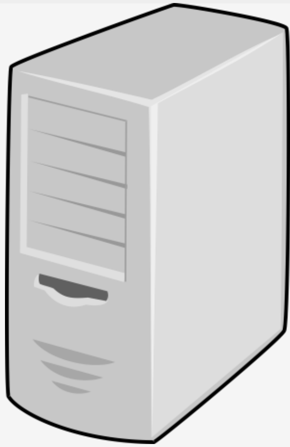
Now let’s take a look at the same situation (online clothes shop) but this time we will apply thick client to the model. I am making the same request to the server, but this time the rendering part is on my side.



After request has been sent to the server with all filtering parameters server will search for the data and retrieve it from the database. This time instead of rendering that data and building it into html response server uses chunkes of data to deliver everything to the client.

This way server is thinner – by that we mean less processing power is used by the server, and more processing power is required from the client side. Server only delivers the data to the client. After data has been sent the rendering part takes place on the client’s browser.

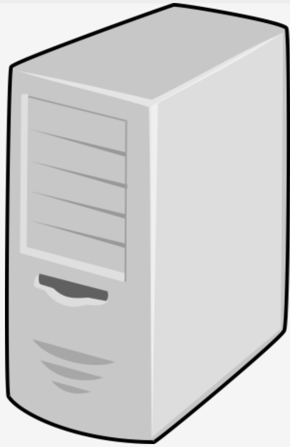
Thick client or thin client – which one to choose



Here are some main differences between those two models:

Compared property	Thick client	Thin client
Who needs more processing power for rendering?	Client requires more power	Server requires more power
Who is responsible for loading logic?	Client is loading all logic in form of browser scripts	Server is loading all logic
Is presentation layer dependant on the platform	Yes, each browser can display content differently	No, client will always receive the same view
What is being send from server to client?	Only data (most of the time in form of xml or json)	Only view in form of html or different browser view
Number of demands	Reduced server demands	More server demands
Need for resources/servers	Requires more resources but less servers	More servers are needed – more processing power is required

Creating our own infrastructure

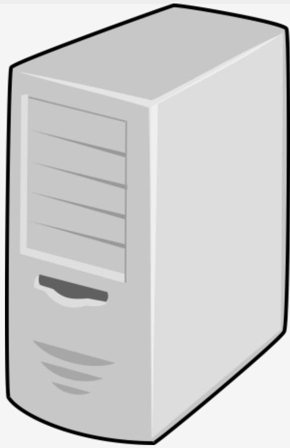


When designing our own solution we’ll always think at least once about creating our own infrastructure to hold serve our websites. In most cases those services will not require too much resources and we’ll be able to support our website using 1000$ server.

But what happens if our project happens to be a hit and we will need to grow.

There comes a time when we’ll need to make a decision on how do we want to build our infrastructure which in this case means our hardware and our network. How do we want to handle our requests? Is it better to upgrade our current server, or is it better to buy a new one and place it next to each other?

Vertical vs Horizontal scaling

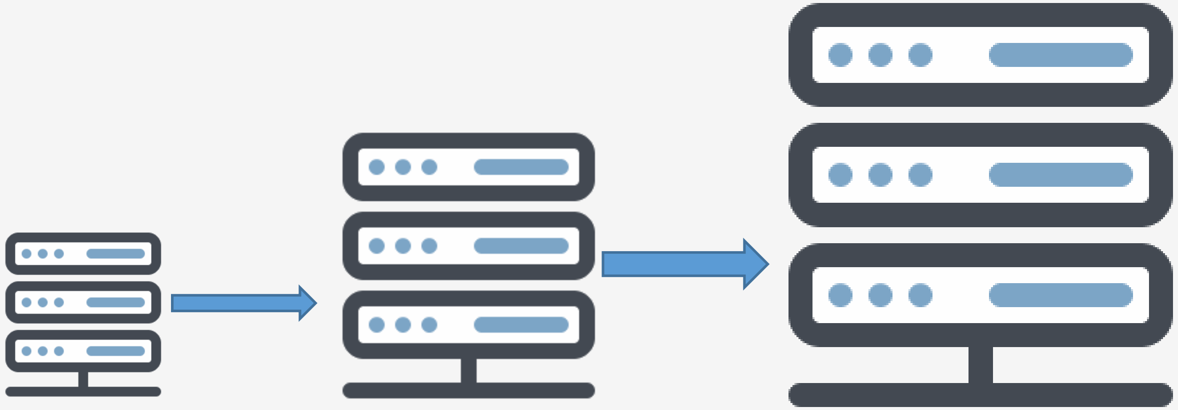


Vertical and horizontal scaling are ways to upgrade our infrastructure to handle more requests.

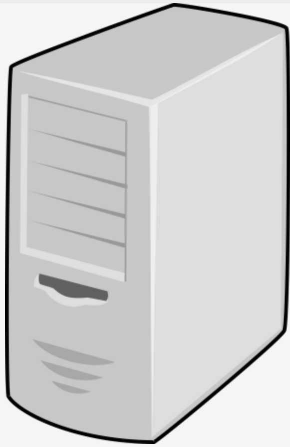
In one case we have vertical scaling which is upgrading our current hardware to be faster. With this solution we will upgrade our current devices and their components such as CPU, RAM, HDD and others.

This solution is good if we want just a little upgrade, but if this is our only machine, we also have to face the reality which is – there will be some downtime for our website while the upgrade is done.

Vertical scaling is also limited. There is only limited ways of upgrading same hardware. There is limited number of CPU or RAM slots in each motherboard. On the plus side, it's easier to maintain this one particular device.



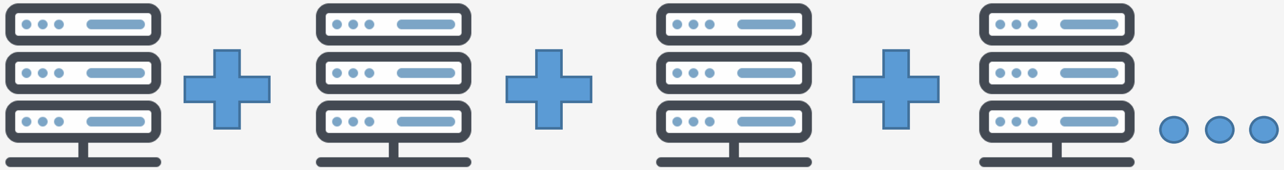
Vertical vs Horizontal scaling



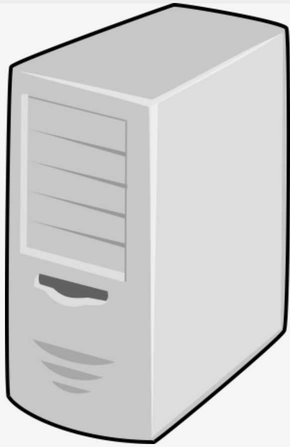
On the other hand we have horizontal scaling which is adding another device in a group of servers, connected with each other or another device called load balancer.

This solution might be more expensive then vertical scaling, but it gives some advantages over the vertical scaling. If we want to grow our system we don’t need to stop other devices, so the downtime is reduced. System might be more complicated than maintaining one device, but there are no limitations on how big the platfom can grow.

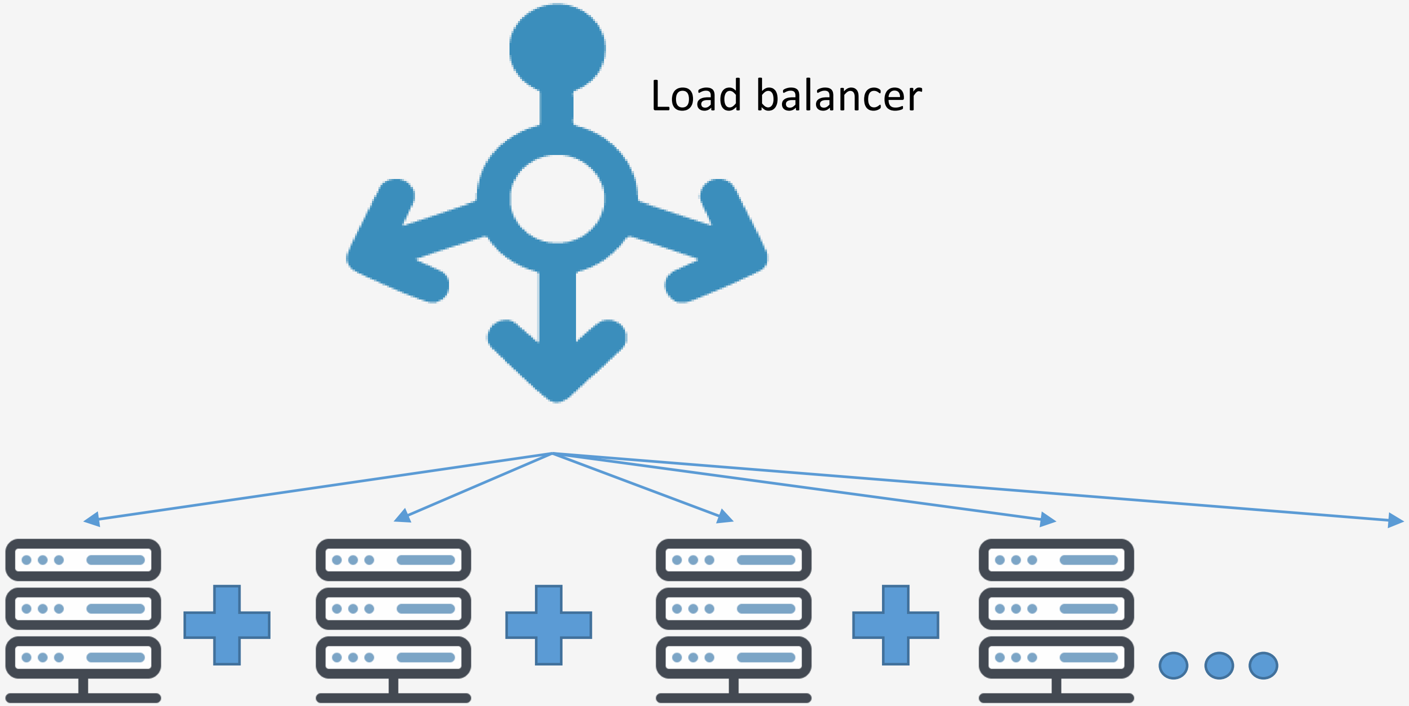
With this solution we might need another device, which is load balancer.



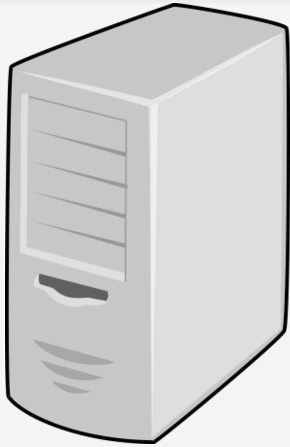
Vertical vs Horizontal scaling – load balancer



Load balancer is a device that takes all network movement and distributes it evenly among our devices. This way our infrastructure is mor resistant to a single device failure.



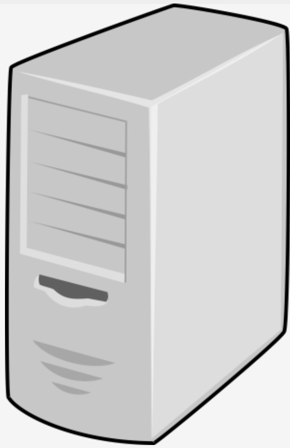
Vertical vs Horizontal scaling - comparison



Here are some main differences between two infrastructure architectures:

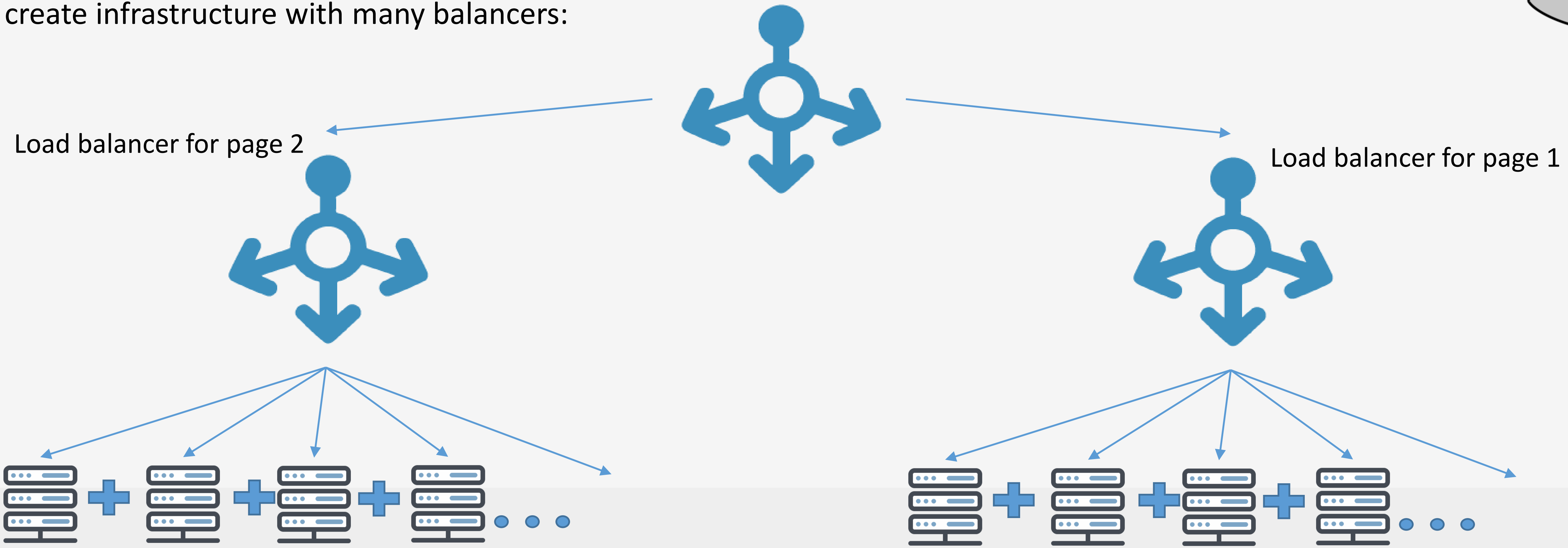
Compared property	Vertical scaling	Horizontal scaling
Cost	Less cost at the beginning, but more expensive on bigger upgrades	Same cost at any level of upgrade. Cost is dependant on the type of machine (expensive cost more)
Limitations	Hardware limitations, socket limitations.	No limitations other than space and infrastructure.
Architecture complexity	Simple, single machine maintenence	More machines are harder to maintain
Load balancing	No need for load balancer, single point is responsible for handling all requests	Load balancer needed to distribute requests evenly
Downtime vulnerability	Hight vulnerability	Low vulnerability – even if one device is down, others might still work properly and handle all requests
Machine strain	Machine is very stressed by all requests	All stress is distributed among all machines

Vertical vs Horizontal scaling – single point of failure



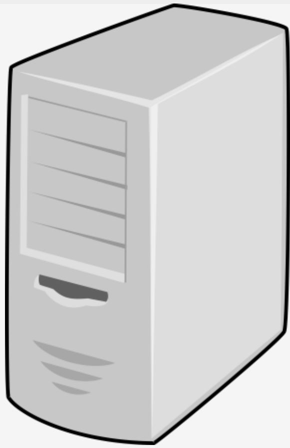
When talking about vertical and horizontal scaling we have to emphasize that using single machine platform might be hazardous. If by any chance our device crashes, than our website will be down and it might be difficult to recover from that.

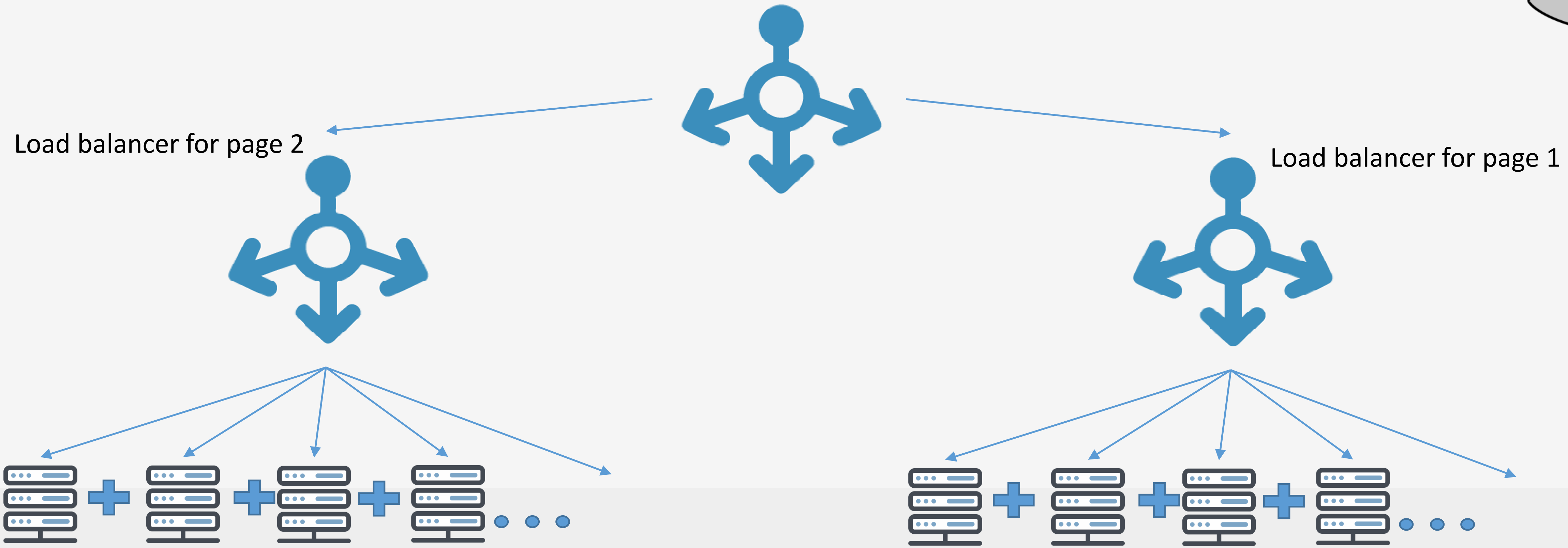
In horizontal scaling we take less risk because if any device should fail, than others might handle it's load. In pessimistic scenario our load balancer might crash, but we can still prepare for that and prepare for that and create infrastructure with many balancers:



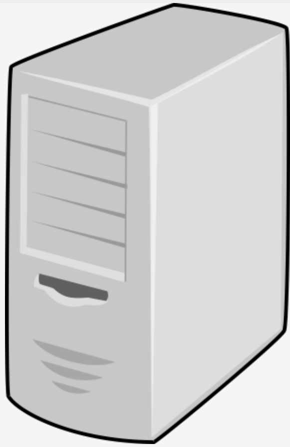
Vertical vs Horizontal scaling – single point of failure

This way if one of our devices crashes, it can only disable some functionality of our service. There are many other ways to create infrastructure, but the are all focused on reducing SPOF (Single Point Of Failure).





Different types of services



In many cases it will be best to use existing infrastructure and focus on programming.

Let’s say we are the company focusing on the software development and we don’t want to be responsible for servers and the infrastructure.

Different types of services

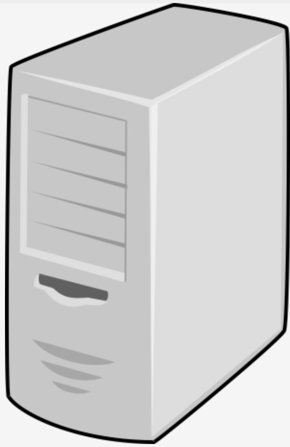
In many cases it will be best to use existing infrastructure and focus on programming.

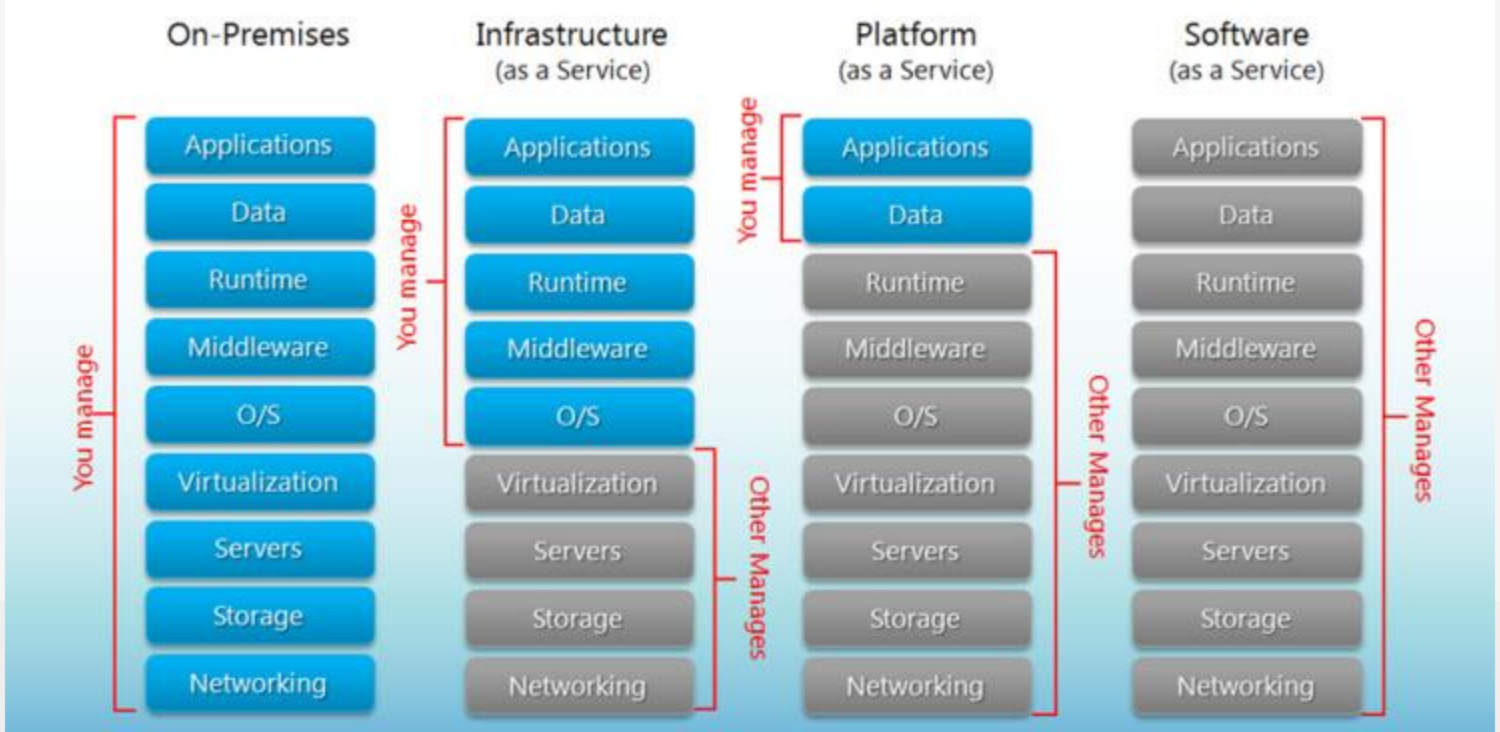
Let's say we are the company focusing on the software development and we don't want to be responsible for servers and the infrastructure.

In first case we want to buy the infrastructure, which means Visualization, Servers, Storage and Networking are the things that are provided for us, and the only thing we need to do is to maintain software and OS. In other words – we are borrowing somones hardware and using it for our own purposes.

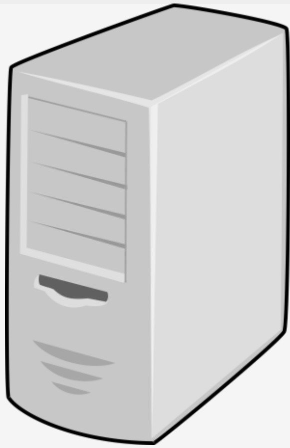
This model is called IaaS

– Infrastructure as a service.





Different types of services



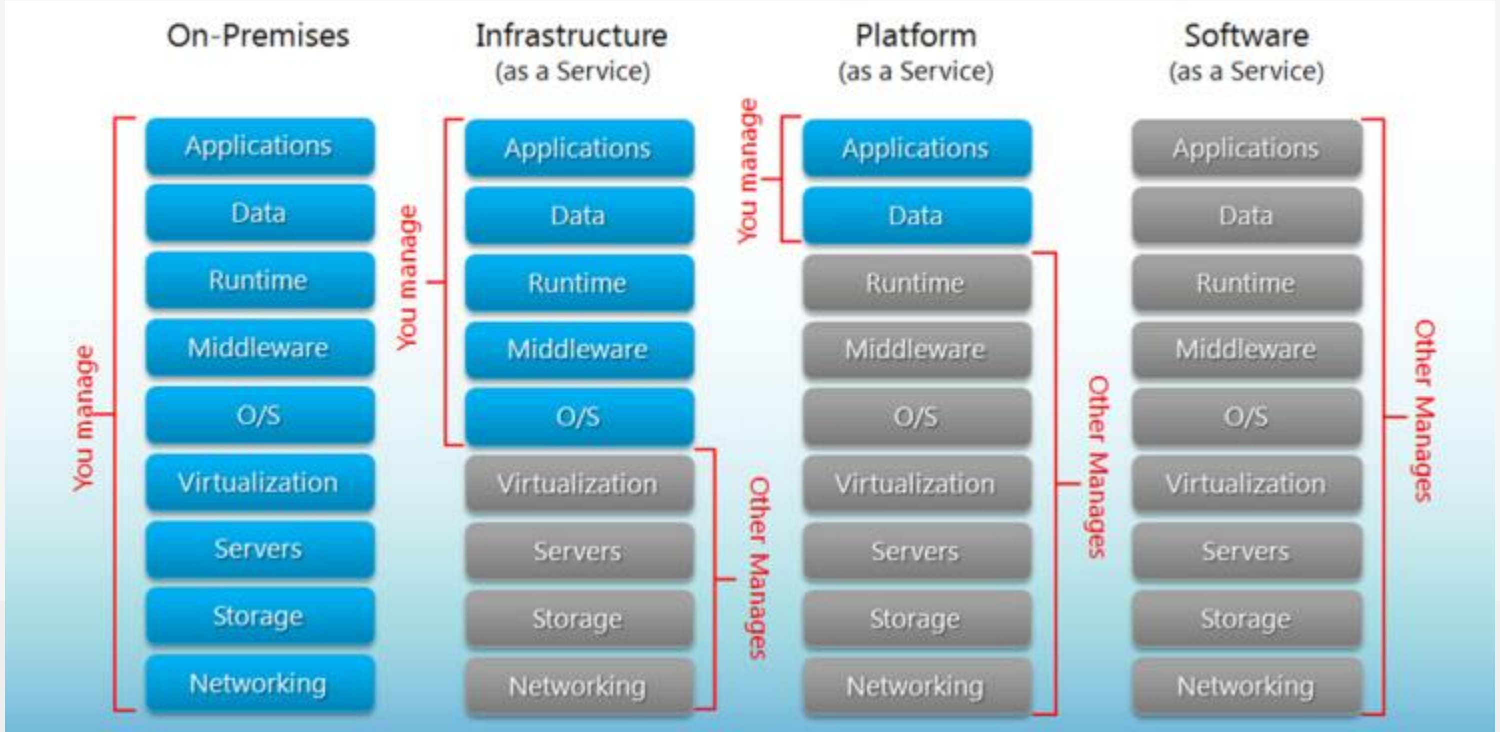
So we have hardware and infrastructure but still we don’t want (or don’t know how) to handle platform – which is Operating System, Middleware and Runtime.

In this case we can also buy it as a service. Many companies deliver solutions in which we only define platform to which we want to deploy our software. This is very commonly used model.

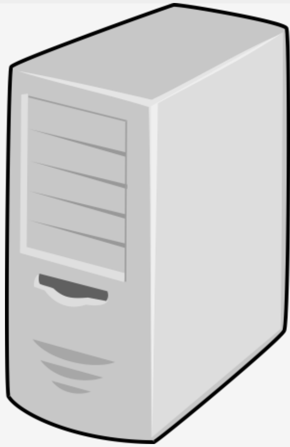
The only thing we need to provide to run our application in this model is our software and the data.

This model is called PaaS

– Platform as a service.



Different types of services

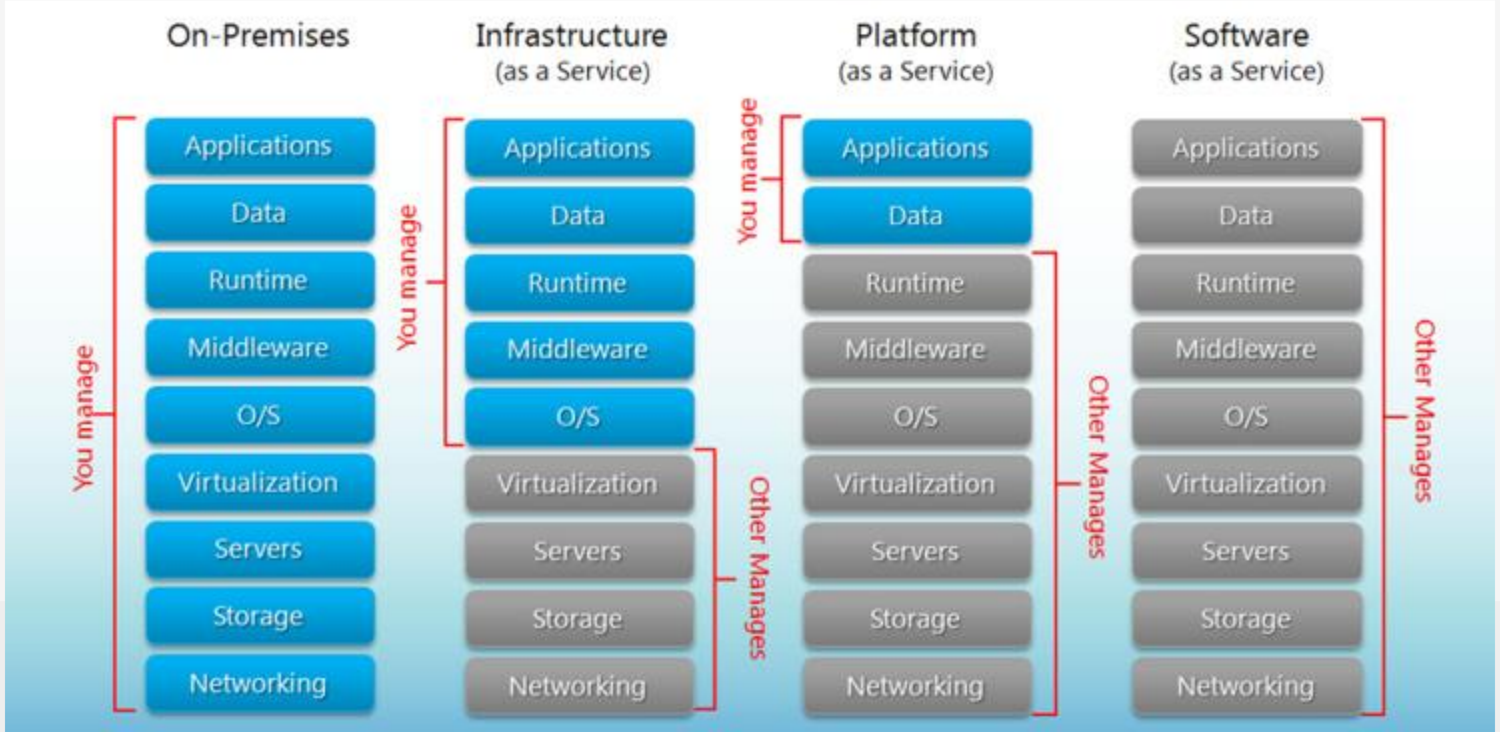


Last model is the one delivered to the client. When every previous layer is combined and we add software to it, we have a working software on a platform. If we're providing this type of solution, we're deliverying software services to the client.

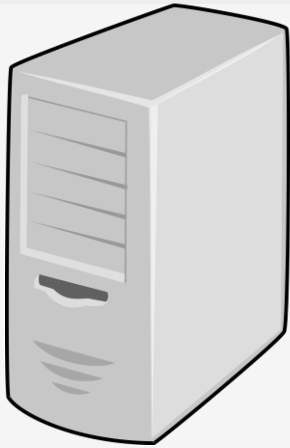
Client can reach his solution using his computer. (for example using his browser)

This model is called SaaS

– Software as a service.



Different types of services - clouds



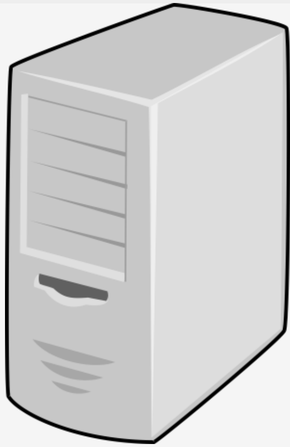
Clouds are services provided by a third party companies working in WAN (Wide Area Network).

Let’s say we want a provider to host our website. We can choose one of the services described earlier, but we can also buy a Cloud network service, which will than allow us to run our application at a providers network. In that network all components, devices and services of our application will be provided, but other than that, we have amazing stability.

All devices inside Cloud network share resources and copy of our website can be hosted on multiple machines. Automatic backups will give us more reliablity and more stability. We can then be sure that even if one device fails, the provider will still be able to host our service inside his infrastructure.

We can choose which service we want to buy (IaaS, Paas, Saas) and than run our software on a very stable virtualized service on a cloud.

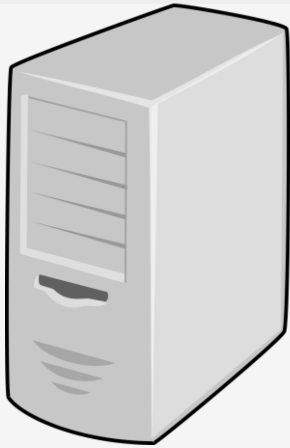
Application servers



From many implementations of application servers we'll focus on few most commonly used:

- **Tomcat** – (open-source) web application server, developed by Apache Software Foundation. Tomcat implements several Java EE specifications including Servlets, JSP, Java EL and WebSocket. All of it is provided by „pure Java" environment for Java code. Tomcat uses Catalina container. Tomcat itself is very commonly called *Http server and Servlet Container.* Great advantage of Tomcat is it's flexibility and stability. Failing a single page will not crash whole server. Tomcat has light footprint (~70MB), so it's very light and easy to manage. (FREE)
- **Glassfish** – (open-source) application server started by Sun Microsystems for Java EE. It is now sponsored by Oracle Corporation. Glassfish is better solution for Java EE enterprise Applications, because it is Application Server, which can also be used as a Web Server (it can Handle HTTP requests, servlets, JSP's etc.). GlassFish is a complete Java EE application server (implements all specifications). (FREE, there is a paying option)
- **WildFly/Jboss** – (open-source, subscription based) application server implementing all Java EE functionalities. It is maintained by Red Hat. It is mostly used for commercial implementations. JBoss is the best choice for applications where developers need full access to the functionality that the Java Enterprise Edition provides and are happy with the default implementations of that functionality that ship with it. If you don't need the full range of JEE features, then choosing JBoss will add a lot of complexity to deployment and resource overhead that will go unused. For example, the JBoss installation files are around an order of magnitude larger than Tomcat's.

Application servers

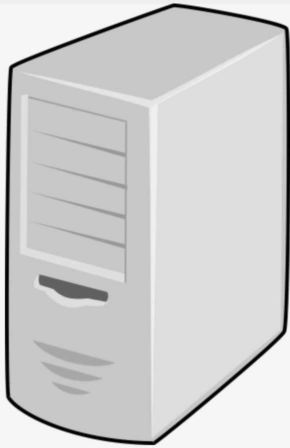


Other implementations:

- **WebLogic** – paid application server developed by Oracle.

- **WebSphere** – paid application server developed by IBM.

Application servers

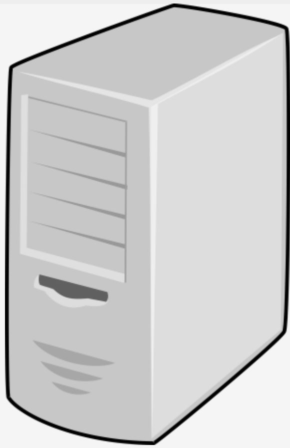


Application servers are designed to run multiple applications simultaneously. We can choose application using context, which is part of the URL. For example:

http://localhost:8080/**application1**/main.jsp

http://localhost:8080/**application2**/otherMain.jsp

Application servers



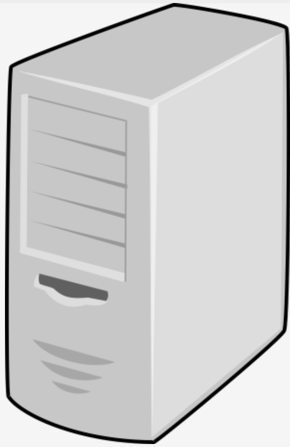
Application servers are designed to run multiple applications simultaneously. We can choose application using context, which is part of the URL. For example:

http://localhost:8080/**application1**/main.jsp

http://localhost:8080/**application2**/otherMain.jsp

As You can see, two links above will run two different applications from two different context. Context name can be set, or in some cases it is given by the application name (from it's executable ‚*.war' file).

Application servers – running applications

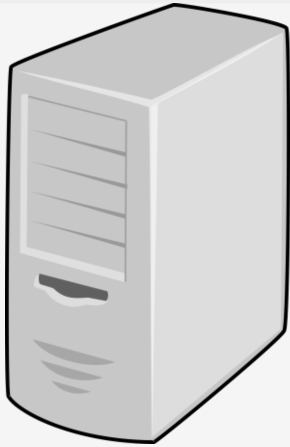


To run application on a server You need to compile Your code into WAR file. In some servers You just need to move copy of Your WAR file into application server folder (for example in Tomcat it is /webapps folder).

You can also log into the Tomcat Management application (web application) and upload it using the manager.

Depending on the server we're using, deployment process is always well documented on the website.

Application servers – WAR files



WAR (Web Application Archive or Web Application Resource) files are archives containing all files required to run application on a web server. They contain all Java Server Pages, Servlets, Java classes, xml files and static web pages. (source https://en.wikipedia.org/wiki/WAR_(file_format)). WAR Files are digitally signed same way as JAR files. In WAR Files the **WEB-INF/** folder contains **web.xml** file in which definition of a web application is held.

Few main folders of WAR Archive:

- **/** - root directory containing all public files of that archive
- **/WEB-INF** – all application resources like JSP, graphics, scripts, css, html files
- **/lib** – libraries of our project
- **/META-INF** – configuration files