A) BASICS
A.1) CONTAINERS

https://www.cio.com/article/247005/what-are-containers-and-why-do-you-need-them.html
https://searchitoperations.techtarget.com/definition/container-containerization-or-container-based-virtualization

Why containers ?
Containers are a solution to the problem of how to get software to run reliably when moved from one computing environment to another. This could be from a developer's laptop to a test environment, from a staging environment into production, and perhaps from a physical machine in a data center to a virtual machine in a private or public cloud.

Containers are a type of software that can virtually package and isolate applications for deployment.

What does a container consist of ?
Put simply, a container consists of an entire runtime environment: an application, plus all its dependencies, libraries and other binaries, and configuration files needed to run it, bundled into one package. By containerizing the application platform and its dependencies, differences in OS distributions and underlying infrastructure are abstracted away.

Containers hold the components necessary to run desired software. These components include files, environment variables, dependencies and libraries. The host OS constrains the container's access to physical resources, such as CPU, storage and memory, so a single container cannot consume all of a host's physical resources.

VM vs containers
With virtualization technology, the package that can be passed around is a virtual machine, and it includes an entire operating system as well as the application. A physical server running three virtual machines would have a hypervisor and three separate operating systems running on top of it.

By contrast a server running three containerized applications with Docker runs a single operating system, and each container shares the operating system kernel with the other containers. Shared parts of the operating system are read only, while each container has its own mount (i.e., a way to access the container) for writing. That means the containers are much more lightweight and use far fewer resources than virtual machines.

Containers can share access to an operating system (OS) kernel without the traditional need for virtual machines (Vms).

VMs can require substantial resource overhead, such as memory, disk and network input/output (I/O), because each VM runs an OS. This means VMs can be large and take longer to create than containers. Because containers share the OS kernel, only one instance of an OS can run many isolated containers. The OS supporting containers can also be smaller, with fewer features, than an OS for a VM or physical application installation.

Because containers share the same OS kernel as the host, containers can be more efficient than VMs, which require separate OS instances.

Advantages of containers
A benefit is that containerization allows for greater modularity. Rather than run an entire complex application inside a single container, the application can be split in to modules (such as the database, the application front end, and so on). This is the so-called microservices approach. Applications built in this way are easier to manage because each module is relatively simple, and changes can be made to modules without having to rebuild the entire application. Each service operates on the same OS while staying individually isolated. Each service can scale up and down to respond to demand. Cloud infrastructure is designed for this kind of elastic, unlimited scaling.

Container images
A container image is a static file with executable code that can create a container on a computing system. A container image is immutable—meaning it cannot be changed, and can be deployed consistently in any environment. It is a core component of a container architecture

Container images include everything a container needs to run—the container engine such as Docker or CoreOS, system libraries, utilities, configuration settings, and specific workloads that should run on the container. The image shares the operating system kernel of the host, so it does not need to include a full operating system.

A container image is composed of layers, added on to a parent image (also known as a base image). Layers

make it possible to reuse components and configurations across images. Constructing layers in an optimal manner can help reduce container size and improve performance.

Container images include the information that executes at runtime on the OS, via a container engine. Containers are inherently stateless and do not retain session information, although they can be used for stateful applications.

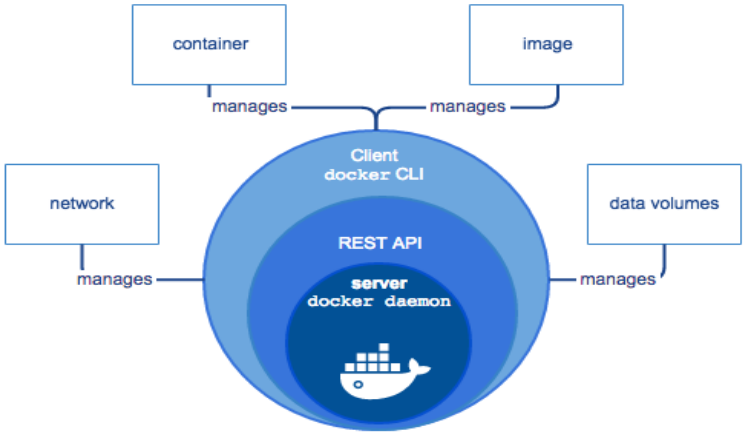
Multiple instances of a container image can run simultaneously, and new instances can replace failed ones without disruption to the application's operation.

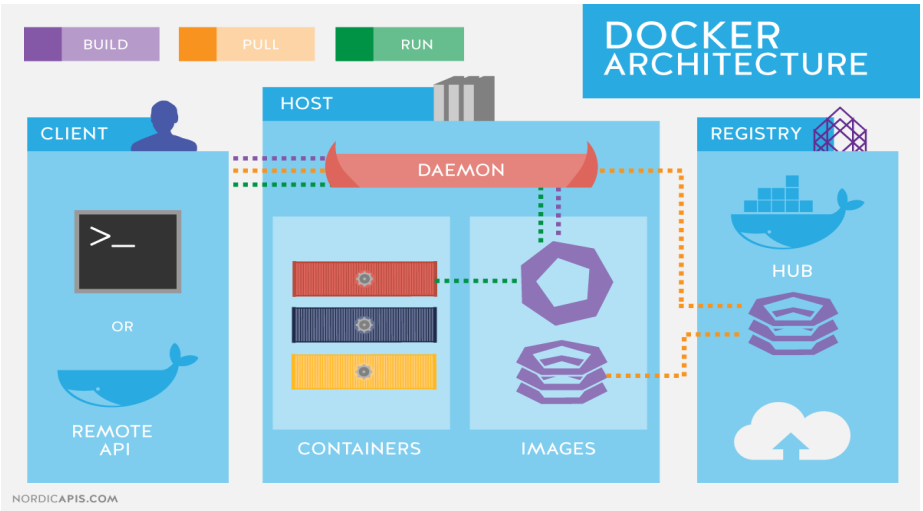
A.2)DOCKER

https://jfrog.com/knowledge-base/a-beginners-guide-to-understanding-and-building-docker-images/
https://docs.docker.com/get-started/overview/
https://docker-curriculum.com/

Docker architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.





Docker CLI

A command line interface client for interacting with the Docker daemon.

REST API
An API used by applications to interact with the Docker daemon; it can be accessed by an HTTP client.

Docker daemon
Docker daemon is a persistent background process that manages the containers on a single host. It is a self-sufficient runtime that manages Docker objects such as images, containers, network, and storage. Docker daemon listens for REST API requests and performs a series of container operations accordingly. Applications or users typically use Docker clients to authenticate and interact with Docker daemons. A Docker daemon can also communicate with other Docker daemons if multiple hosts are managed as a service or a cluster.

Docker client
The Docker client enables users to interact with Docker. The Docker client can reside on the same host as the daemon or connect to a daemon on a remote host. A docker client can communicate with more than one daemon. The Docker client provides a command line interface (CLI) that allows you to issue build, run, and stop application commands to a Docker daemon.

The main purpose of the Docker Client is to provide a means to direct the pull of images from a registry and to have it run on a Docker host.

docker build
docker pull
docker run

Docker Host
The Docker host provides a complete environment to execute and run applications. It comprises of the Docker daemon, Images, Containers, Networks, and Storage. As previously mentioned, the daemon is responsible for all container-related actions and receives commands via the CLI or the REST API. It can also communicate with other daemons to manage its services. The Docker daemon pulls and builds container images as requested by the client. Once it pulls a requested image, it builds a working model for the container by utilizing a set of instructions known as a build file. The build file can also include instructions for the daemon to pre-load other components prior to running the container, or instructions to be sent to the local command line once the container is built.

Docker image
A Docker image is a collection of files, including binaries, source code and other dependencies, needed to deploy a container environment. In Docker, there are two ways to create an images:

- Dockerfile—Docker provides a simple, human-readable configuration file that specifies what a Docker image should contain.
- Create an image from an existing container—you can run a container from an existing image, modify the container environment, and save the result as a new image.

A Docker container image describes a container environment. A Docker container is an instance of that environment, running on Docker Engine. You can run multiple containers from the same image, and all of them will contain the same software and configuration, as specified in the image.

Image layers
Each of the files that make up a Docker image is known as a layer. These layers form a series of intermediate images, built one on top of the other in stages, where each layer is dependent on the layer immediately below it. The hierarchy of your layers is key to efficient lifecycle management of your Docker images. Thus, you should organize layers that change most often as high up the stack as possible. This is because, when you make changes to a layer in your image, Docker not only rebuilds that particular layer, but all layers built from it. Therefore, a change to a layer at the top of a stack involves the least amount of computational work to rebuild the entire image.

Container layer
When a container runs, Docker adds a readable/writable top layer over the static image layers. This top layer is used by the container to modify files during runtime, and can also be used to customize the container. This way, multiple containers created from the same image can have different data.

Parent vs base image
There is a subtle technical different between parent and base images:

- A base image is an empty container image, which allows advanced users to create an image from scratch.
- A parent image is a pre-configured image that provides some basic functionality, such as a stripped-down Linux system, a database such as MySQL or PostgreSQL, or a content management system such as WordPress.

Dockerfile method

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

Containers

Containers are encapsulated environments in which you run applications. The container is defined by the image and any additional configuration options provided on starting the container, including and not limited to the network connections and storage options. Containers only have access to resources that are defined in the image, unless additional access is defined when building the image into a container. You can also create a new image based on the current state of a container.

Networking

Docker implements networking in an application-driven manner and provides various options while maintaining enough abstraction for application developers. There are basically two types of networks available – the default Docker network and user-defined networks. By default, you get three different networks on the installation of Docker – none, bridge, and host. The none and host networks are part of the network stack in Docker. The bridge network automatically creates a gateway and IP subnet and all containers that belong to this network can talk to each other via IP addressing. This network is not commonly used as it does not scale well and has constraints in terms of network usability and service discovery.

Storage

You can store data within the writable layer of a container but it requires a storage driver. Being non-persistent, it perishes whenever the container is not running. Moreover, it is not easy to transfer this data. In terms of persistent storage, Docker offers four options:

- **Data Volumes**: Data Volumes provide the ability to create persistent storage, with the ability to rename volumes, list volumes, and also list the container that is associated with the volume. Data Volumes sit on the host file system, outside the containers copy on write mechanism and are fairly efficient.
- **Data Volume Container**: A Data Volume Container is an alternative approach wherein a dedicated container hosts a volume and to mount that volume to other containers. In this case, the volume container is independent of the application container and therefore can be shared across more than one container.
- **Directory Mounts**: Another option is to mount a host's local directory into a container. In the previously mentioned cases, the volumes would have to be within the Docker volumes folder, whereas when it comes to Directory Mounts any directory on the Host machine can be used as a source for the volume.

Any file changes that are made within a container are reflected as a copy of modified data from the read-only layer. The version in the read-write layer hides the underlying file, but does not remove it. When deleting a container, the read-write layer containing the changes are destroyed, and upon new container creation those changes are not reflected – gone forever!
Intro Docker data volumes – the ability to persist data in an organized way outside of the container. A data volume is a specially-designated directory within one or more containers that bypasses the Union File System. Data volumes provide several useful features for persistent or shared data (from the Docker User Guide):

Data volumes are designed to persist data, independent of the container's lifecycle. Docker therefore never automatically deletes volumes when you remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container.

A.3)DOCKER COMPOSE

https://docs.docker.com/compose/
https://gabrieltanner.org/blog/docker-compose

Definition

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

Docker Compose is used for running multiple containers as a single service. Each of the containers here run in isolation but can interact with each other when required. Docker Compose files are very easy to write in a scripting language called YAML, which is an XML-based language that stands for Yet Another Markup Language. Another great thing about Docker Compose is that users can activate all the services (containers) using a single command.

Benefits
->Single host deployment - This means you can run everything on a single piece of hardware
->Quick and easy configuration - Due to YAML scripts

->High productivity - Docker Compose reduces the time it takes to perform tasks
->Security - All the containers are isolated from each other, reducing the threat landscape

Three steps

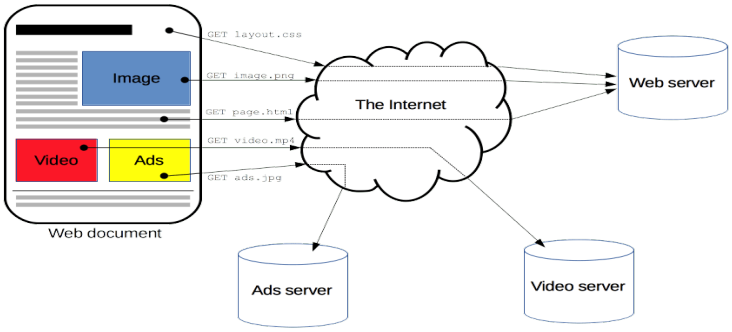
1) Define your app's environment with a Dockerfile so it can be reproduced anywhere.
2) Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
3) Run docker compose up and the Docker compose command starts and runs your entire app.

A.4)HTTP/HTTPS

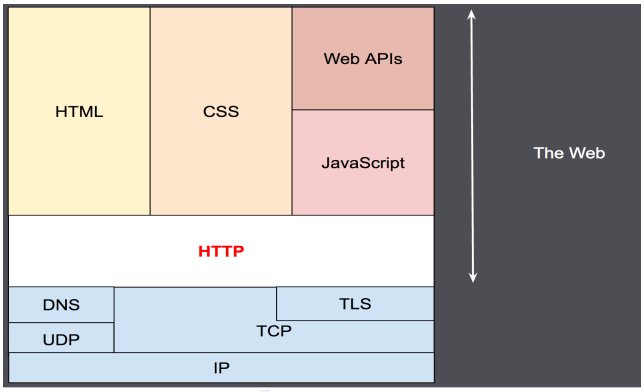
Hypertext Transfer Protocol (HTTP) is an application layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers-client and web servers-server, but it can also be used for other purposes. HTTP follows a classical client server model (Client-server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers and service requesters, called clients), with a client opening a connection to make a request, then waiting until it receives a response. HTTP is a stateless protocol, meaning that the server does not keep any data (state) between two requests.

Overview

HTTP is a protocol for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. A complete document is reconstructed from the different sub-documents fetched, for instance, text, layout description, images, videos, scripts, and more.



Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client, usually a Web browser, are called *requests* and the messages sent by the server as an answer are called *responses*.



Components of HTTP-based systems

HTTP is a client-server protocol: requests are sent by one entity, the user-agent (or a proxy on behalf of it). Most of the time the user-agent is a Web browser, but it can be anything, for example, a robot that crawls the Web to populate and maintain a search engine index.

Each individual request is sent to a server, which handles it and provides an answer called the response. Between the client and the server there are numerous entities, collectively called proxies, which perform different operations and act as gateways for example.

n reality, there are more computers between a browser and the server handling the request: there are routers, modems, and more. Thanks to the layered design of the Web, these are hidden in the network and transport layers. HTTP is on top, at the application layer. Although important for diagnosing network problems, the underlying layers are mostly irrelevant to the description of HTTP.

Client : the user agent

The user-agent is any tool that acts on behalf of the user. This role is primarily performed by the Web browser, but it may also be performed by programs used by engineers and Web developers to debug their applications. The browser is always the entity initiating the request. It is never the server (though some mechanisms have been added over the years to simulate server-initiated messages).

To display a Web page, the browser sends an original request to fetch the HTML document that represents the page. It then parses this file, making additional requests corresponding to execution scripts, layout information (CSS) to display, and sub-resources contained within the page (usually images and videos). The Web browser then combines these resources to present the complete document, the Web page. Scripts executed by the browser can fetch more resources in later phases and the browser updates the Web page accordingly.

A Web page is a hypertext document. This means some parts of the displayed content are links, which can be activated (usually by a click of the mouse) to fetch a new Web page, allowing the user to direct their user-agent and navigate through the Web. The browser translates these directions into HTTP requests, and further interprets the HTTP responses to present the user with a clear response.

The webserver

On the opposite side of the communication channel is the server, which serves the document as requested by the client. A server appears as only a single machine virtually; but it may actually be a collection of servers sharing the load (load balancing), or a complex piece of software interrogating other computers (like cache, a DB server, or e-commerce servers), totally or partially generating the document on demand.
A server is not necessarily a single machine, but several server software instances can be hosted on the same machine. With HTTP/1.1 and the Host header, they may even share the same IP address.

Stateless but not sessionless

HTTP is stateless: there is no link between two requests being successively carried out on the same connection. This immediately has the prospect of being problematic for users attempting to interact with certain pages coherently, for example, using e-commerce shopping baskets. But while the core of HTTP itself is stateless, HTTP cookies allow the use of stateful sessions. Using header extensibility, HTTP Cookies are added to the workflow, allowing session creation on each HTTP request to share the same context, or the same state.

Http and connections

A connection is controlled at the transport layer, and therefore fundamentally out of scope for HTTP. HTTP doesn't require the underlying transport protocol to be connection-based; it only requires it to be reliable, or not lose messages (at minimum, presenting an error in such cases). Among the two most common transport protocols on the Internet, TCP is reliable and UDP isn't. HTTP therefore relies on the TCP standard, which is connection-based.

Before a client and server can exchange an HTTP request/response pair, they must establish a TCP connection, a process which requires several round-trips. The default behavior of HTTP/1.0 is to open a separate TCP connection for each HTTP request/response pair. This is less efficient than sharing a single TCP connection when multiple requests are sent in close succession.

In order to mitigate this flaw, HTTP/1.1 introduced pipelining (which proved difficult to implement) and persistent connections: the underlying TCP connection can be partially controlled using the Connection header. HTTP/2 went a step further by multiplexing messages over a single connection, helping keep the connection warm and more efficient.

HTTP flow

When a client wants to communicate with a server, either the final server or an intermediate proxy, it performs the following steps:

- Open a TCP connection: The TCP connection is used to send a request, or several, and receive an answer. The client may open a new connection, reuse an existing connection, or open several TCP connections to the servers.
- Send an HTTP message: HTTP messages (before HTTP/2) are human-readable. With HTTP/2, these simple messages are encapsulated in frames, making them impossible to read directly, but the principle remains the same.
- Read the response sent by the server
- Close or reuse the connection for further requests.

A.5) LOCAL HOST

What is a loopback address?

IP addresses can identify individual servers on the internet, as well as devices outside of the internet on local networks. Whenever a new networked device is created, it gets an IP address.

However, some IP addresses are reserved for certain reasons. For instance, all addresses beginning with the number “127” are special IP addresses called “local loopback addresses.” Instead of identifying another device on the internet, a loopback address references a device on your private, local network. This is why no website may have an IP address starting with “127.”

Loopback addresses can’t be reached by outside devices. When you send a request to a loopback address, this triggers a loopback, meaning the request is sent back to the server it came from. As a result, loopbacks don’t go through the internet — they stay in your local network.
Now that we understand IP addresses and loopbacks, we can turn our attention back to localhost.

What does localhost mean?

In a computer network, localhost is a hostname that refers to the computer that is executing a program — you can think of it as meaning “this computer.” The term is used when making a loopback request to one’s own device. These types of requests are useful for testing and security reasons, as we’ll see later.
Usually, you can access the localhost of any computer through the loopback address 127.0.0.1. By default, this IP address references a server running on the current device. In other words, when your computer requests the IP address 127.0.0.1, it’s making a request to itself, its "local host."

The term “localhost” also serves as the domain name for the loopback IP address 127.0.0.1, sort of like how “hubspot.com” stands in for the IP address 104.19.154.83. There’s an important difference, though: If you put “localhost” into your browser bar, your request won’t go through the internet. Instead, you’ll cause a loopback and the request will end back at your computer.

No matter what device you’re using, a request to 127.0.0.1 or “localhost” will be sent back to the same device you’re working on. This doesn’t require any special authorization or equipment — the computer's operating system comes with the ability to act as a server and field loopback requests.
One more quick note about localhost: 127.0.0.1 is the default IP address for localhost in IPv4. In IPv6, the default localhost address is ::1.

A.6)SOCKETS/PORTS

A *socket* can be thought of as an endpoint in a two-way communication channel. Socket routines create the communication channel, and the channel is used to send data between application programs either locally or over networks. Each socket within the network has a unique name associated with it called a *socket descriptor*—a fullword integer that designates a socket and allows application programs to refer to it when needed.

A socket is one endpoint of a two way communication link between two programs running on the network. The socket mechanism provides a means of inter-process communication (IPC) by establishing named contact points between which the communication take place.

Socket are generally employed in client server applications. The server creates a socket, attaches it to a network port addresses then waits for the client to contact it. The client creates a socket and then attempts to connect to the server socket. When the connection is established, transfer of data takes place.

When the client starts a socket call, a socket connection is made between an application on the client and an application on the server.

Another analogy used to describe socket communication is a telephone conversation. Dialing a phone number from your telephone is similar to starting a socket call. The telephone switching unit knows where to logically make the correct switch to complete the call at the remote location. During your telephone conversation, this connection is present and information is exchanged. After you hang up, the connection is broken and you must start it again. The client uses the socket() function call to start the logical switch mechanism to connect to the server.

A socket connecting to the network is created at each end of the communication. Each socket has a specific address. This address is composed of an IP address and a port number.

What is a port ?

Port is a part of the transport layer and helps in network communication. A port is a logical identifier assigned to a process in order to identify that process uniquely in a network system. When two network devices communicate, they do so by sending packets to each other. Each packet received by a receiver device contains a

port number that uniquely identifies the process where the packet needs to be sent.
Not all the network protocol uses a port for communication. For example, ICMP doesn't use a port. On the other hand, protocols like TCP, UDP, HTTP utilize a port for communication.

Port VS Socket
We now know the basics of port and socket. Let's see the differences between a port and a socket:

Port	Socket
Port specifies a number that is used by a program in a computer.	A socket is a combination of IP address and port number.
A program running on different computers can use the same port number. Hence port numbers can't be used to identify a computer uniquely.	It identifies a computer as well as a program within the computer uniquely.
Port number is used in the transport layer.	Sockets are involved in the application layer. A socket is an interface between the transport and application layer.
Port uses a socket to drop the data to a correct application.	A server and a client uses a socket to keep an eye on the data request and responses.

A.7)NGINX

https://www.youtube.com/watch?v=JKxlsvZXG7c
http://nginx.org/en/docs/beginners_guide.html

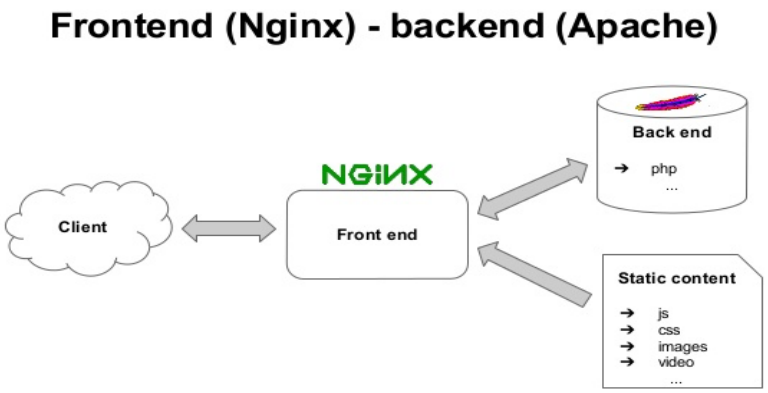
NGINX is open source software for web serving, reverse proxying, caching, load balancing, media streaming, and more. It started out as a web server designed for maximum performance and stability. In addition to its HTTP server capabilities, NGINX can also function as a proxy server for email (IMAP, POP3, and SMTP) and a reverse proxy and load balancer for HTTP, TCP, and UDP servers.

The goal behind NGINX was to create the fastest web server around. Since the original release of NGINX, however, websites have expanded from simple HTML pages to dynamic, multifaceted content.

NGINX is one of the preferred web servers by many websites dealing with high traffic due to its ability to handle massive connections with astonishing speed. It comes into the market as a significant competitor to the Apache HTTP server.

It does not allow you to allocate a process to a particular connection, but it creates a process pool that can be easily shared among multiple connections within the network. Whenever a request is made, a resource will be allocated to the process resulting in better resource utilization that can easily handle extensive connections.

NGINX also helps in setting up a secured connection between your data-centers and the outside network.

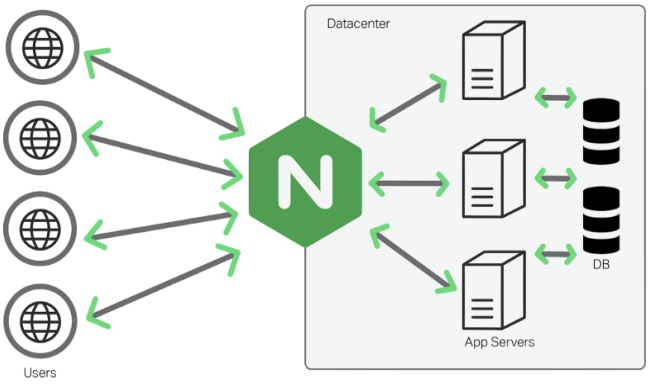


Concurrency and faster loading times have always been a challenge for any high traffic website. In order to load sites faster, browsers open multiple connections to a web server to load content in parallel. Combined with persistent connections, the web server needs to be really robust and be able to scale non-linearly with the number of requests.

Nginx processes connections as modules piped to each other, as a chain. For every operation there's a module which is doing the relevant work, e.g. compression, modifying content, executing server-side includes, and communicating to the upstream application servers. A typical HTTP request processing cycle looks like the following:

1. Client sends HTTP request.
2. Nginx core chooses the appropriate phase handler based on the configured location matching the request.
3. If configured to do so, a load balancer picks an upstream server for proxying.
4. Phase handler does its job and passes each output buffer to the first filter.
5. First filter passes the output to the second filter.
6. Second filter passes the output to the third (and so on).
7. Final response is sent to the client.

This layered and loosely coupled but highly cohesive approach makes it really good at what it does.



A.8)WORDPRESS + PHP FPM / FAST CGI

Wordpress

WordPress (WP, WordPress.org) is a free and open-source content management system (CMS) written in PHP and paired with a MySQL or MariaDB database.

"WordPress is a factory that makes webpages" is a core analogy designed to clarify the functions of WordPress: it stores content and enables a user to create and publish webpages, requiring nothing beyond a domain and a hosting service.

WordPress has a web template system using a template processor. Its architecture is a front controller, routing all requests for non-static URIs to a single PHP file which parses the URI and identifies the target page.

https://danielmiessler.com/study/difference-between-uri-url/

PHP

PHP is a general-purpose scripting language geared towards web development.

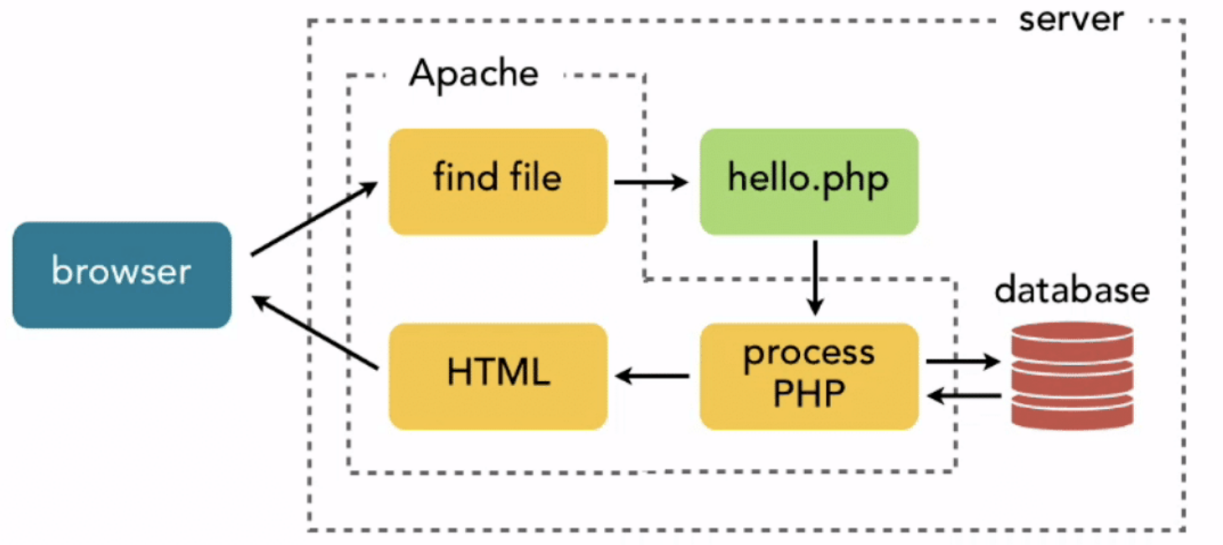
PHP code is usually processed on a web server by a PHP interpreter implemented as a module, a daemon or as a Common Gateway Interface (CGI) executable. On a web server, the result of the interpreted and executed PHP code – which may be any type of data, such as generated HTML or binary image data – would form the whole or part of an HTTP response. Various web template systems, web content management systems, and web frameworks exist which can be employed to orchestrate or facilitate the generation of that response.

PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.

PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.

- •PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
- •You add, delete, modify elements within your database through PHP.
- •Access cookies variables and set cookies.
- •Using PHP, you can restrict users to access some pages of your website.

- •It can encrypt data.



FAST CGI

FastCGI is a binary protocol for interfacing interactive programs with a web server. It is a variation on the earlier Common Gateway Interface (CGI). FastCGI's main aim is to reduce the overhead related to interfacing between web server and CGI programs, allowing a server to handle more web page requests per unit of time. Instead of creating a new process for each request, FastCGI uses persistent processes to handle a series of requests. These processes are owned by the FastCGI server, not the web server.

To service an incoming request, the web server sends environment variable information and the page request to a FastCGI process over either a Unix domain socket, a named pipe, or a Transmission Control Protocol (TCP) connection. Responses are returned from the process to the web server over the same connection, and the web server then delivers that response to the end user. The connection may be closed at the end of a response, but both web server and FastCGI service processes persist.

Each individual FastCGI process can handle many requests over its lifetime, thereby avoiding the overhead of per-request process creation and termination. Processing multiple requests concurrently can be done in several ways: by using one connection with internal multiplexing (i.e., multiple requests over one connection); by using multiple connections; or by a mix of these methods. Multiple FastCGI servers can be configured, increasing stability and scalability.

Web site administrators and programmers can find that separating web applications from the web server in FastCGI has many advantages over embedded interpreters (mod_perl, mod_php, etc.). This separation allows server and application processes to be restarted independently – an important consideration for busy web sites. It also enables the implementation of per-application, hosting service security policies, which is an important requirement for ISPs and web hosting companies.Different types of incoming requests can be distributed to specific FastCGI servers which have been equipped to handle those types of requests efficiently.

PHP FPM
https://geekflare.com/fr/php-fpm-optimization/

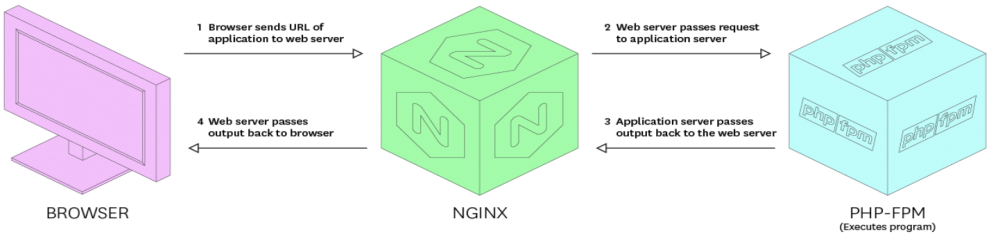
PHP-FPM (an acronym of FastCGI Process Manager) is a hugely-popular alternative PHP (Hypertext Processor) FastCGI implementation.

As you may or may not know, PHP is one of the biggest open-source software programming languages utilized online. It features heavily in web development across such well-known platforms as Drupal, Magento, and Wordpress.

PHP-FPM includes numerous features that can prove beneficial for websites receiving traffic in large volumes frequently. These are:

- •Ability to start workers using various uid/gid/chroot/environment and php.ini, which replaces the safe mode users may expect
- •In-depth management for simple stop/start processing
- •Logging of stdout and stderr
- •Emergency restart available, in the event of an opcode cache being destroyed accidentally
- •Support for uploads is faster
- •Based on php.ini configuration files
- •Slowlog variable configuration for detecting functions that take longer than usual to execute
- •FastCGI improvements, with a special function for stopping and downloading data while completing long processes (e.g. processing statistics)

PHP-FPM and Nginx



Nginx the ideal combination with PHP-FPM. Why? Because it's a stable web server recognized for its impressive performance and low resource-consumption.

It features an asynchronous structure that's highly-scalable, according to events. On top of this, memory consumption performance is significantly better when using Nginx and PHP-FPM together.

PHP runs as an isolated service when you use PHP-FPM. Employing this PHP version as the language interpreter means requests will be processed via a TCP/IP socket, and the Nginx server handles HTTP requests only, while PHP-FPM interprets the PHP code. Taking advantage of two separate services is vital to become more efficient.

PHP-FPM and Wordpress
An Nginx server with PHP-FPM support is crucial if you operate an online newspaper, content platform, or WordPress site receiving a huge number of visits daily. This set up enables you to facilitate the execution of your WordPress CMS's PHP code to a higher standard.

A.9)MYSQL

In regard to the general definition, MySQL is an open source relational database management system (RDBMS) with a client-server model. RDBMS is a software or service used to create and manage databases based on a relational model.

Database

A database is simply a collection of structured data. Think of taking a selfie: you push a button and capture an image of yourself. Your photo is data, and your phone's gallery is the database. A database is a place in which data is stored and organized. The word "relational" means that the data stored in the dataset is organized as tables. Every table relates in some ways. If the software doesn't support the relational data model, just call it DBMS.

SQL

MySQL and SQL are not the same. Be aware that MySQL is one of the most popular RDBMS software's brand names, which implements a client-server model. So, how do the client and server communicate in an RDBMS environment? They use a domain-specific language – Structured Query Language (SQL). If you ever encounter other names that have SQL in them, like PostgreSQL and Microsoft SQL server, they are most likely brands which also use Structured Query Language syntax. RDBMS software is often written in other programming languages, but always use SQL as their primary language to interact with the database. MySQL itself is written in C and C++.

History aside, SQL tells the server what to do with the data. It is similar to your WordPress password or code. You input it into the system to gain access to the dashboard area. In this case, SQL statements can instruct the server to perform certain operations:

- Data query: requesting specific information from the existing database.
- Data manipulation: adding, deleting, changing, sorting, and other operations to modify the data, the values or the visuals.
- Data identity: defining data types, e.g. changing numerical data to integers. This also includes defining a schema or the relationship of each table in the database
- Data access control: providing security techniques to protect data, this includes deciding who can view or use any information stored in the database

One or more devices (clients) connect to a server through a specific network. Every client can make a request from the graphical user interface (GUI) on their screens, and the server will produce the desired output, as long as both ends understand the instruction. Without getting too technical, the main processes taking place in a

MySQL environment are the same, which are:

1. MySQL creates a database for storing and manipulating data, defining the relationship of each table.
2. Clients can make requests by typing specific SQL statements on MySQL.
3. The server application will respond with the requested information and it will appear on the clients' side.

What is MySQL?

MySQL is a relational database management system (RDBMS) developed by Oracle that is based on structured query language (SQL).

A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or a place to hold the vast amounts of information in a corporate network. In particular, a relational database is a digital store collecting data and organizing it according to the relational model. In this model, tables consist of rows and columns, and relationships between data elements all follow a strict logical structure. An RDBMS is simply the set of software tools used to actually implement, manage, and query such a database.

Its open-source nature, stability, and rich feature set, paired with ongoing development and support from Oracle, have meant that internet-critical organizations such as Facebook, Flickr, Twitter, Wikipedia, all employ mySQL backends.

Because MySQL enjoys the most widespread use in many industries, business users from new webmasters to experienced managers should strive to understand its main characteristics. Deciding whether to use this technology, and communicating about it effectively, starts with a review of MySQL's basic availability, structure, philosophy, and usability.

MySQL is widely compatible

Though often associated with internet applications or web services, MySQL was designed to be extensively compatible with other technologies and architectures. The RDBMS runs on all major computing platforms, including Unix-based operating systems, such as the myriad Linux distributions or Mac OS, and Windows.

MySQL's client-server architecture means it can support a variety of backends, as well as different programming interfaces. Data can be directly migrated from MySQL to its forks (e.g. MariaDB), as well as most other RDBMSs thanks to architectural and language similarities.

MySQL databases are relational
The primary factor differentiating relational databases from other digital storage lies in how data is organized at a high level. Databases like MySQL contain records in multiple, separate, and highly codified tables, as opposed to a single all-encompassing repository, or collections of semi- or unstructured documents.
This allows RDBMSs to better optimize actions like data retrieval, updating information, or more complex actions like aggregations. A logical model is defined over all of the contents of the database, describing for example the values allowed in individual columns, characteristics of tables and views, or how indices from two tables are related.

Relational models have remained popular for several reasons. They empower users with intuitive, declarative programming languages — essentially telling the database what result is wanted in language akin to, or at least comprehensible as, written english, instead of meticulously coding up each step of the procedure leading to that result. This moves a lot of the work into the RDBMS and SQL engines, better enforcing logical rules and saving valuable resources and manpower.

A.10)MARIADB DATABASE

https://www.guru99.com/mariadb-vs-mysql.html#1
MariaDB is a fork of MySQL. In other words, it is an enhanced, drop-in replacement of MySQL.
A drop-in replacement means that you can substitute the standard MySQL server with the analog version of the MariaDB server and take full advantage of the improvements in the MariaDB without the need to modify your application code.

B) INCEPTION

Elements that need to be modified/created in the host (machine on which the containers will run)

- create the volumes required by the subject : one for containing the Wordpress database (here called db_volume) and one containing the Wordpress website files (called php_nginx_volume) in the host. The path to those volume is indicated in the docker-compose.yml.
- Wordpress need to be downloaded and placed in the php_nginx_volume
- The wp-config-sample.php file need to be renamed to wp-config.php and some lines modified:

```
/** The name of the database for WordPress */
define( 'DB_NAME', 'wordpress' );
/** Database username */
define( 'DB_USER', 'root' );
/** Database password */
define( 'DB_PASSWORD', 'root' );
/** Database hostname */
define( 'DB_HOST', 'mariadb:3306' );
```

- Modify the etc/hosts file : The /etc/hosts file contains the Internet Protocol (IP) host names and addresses for the local host and other hosts in the Internet network. This file is used to resolve a name into an address (that is, to translate a host name into its Internet address).
  Two lines are added : 127.0.0.1 `www.login.42.fr` and 127.0.0.1 login.42.fr

B.1) NGINX

B.1.a) DOCKERFILE

`FROM debian:buster`
Current oldstable version debian

`RUN apt-get update`
apt-get is the command-line tool for handling packages, and may be considered the user's "back-end" to other tools using the APT library. Several "front-end" interfaces exist, such as synaptic and aptitude.

Downloads the package lists from the repositories and "updates" them to get information on the newest versions of packages and their dependencies.

`RUN apt-get upgrade`
will fetch new versions of packages existing on the machine if APT knows about these new versions by way of apt-get update.

```
RUN apt-get install -y nginx
```
-y, --yes, --assume-yes
Automatic yes to prompts. Assume "yes" as answer to all prompts and run non-interactively.

```
RUN apt-get install -y openssl
```
OpenSSL is a general purpose cryptography library that provides an open source implementation of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols. It is an open-source command line tool that is commonly used to generate private keys, create CSRs, install your SSL/TLS certificate, and identify certificate information.

SSL stands for Secure Sockets Layer and, in short, it's the standard technology for keeping an internet connection secure and safeguarding any sensitive data that is being sent between two systems, preventing criminals from reading and modifying any information transferred, including potential personal details. The two systems can be a server and a client (for example, a shopping website and browser) or server to server (for example, an application with personal identifiable information or with payroll information).

It does this by making sure that any data transferred between users and sites, or between two systems remain impossible to read. It uses encryption algorithms to scramble data in transit, preventing hackers from reading it as it is sent over the connection. This information could be anything sensitive or personal which can include credit card numbers and other financial information, names and addresses.

HTTPS (Hyper Text Transfer Protocol Secure) appears in the URL when a website is secured by an SSL certificate. The details of the certificate, including the issuing authority and the corporate name of the website owner, can be viewed by clicking on the lock symbol on the browser bar.

```
COPY tools/nginx.conf /etc/nginx/
```
If you want to copy files and directories inside a Docker Container from your Local machine, you can use the COPY instruction inside your *Dockerfile*. The general form of a COPY instruction is COPY <src-path> <destination-path>

nginx.conf is the default configuration file for nginx and it is placed in the directory /etc/nginx

```
COPY tools/wordpress.conf /etc/nginx/sites-enabled/
```
By default on Debian systems, Nginx server blocks configuration files are stored in /etc/nginx/sites-available directory, which are enabled through symbolic links to the /etc/nginx/sites-enabled/ directory.

The sites-available folder is for storing *all* of your vhost configurations, whether or not they're currently enabled. The sites-enabled folder contains symlinks to files in the sites-available folder. This allows you to selectively disable vhosts by removing the symlink.

https://linuxize.com/post/how-to-set-up-nginx-server-blocks-on-debian-9/
https://serverfault.com/questions/527630/difference-in-sites-available-vs-sites-enabled-vs-conf-d-directories-nginx

```
RUN openssl req -x509 -nodes -days 30 -subj "/C=BE/ST=Belgium/L=Brussels/O=42 Network/OU=s19/CN=yolo" -newkey rsa:4096 -keyout /etc/ssl/nginx-selfsigned.key -out /etc/ssl/nginx-selfsigned.crt;
```
https://www.digicert.com/kb/ssl-support/openssl-quick-reference-guide.htm#:~:text=OpenSSL%20is%20an%20open%2Dsource,certificate%2C%20and%20identify%20certificate%20information.

To give some context as to what we're doing in out openssl options:

- **req** — to specify we want to use -x509
- **-x509** — to specify we want to create a self-signed certificate instead of generating a certificate signing request.
- **-nodes** — makes it so that we skip the option to secure our certificate with a passphrase, so that nginx can read it.
- **-days 365** — specifies how long the certificate would be valid for, which is 365 days.
- **-subj "/C=CA/ST=QC/O=Company, Inc./CN=mydomain.com"** — this allows us to specify subject without filling in prompts. /C for country, /ST for state, /O for organization, and /CN for common name.
- **-newrsa rsa:2048** — specifies that we want to generate both a new certificate and a new key with an RSA key of 2048 bits.
- **-keyout /etc/.../yourfile.key** — specifies the location of the output .key file.

- **-out /etc/.../your file.crt** — specifies the location of the output .crt file.

A Certificate Signing Request (CSR) is required when applying for an SSL certificate.
Instead of generating a private key and then creating a CSR in two separate steps, you can actually perform both tasks at once.

```
CMD nginx -g "daemon off;"
```

For Docker containers (or for debugging), the daemon off; directive tells Nginx to stay in the foreground. For containers this is useful as best practice is for one container = one process. One server (container) has only one service.

Launch the NGINX process in the foreground as opposed as to how it is generally launched : a daemon. We can't use "tail -f" or handy tricks like that and if you don't have an active process in your container, once it's done executing everything it just stops. So we need nginx to run with the specific option of "daemon off;".

In order to keep the docker container running, needs the CMD run in foreground.

https://stackoverflow.com/questions/42319649/docker-custom-nginx-container-failed-to-start

B.1.b) WORDPRESS.CONF

```
server {
```

Regardless of the installation source, server configuration files contain a server block for a website. With Server Blocks, you can specify the site document root (the directory which contains the website files), create a separate security policy for each site, use different SSL certificates for each site, and much more.

```
listen 443 ssl;
listen [::]:443 ssl;
```

listen - tells NGINX the hostname/IP and the TCP port where it should listen for HTTP connections
`listen 443 ssl` : makes nginx listen on all ipv4 address on the server, on port 443 (`0.0.0.0:443`)
while `listen [::]:443 ssl` : makes nginx listen on all ipv6 address on the server, on port 443 (`:::443`)

```
server_name 127.0.0.1;
```

allows multiple domains to be served from a single IP address, here localhost.
Sets names of a virtual server, for example:

```
server {
    server_name example.com www.example.com;
}
```

The first name becomes the primary server name.

```
root /var/www/html/wordpress;
```

Root : var/www/html is just the default root folder of the web server (the directory which contains the website files)

```
ssl on;
ssl_protocols TLSv1.2 TLSv1.3;
```

Since its initial definition in January 1999, Transport Layer Security has gone through a series of updates. The most recent, TLS 1.3, was released in August 2018. The differences between TLS 1.2 and 1.3 are extensive and significant, offering improvements in both performance and security.

```
ssl_certificate /etc/ssl/nginx-selfsigned.crt;
ssl_certificate_key /etc/ssl/nginx-selfsigned.key;
```

Need to tell where the certificates we created earlier by opensssl are.

```
add_header Strict-Transport-Security "max-age=0";
```

deletes the cached HSTS settings in browsers if the TLS certificate is valid (I added this line when I had torubles with an automatic redirection from http to https that was in the end caused by a firefox bug)

```
index index.html index.htm index.php;
```

If you want your visitors to see a home page by typing http://yourdomainname.com/ instead of http://yourdomainname.com/mypage.html, you need to define a “default” page. On some servers, this page must be named index followed by an allowable extension. With most shared hosts allowable index page names are:
index.htm
index.html

index.cgi
index.php

https://www.hostingmanual.net/your-index-home-page/

```
location / {
try_files $uri $uri/ =404;
```
Location directives cover requests for specific files and folders. It also allows NGINX to respond to requests for resources within the server.

If a request ends with a slash, NGINX treats it as a request for a directory and tries to find an index file in the directory. The index directive defines the index file's name (the default value is index.html). To continue with the example, if the request URI is /images/some/path/, NGINX delivers the file /www/data/images/some/path/index.html if it exists. If it does not, NGINX returns HTTP code 404 (Not Found) by default.

In computer network communications, the HTTP 404, 404 not found, 404, 404 error, page not found or file not found error message is a hypertext transfer protocol (HTTP) standard response code, to indicate that the browser was able to communicate with a given server, but the server could not find what was requested.

The last parameter can also be a status code (directly preceded by the equals sign) or the name of a location. In the following example, a 404 error is returned if none of the parameters to the try_files directive resolve to an existing file or directory.

https://docs.nginx.com/nginx/admin-guide/web-server/serving-static-content/
https://nginx.org/en/docs/http/ngx_http_core_module.html#error_page
https://gist.github.com/xameeramir/a5cb675fb6a6a64098365e89a239541d

http://nginx.org/en/docs/http/ngx_http_fastcgi_module.html#fastcgi_split_path_info
```
location ~ \.php$ {
root /var/www/html/wordpress;
try_files $uri $uri/ =404;
fastcgi_split_path_info ^(.+\.php)(/.+)$;
```
Defines a regular expression that captures a value for the $fastcgi_path_info variable. The regular expression should have two captures: the first becomes a value of the $fastcgi_script_name variable, the second becomes a value of the $fastcgi_path_info variable. For example, with these settings

```
location ~ ^(.+\.php)(.*)$ {
    fastcgi_split_path_info       ^(.+\.php)(.*)$;
    fastcgi_param SCRIPT_FILENAME /path/to/php$fastcgi_script_name;
    fastcgi_param PATH_INFO       $fastcgi_path_info;
```
and the "/show.php/article/0001" request, the SCRIPT_FILENAME parameter will be equal to "/path/to/php/show.php", and the PATH_INFO parameter will be equal to "/article/0001".
```
fastcgi_pass wordpress:9000;
```
pass the PHP scripts to FastCGI server listening on wordpress:9000

```
fastcgi_index index.php;
```
Sets a file name that will be appended after a URI that ends with a slash, in the value of the $fastcgi_script_name variable. For example, with these settings

```
fastcgi_index index.php;
fastcgi_param SCRIPT_FILENAME /home/www/scripts/php$fastcgi_script_name;
```
and the "/page.php" request, the SCRIPT_FILENAME parameter will be equal to "/home/www/scripts/php/page.php", and with the "/" request it will be equal to "/home/www/scripts/php/index.php".
```
include fastcgi_params;
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
fastcgi_param SCRIPT_NAME $fastcgi_script_name;
fastcgi_param PATH_INFO $fastcgi_path_info;
```

B.1.c) NGINX.CONF

https://www.linode.com/docs/guides/how-to-configure-nginx/
https://doc.ubuntu-fr.org/nginx
https://codingwithmanny.medium.com/configure-self-signed-ssl-for-nginx-docker-from-a-scratch-7c2bcd5478c6
https://www.nginx.com/resources/wiki/start/topics/recipes/wordpress/
http://nginx.org/en/docs/beginners_guide.html

All NGINX configuration files are located in the /etc/nginx/ directory. The primary configuration file is /etc/nginx/nginx.conf.

Nginx consists of modules which are controlled by directives specified in the configuration file. Directives are divided into simple directives and block directives. A simple directive consists of the name and parameters separated by spaces and ends with a semicolon (;). A block directive has the same structure as a simple directive, but instead of the semicolon it ends with a set of additional instructions surrounded by braces ({ and }). If a block directive can have other directives inside braces, it is called a context (examples: events, http, server, and location).

Directives placed in the configuration file outside of any contexts are considered to be in the main context.
The events and http directives reside in the main context, server in http, and location in server.
The way NGINX configurations are setup is by:

1. directives - they are NGINX configuration options
2. Blocks (also known as contexts) - Groups in which Directives are organized

```
user www-data;
```
Defines *user* and *group* credentials used by worker processes. If *group* is omitted, a group whose name equals that of *user* is used.

```
worker_processes auto;
```
Defines the number of worker processes.The optimal value depends on many factors including (but not limited to) the number of CPU cores, the number of hard disk drives that store data, and load pattern. When one is in doubt, setting it to the number of available CPU cores would be a good start (the value “auto” will try to autodetect it).

```
error_log /var/log/nginx/error.log error;
```
Configures logging. Several logs can be specified on the same configuration level (1.5.2). If on the main configuration level writing a log to a file is not explicitly defined, the default file will be used.
The first parameter defines a *file* that will store the log. The special value stderr selects the standard error file. Logging to syslog can be configured by specifying the “syslog:” prefix. Logging to a cyclic memory buffer can be configured by specifying the “memory:” prefix and buffer *size*, and is generally used for debugging (1.7.11).

The second parameter determines the *level* of logging, and can be one of the following: debug, info, notice, warn, error, crit, alert, or emerg. Log levels above are listed in the order of increasing severity. Setting a certain log level will cause all messages of the specified and more severe log levels to be logged. For example, the default level error will cause error, crit, alert, and emerg messages to be logged. If this parameter is omitted then error is used.

```
pid /var/run/nginx.pid;
```
Defines a *file* that will store the process ID of the main process.

```
Events {
```
The “events” context is contained within the “main” context. It is used to set global options that affect how Nginx handles connections at a general level. There can only be a single events context defined within the Nginx configuration.

```
worker_connections 1024;
```
Nginx worker_connections "sets the maximum number of simultaneous connections that can be opened by a worker process. This number includes all connections (e.g. connections with proxied servers, among others), not only connections with clients.

```
http
```
http blocks contain directives for handling web traffic. These directives are often universal as they are passed on to all website configurations NGINX serves. A list of available directives for http blocks are available on official NGINX http block documentation.
```
include /etc/nginx/mime.types;
default_type application/octet-stream;
access_log /var/log/nginx/access.log combined;
sendfile on;
keepalive_timeout 65;
```

```
include /etc/nginx/sites-enabled/*.conf;
```
Installation from Debian or Ubuntu repositories: the include directive here would now be include /etc/nginx/sites-enabled/*

B.2) WORDPRESS
B.2.a) DOCKFILE
https://developers.redhat.com/blog/2014/12/29/running-php-fpm-in-docker#
https://stackoverflow.com/questions/62525233/php-fpm-wont-start-from-a-dockerfile
https://packages.debian.org/source/buster/php7.3

```
FROM debian:buster

RUN apt-get update
RUN apt-get upgrade

RUN apt-get install -y php7.3-fpm php7.3-mbstring php7.3-mysql php7.3-gd
```

- Mbstring is **an extension of php used to manage non-ASCII strings**. Mbstring is used to convert strings to different encodings.
- Php 7.3-fpm : This package provides the Fast Process Manager interpreter that runs as a daemon and receives Fast/CGI requests.
- Php7.3-gd : GD is an open source code library for the **dynamic creation of images**. GD is used for creating PNG, JPEG and GIF images and is commonly used to generate charts, graphics, thumbnails on the fly.

```
COPY ./tools/php.ini /etc/php/7.3/fpm
```
The configuration file (php.ini) is read when PHP starts up. For the server module versions of PHP, this happens only once when the web server is started. For the CGI and CLI versions, it happens on every invocation. The php.ini file is a special file for PHP. It is where you declare changes to your PHP settings. The server is already configured with standard settings for PHP, which your site will use by default. Unless you need to change one or more settings, there is no need to create or modify a php.ini file.

```
COPY ./tools/php-fmp.conf /etc/php/7.3/fpm
```
Above we are on version 7.2, therefore, php-fpm.conf should be in /etc/php/7.2/fpm/php-fpm.conf

```
COPY ./tools/www.conf /etc/php/7.3/fpm/pool.d
```
php-fpm.conf isn't all of the php-fpm configuration. For serving web requests, php-fpm creates a new pool of processes, which have a separate configuration file www.conf

The php-fpm service creates a default pool, the configuration (www.conf) for which can be found in /etc/php/7.3/fpm/pool.d folder. You can customize the default pool as per your requirements. But it is a standard practice to create separate pools to have better control over resource allocation to each FPM processes.

https://devanswers.co/php-7-php-fpm-configuration-file-location/
https://www.php.net/manual/en/install.fpm.configuration.php
https://devanswers.co/ubuntu-php-php-ini-configuration-file/

Each site is also deployed with a PHP-FPM resource pool, which is owned by the site user. This prevents PHP scripts from reading or modifying files outside of the current site's root directory. Meaning, if a malicious user were to gain access to a site on your server, they would be unable to infect other sites.
```
EXPOSE 9000
```
The EXPOSE instruction exposes the specified port and makes it available only for inter-container communication.

Given the limitation of the `EXPOSE` instruction, a Dockerfile author will often include an `EXPOSE` rule only as a hint to which ports will provide services. It is up to the operator of the container to specify further networking rules.

```
RUN mkdir -p /run/php
CMD /usr/sbin/php-fpm7.3 --nodaemonize
```
force to stay in foreground, and ignore daemonize option from config file

B.2.b) EXTRA FILES

PHP.INI

The php.ini file is a special file for PHP. It is where you declare changes to your PHP settings. The server is already configured with standard settings for PHP, which your site will use by default. Unless you need to change one or more settings, there is no need to create or modify a php.ini file.

At the time of PHP installation, **php.ini** is a special file provided as a default configuration file. It's very essential configuration file which controls, what a user can or cannot do with the website. Each time PHP is initialized, the **php.ini** file is read by the system. Sometimes you need to change the behavior of PHP at runtime, then this configuration file is to use.

All the settings related to register global variables, upload maximum size, display log errors, resource limits, the maximum time to execute a PHP script and others are written in a file as a set of directives which helps in declaring changes.

Important settings or common parameters of the php.ini file:

1. **enable_safe_mode = on** Its default setting to ON whenever PHP is compiled. Safe mode is most relevant to CGI use.
2. **register_globals = on** its default setting to ON which tells that the contents of EGPCS (Environment, GET, POST, Cookie, Server) variables are registered as global variables. But due to a security risk, the user has to ensure if it set to OFF for all scripts.
3. **upload_max_filesize** This setting is for the maximum allowed size for uploaded files in the scripts.
4. **upload_tmp_dir = [DIR]** Don't uncomment this setting.
5. **post_max_size** This setting is for the maximum allowed size of POST data that PHP will accept.
6. **display_errors = off** This setting will not allow showing errors while running PHP project in the specified host.
7. **error_reporting = E_ALL & ~E_NOTICE**: This setting has default values as E_ALL and ~E_NOTICE which shows all errors except notices.
8. **error_prepend_string = [""]** This setting allow you to make different color of messages.
9. **max_execution_time = 30** Maximum execution time is set to seconds for any script to limit the time in production servers.
10. **short_open_tags = Off** To use XML functions, we have to set this option as *off.*
11. **session.save-handler = files** You don't need to change anything in this setting.
12. **variables_order = EGPCS** This setting is done to set the order of variables as Environment, GET, POST, COOKIE, SERVER. The developer can change the order as per the need also.
13. **warn_plus_overloading = Off** This setting issues a warning if + used with strings in a form of value.
14. **gpc_order = GPC** This setting has been GPC Deprecated.
15. **magic_quotes_gpc = o**n This setting is done in case of many forms are used which submits to themselves or others and display form values.
16. **magic_quotes_runtime = Off** If magic_quotes_sybase is set to On, this must be Off, this setting escape quotes.
17. **magic_quotes_sybase = Off** If this setting is set to off it should be off, this setting escape quotes.
18. **auto-prepend-file = [filepath]** This setting is done when we need to automatically *include()* it at the beginning of every PHP file.
19. **auto-append-file = [filepath]** This setting is done when we need to automatically *include()* it at the end of every PHP file.
20. **include_path = [DIR]** This setting is done when we need to require files from the specified directories. Multiple directories are set using colons.
21. **ignore_user_abort = [On/Off]** This settings control what will happen when the user click any stop button. The default value is on this setting doesn't work on CGI mode it works on only module mode.
22. **doc_root = [DIR]** This setting is done if we want to apply PHP to a portion of our website.
23. **file_uploads = [on/off]** This flag is set to *ON* if file uploads are included in the PHP code.
24. **mysql.default_host = hostname** This setting is done to connect to MySQL default server if no other server host is mentioned.
25. **mysql.default_user = username** This setting is done to connect MySQL default username, if no other name is mentioned.
26. **mysql.default_password = password** This setting is done to connect MySQL default password if no other password is mentioned.

PHP-FPM.CONF
https://gist.github.com/lidaobing/673798

WWW.CONF
https://myjeeva.com/php-fpm-configuration-101.html

Pool Directives are a PHP-FPM convention where multiple "pools" of child processes can be started and have different configurations. The default name for the pool directives file is www.conf.

Multiple pools of child processes may be started with different listening ports and different management options. The name of the pool will be used in logs and stats. There is no limitation on the number of pools which FPM can handle. So system limit is the FPM limit.

B.3) MARIADB
B.3.a) DOCKFILE

https://mariadb.com/kb/en/creating-a-custom-docker-image/
https://getmoven.com/index.php/knowledgebase/18/Sample-my.cnf---MySQL-Configuration-File.html
https://mariadb.com/kb/en/configuring-mariadb-with-option-files/
https://www.ibm.com/docs/en/spectrum-lsf-rtm/10.2.0?topic=ssl-configuring-default-root-password-mysqlmariadb

```
FROM debian:buster
```

```
RUN apt-get update
RUN apt-get upgrade
```

```
RUN apt-get install -y mariadb-server
```

```
RUN apt-get -y install vim
```

```
COPY tools/setup.sh .
```
We want to copy to the container our own setup script that we will use to config and launch mysql to the current directory.

```
EXPOSE 3306
```
The EXPOSE instruction exposes the specified port and makes it available only for inter-container communication.

Given the limitation of the `EXPOSE` instruction, a Dockerfile author will often include an `EXPOSE` rule only as a hint to which ports will provide services. It is up to the operator of the container to specify further networking rules.

```
COPY ./tools/my.cnf /etc/mysql
```
The default configuration file is called my.cnf (or my.ini for Microsoft Windows) and can be located in a number of directories. On Linux and other Unix related platforms, the locations are using /etc/my.cnf, /usr/my.cnf or in the default installation directory. This file contains configuration settings that will be loaded when the server is first started including settings for the clients, server, mysqld_safe wrapper and various other mysql client programs.

```
COPY ./tools/setup.sh .
CMD /setup.sh
```

B.3.b) SETUP.SH

```
if [ ! -d "/var/lib/mysql/wordpress" ]; then
```
These commands only have to be executed the first time the container is launched. If the /var/lib/mysql/wordpress file exists, it means the set up is done.

```
mysql_install_db
```
mysql_install_db initializes the MySQL data directory and creates the system tables that it contains, if they do not exist.

```
service mysql start
```
starts mysql

```
echo "DROP DATABASE wordpress;" | mysql -u root -password=root;
```
All of the lines starting with echo are input in my sql. The DROP DATABASE statement is used to drop/delete an existing SQL database. First as a safety we remove wordpress if it already exists

```
echo "CREATE USER 'root'@'%' IDENTIFIED BY 'root';" | mysql -u root -password=root;
```
We then create a new user, root, with a password root. The % is a default value for the user's host. The identified by refers to the password

```
echo "CREATE DATABASE wordpress;" | mysql -u root -skip-password;
```
CREATE DATABASE creates a database with the given name. To use this statement, you need the CREATE privilege for the database. CREATE SCHEMA is a synonym for CREATE DATABASE.

```
echo "ALTER DATABASE wordpress CHARACTER SET = 'utf8mb4' COLLATE =
'utf8mb4_general_ci';" | mysql -u root --password=root;
```
A collation is a set of rules that defines how to compare and sort character strings. Each collation in MySQL belongs to a single character set. Every character set has at least one collation, and most have two or more collations.

```
echo "GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'root' WITH GRANT OPTION;"
| mysql -u root -password=root;
```
The asterisks in this command refer to the database and table (respectively) that they can access—this specific command allows to the user to read, edit, execute and perform all tasks across all the databases and tables.Please note that in this example we are granting full root access to everything in our database.

```
echo "FLUSH PRIVILEGES;" | mysql -u root -password=root;
```
To put the changes (full access for root) into effect.

```
service mysql stop
fi
#sleep 5
echo "ALTER USER 'root'@'localhost'IDENTIFIED BY '$DB_PASSWORD';" > /var/lib/mysql/pass-
reset
```
These two lines where added because a connection to mysql without a password was possible. We put in `/var/lib/mysql/pass-reset` the new password.

```
mysqld_safe -init-file=/var/lib/mysql/pass-reset
```
This launches MySQL, and apply the text-file password change. mysqld_safe is the recommended way to start a mysqld server on Unix. mysqld_safe adds some safety features such as restarting the server when an error occurs and logging runtime information to an error log.

B.3.c) MY.CNF
https://dev.mysql.com/doc/refman/8.0/en/option-files.html

B.4) DOCKER-COMPOSE
https://docs.docker.com/engine/reference/commandline/volume_create/#/driver-specific-options
https://stackoverflow.com/questions/35841241/docker-compose-named-mounted-volume
https://medium.com/swlh/wordpress-deployment-with-nginx-php-fpm-and-mariadb-using-docker-compose-55f59e5c1a
https://gabrieltanner.org/blog/docker-compose
https://docs.docker.com/compose/compose-file/compose-file-v2/#init
https://github.com/compose-spec/compose-spec/blob/master/spec.md

```
version: "3.7"
services:
```
Services : NGINX, MariaDB, PHP-FPM, all the containers which are included in the Compose file.

```
Mariadb:
```
Mariadb : handles the database (we want the data to be persistent so we use volumes to have a shared directory between the host machine and the MariaDB container)

```
init: true
```
init : Run an init inside the container that forwards signals and reaps processes.
```
build:
context: ./requirements/mariadb
```
build : path to dockerfile and directory in which the container will be built

```
volumes:
- "db_volume:/var/lib/mysql/"
```
Specify where the volume is going to be located inside the container (Docker volumes are file systems mounted on Docker containers to preserve data generated by the running container. The volumes are stored on the host, independent of the container life cycle. This allows users to back up data and share file systems between containers easily.)

```
restart: always
```
restart: which will restart the container automatically if it crashes.

```
networks:
- my-network
```
Docker networking is primarily used to establish communication between Docker containers and the outside world via the host machine where the Docker daemon is running. Docker networking allows you to attach a container to as many networks as you like. You can also attach an already running container. Docker Compose sets up a single network for your application(s) by default, adding each container for a service to the default network. Containers on a single network can reach and discover every other container on the network.

```
env_file:
- .env
```
env_file: use the .env_local to supply environment variables to the container, Environment variables are used to bring configuration data into your applications. This is often the case if you have some configurations that are dependent on the host operating system or some other variable things that can change.
Sometimes a few environment variables aren't enough and managing them in the Compose file can get pretty messy. That is what .env files are for. They contain all the environment variables for your container and can be added using one line in your Compose file.
Add environment variables from a file. Can be a single value or a list.

```
wordpress:
```
PHP : fill the php page from our database.
```
init: true
build:
context: ./requirements/wordpress
volumes:
- "php_nginx_volume:/var/www/html/"
restart: always
networks:
- my-network
env_file:
.env
```

```
depends_on:
- mariadb
```
depends_on: Dependencies in Docker are used to make sure that a specific service is available before the dependent container starts. This is often used if you have a service that can't be used without another one e.g. a CMS (Content Management System) without its database.

```
nginx:
```
Nginx : accepts HTTPS requests from browser and fetch the php page from the Wordpress/php-fpm container.
```
init: true
build:
context: ./requirements/nginx
volumes:
- "php_nginx_volume:/var/www/html/"
restart: always
networks:
- my-network
```

```
ports:
- "443:443"
```
ports: exposing the port to the host system eg “8000:80/udp”, Expose ports. Either specify both ports (HOST:CONTAINER), or just the container port (an ephemeral host port is chosen). Activates the container to listen for specified port(s) from the world outside of the docker(can be same host machine or a different machine) AND also accessible world inside docker.

```
depends_on:
- wordpress
- mariadb
env_file:
- .env
```

```
networks:
```
Network: tell how the containers can communicate. Default settings and bridges (use one for the three containers so potentially, they can all communicate).
```
my-network:
```

```
volumes:
```
Volume : shared directory between host and the containers. Need one for DB and one shared between the host and the NGINX/Wordpress container (when NGINX need to have access to a php file).

```
db_volume:
driver: local
driver_opts:
type: none
o: bind
device: /home/mdeclerf/data/db_volume
php_nginx_volume:
driver: local
driver_opts:
type: none
o: bind
device: /home/mdeclerf/data/php_nginx_volume
```

B.5) Makefile

```
all : up
up :
docker-compose -f srcs/docker-compose.yml up –build
```
Builds, (re)creates, starts, and attaches to containers for a service.
Unless they are already running, this command also starts any linked services.
The docker compose up command aggregates the output of each container. When the command exits, all containers are stopped.
--build : build images before starting the containers

```
stop :
docker-compose -f srcs/docker-compose.yml down -v
```
Stops containers and removes containers, networks, volumes, and images created by up.
By default, the only things removed are:

- Containers for services defined in the Compose file
- Networks defined in the networks section of the Compose file
- The default network, if one is used

Networks and volumes defined as external are never removed.
Remove named volumes declared in the volumes section of the Compose file and anonymous volumes attached to containers.

```
clean :
docker-compose -f srcs/docker-compose.yml rm
```
Remove one or more containers

C) COMMANDS

C.1)DOCKER

docker volume rm removes the containers

docker-compose up –build (--build: Build images before starting containers.) build the images and starts the containers

docker-compose down Stops containers and removes containers, networks, volumes, and images created by up.

docker ps list containers

docker exec -it nom_du_container bash run a command inside the container

docker exec -it srcs_mariadb_1 bash

docker network ls list the containers networks

docker volume ls list the volumes

docker volume inspect details about the volumes

docker-compose ps

docker stop $(docker ps -qa); docker rm $(docker ps -qa);docker rmi -f $(docker images -qa); docker volume rm $(docker volume ls -q);docker network rm $(docker network ls -q) 2>/dev/null

C.2) MYSQL

USE database_name; use wordpress database

SHOW DATABASES; show which ones there are

Describe wp_users; describes what is in a user

SELECT * FROM wp_users; show current users (mdeclerf admin)

RENAME USER *old_user* TO *new_user*

mysql -uroot -p connection with a password

SHOW TABLES

ssh-keygen -t rsa / cat id_rsa.pub

wp-login.php

w-p-admin.php