ft_server

Class	3rd Circle
Created	@Jan 8, 2021 2:06 PM

Project

Summary:

> This is a System Administration subject. You will discover Docker and you will set up your `first web server`

This topic is intended to introduce you to system administration. It will make you aware of the importance of using scripts to automate your tasks. For that, you will discover the "docker" technology and use it to install a complete web server. This server will run multiples services: Wordpress, phpMyAdmin, and a SQL database.

Mandatory Part:

•You must set up a web server with Nginx, in only one docker container. The container OS must be debian buster.
• Your web server must be able to run several services at the same time. The services will be a WordPress website, phpMyAdmin and MySQL. You will need to make sure your SQL database works with the WordPress and phpMyAdmin.
• Your server should be able to use the SSL protocol.
• You will have to make sure that, depending on the url, your server redirects to the correct website.
• You will also need to make sure your server is running with an autoindex that must be able to be disabled.

My Notes

What is 'system administration'?

> System administration refers to the management of one or more hardware and software systems.

A system administrator, or sysadmin, is a person who is responsible for the upkeep, configuration, and reliable operation of computer systems; `especially multi-user computers, such as servers`

System administration is a job done by IT experts for an organization. The job is to ensure that computer systems and all related services are working well. The duties in system administration are wide ranging and often vary depending on the type of computer systems being maintained, although most of them share some common tasks that may be executed in different ways.

Common tasks include installation of new hardware or software, creating and managing user accounts, maintaining computer systems such as servers and databases, and planning and properly responding to system outages and various other problems. Other responsibilities may include light programing or scripting to make the system workflows easier as well as training computer users and assistants.

Tools

First, a quick overview of the tools we are going to use:

- **Docker** is a software platform for building applications based on containers.
- **NGINX** is a free web server software particularly suitable for websites with high traffic. Unlike Apache, it is light and fast but has fewer modules than Apache (this tends to weaken over the years).

- **MySQL** is a Relational Database Management System that helps you to keep the data that exists in a database organised. **+ MariaDB:** MariaDB is a fork of the MySQL database management system.
- **phpMyAdmin** is a web application for managing MySQL databases.
- **WordPress** is a free and open-source content management system based on the PHP and MySQL programming languages.
- **Debian** is a popular and freely-available computer operating system that uses the Linux kernel and other program components obtained from the GNU project.

What to do explained in simple steps?

1) Create OS container based on debian with Docker

2) Load nginx to debian

3) Upload Mysql to be used by nginx web server

4) Use MySQL with wordpress and phpmyadmin

5) Automate the above process using dockerfile

Detailed Information > Tools

Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications.

Before Docker

The explanation of the Docker technology cannot directly start with the containers.

Prerequisites :

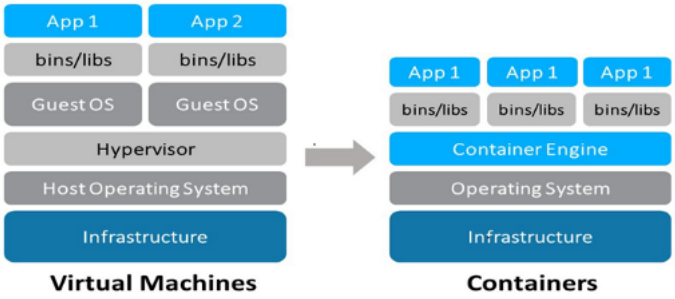
Let's say you need to build an application. In order to make this application accessible to the public, you need a place to host it. Today, we have three options for doing it:

a) Public web-hosting services :These are websites like GoDaddy, 1&1, etc, which can help to host your application

b) Leveraging the Cloud Computing model : Companies like Oracle (with Oracle Cloud Infrastructure) and Amazon (with AWS) can host your application through their own data centers(collection of servers).

c) Docker: this technology provides you the third option with a concept known as "Containerization". Let's try to comprehend this term by comparing it with Virtualization. Both of these help you in deploying applications inside environments that are isolated from the underlying hardware. The chief difference is the level of isolation.

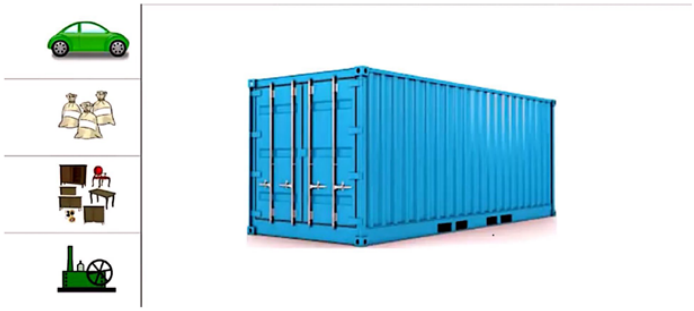
- Virtualization enables the creation of Virtual Machines(VM) from the underlying hardware through Hypervisor. You can think of Hypervisor as a Virtual Machine Monitor that creates and manages virtual machines. However, each VM contains the Operating System(OS) layer(which you as a user have to manage). This makes the VM heavyweight(Imagine a VM of size 1 GB to deploy an application of size, just about 300 MB)
- Containerization mitigates this issue with containers. A Container is a package that constitutes an application with all of the parts it needs such as libraries and dependencies. Instead of hosting each OS per application, another layer called a "Container Engine" is added between the OS and the applications. This allows the OS to be shared amongst all the containers. As a consequence of containers not needing to embed a full OS, they are very lightweight, commonly around 5–100 MB.



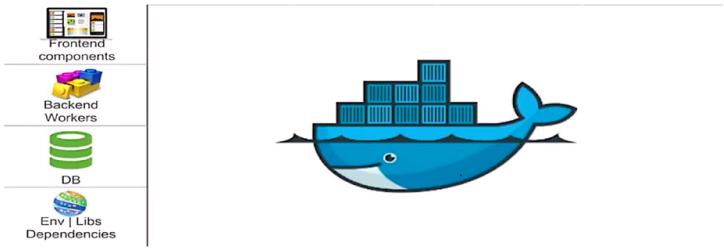
What is Docker?

At this point, we have come to a point where we can define Docker. Technical concepts can often be understood with an analogy, a classical problem in the shipping industry : We have items of different shapes, characteristics. Some are fragile, some are hard and there is no standardized way to package these items. What do we do?

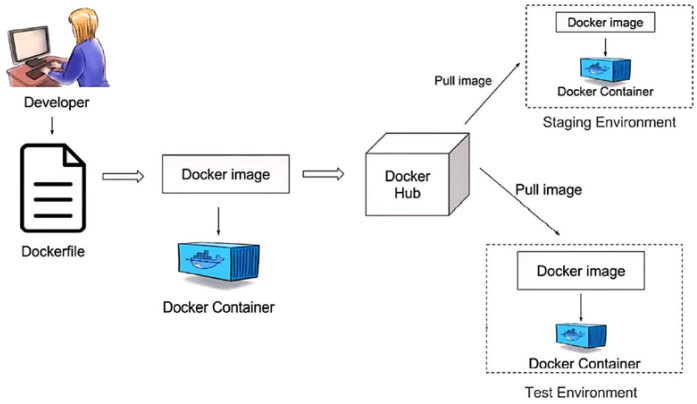
This is where a CONTAINER can help!



No matter what the item, we can package it in this container and transport it from the source to the destination without any changes. Now map the same scenario to the Software Industry. Each application has differences in the frontend, backend, and database components. We need a standardized way to package the application with its dependencies and deploy it on any environment. The docker with its containers does exactly that. **Docker** is a tool designed to make it easier to create, deploy, and run applications by using containers.



The Docker workflow :



1. Every single Docker container begins as a pure vanilla Linux machine that knows nothing. Then, we tell the container everything it needs to know — all the dependencies it needs to download and install in order to run the application. This process is done with `a Dockerfile`. A Dockerfile is a text file written in an easy-to-understand syntax that includes the instructions to build `a Docker image`. A Dockerfile specifies the operating system that will underlie the container, along with the languages, environmental variables, file locations, network ports, and other components it needs — and, of course, what the container will actually be doing once we run it.
2. We then construct a Docker image out of this. You can run a container(s) from the same image whenever you need it.
3. The Docker image is uploaded to `the Docker Hub` (or any other repository to store docker images at once).
4. The images can be pulled from the Docker hub and the corresponding containers built out of it can be deployed on any environments(Testers' environment, production environment).

The difference between a Docker image and a Docker container:

Docker **images** are executable packages that include everything needed to run an application — the code, runtime libraries, environment variables, and configuration files.

Docker **containers** are a runtime instance of an image — what the image becomes in memory when executed (that is, an image with the state, or a user process).

Why Docker?

1. Imagine you have a software company and everyone uses their own Linux distribution or simply a different OS like Mac or Windows, this means that the code might work on their computer, but not on a production server. Docker was made to unify the developer experience, so that the code will work no matter on which operating system you are.
2. Docker images start up really fast, and big corporations usually have thousands of servers, so quickly updating your software becomes quite a hassle. Therefore they use Docker images to run their code, as it allows them turn them off, download a new one, and run it again. This is also known as rolling updates. Because of this sole reason, they can more easily add servers and deploy their software to them, as it just needs to download a tiny image and run it, after which it will work accordingly.
3. Docker eliminates the inevitable confusion that comes when a developer's been working on their local machine on a project for days (or weeks), and as soon as it's deployed to a new lifecycle, the app won't run. Most likely because there's a host of installed dependencies that are necessary to run the application. We essentially "ship" the developer's application (in a container) and deploy it to any environment without any hassle in the deployment.
4. Docker makes it easier to share and run a complex dev environment
5. Simplicity and Faster configurations is an implicit benefit of using Docker

Docker Documentation Website

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight

because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines!

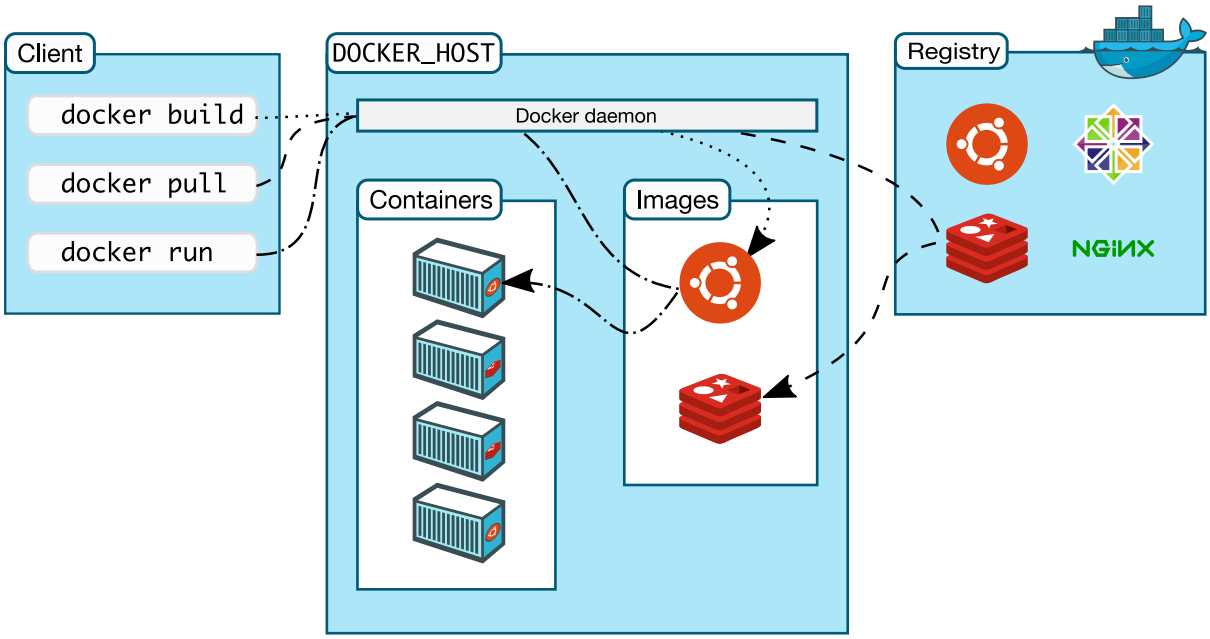
Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

`> This part in ft_server:` "You must set up a web server with Nginx, in only one docker container." = This contradicts the idea of Docker with working with running more containers on a given hardware combination than if you were using virtual machines. So do all the services have to be in one container? = The reason I believe it's like this, is to provide very basic docker knowledge before you start ft_services. Its so that you figure out and see why it should not be done like that. You learn the most from worst practices and mistakes, thats only what this enforces.

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.



The Docker daemon

The Docker daemon (`dockerd`) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client

The Docker client (`docker`) is the primary way that many Docker users interact with Docker. When you use commands such as `docker run`, the client sends these commands to `dockerd`, which carries them out. The `docker` command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker registries

A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the `docker pull` or `docker run` commands, the required images are pulled from your configured registry. When you use the `docker push` command, your image is pushed to your configured registry.

Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

IMAGES

An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization. For example, you may build an image which is based on the `ubuntu` image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

CONTAINERS

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

SERVICES

Services allow you to scale containers across multiple Docker daemons, which all work together as a *swarm* with multiple *managers* and *workers*. Each member of a swarm is a Docker daemon, and all the daemons communicate using the Docker API. A service allows you to define the desired state, such as the number of replicas of the service that must be available at any given time. By default, the service is load-balanced across all worker nodes. To the consumer, the Docker service appears to be a single application. Docker Engine supports swarm mode in Docker 1.12 and higher.

Containerization and Container in General

In traditional software development, code developed in one computing environment often runs with bugs and errors when deployed in another environment. Software developers solve this problem by running software in 'containers' in the cloud.

Containerization is defined as a form of operating system virtualization, through which applications are run in isolated user spaces called containers, all using the same shared operating system (OS). A container is essentially a fully packaged and portable computing environment:

- Everything an application needs to run – its binaries, libraries, configuration files and dependencies – is encapsulated and isolated in its container.
- The container itself is abstracted away from the host OS, with only limited access to underlying resources – much like a lightweight virtual machine (VM).
- As a result, the containerized application can be run on various types of infrastructure—on bare metal, within VMs, and in the cloud—without needing to refactor it for each environment.

How Does Containerization Actually Work?

Each container is an executable package of software, running on top of a host OS. A host(s) may support many containers (tens, hundreds or even thousands) concurrently, such as in the case of a complex microservices architecture that uses numerous containerized ADCs. This setup works because all containers run minimal, resource-isolated processes that others cannot access.

Think of a containerized application as the top layer of a multi-tier cake:

1. At the bottom, there is the hardware of the infrastructure in question, including its CPU(s), disk storage and network interfaces.
2. Above that, there is the host OS and its kernel – the latter serves as a bridge between the software of the OS and the hardware of the underlying system.
3. The container engine and its minimal guest OS, which are particular to the containerization technology being used, sit atop the host OS.
4. At the very top are the binaries and libraries (bins/libs) for each application and the apps themselves, running in their isolated user spaces (containers).

`Docker commands?`

LEMP Stack

The LEMP software stack is a group of software that can be used to **serve dynamic web pages and web applications**. The name "LEMP" is an acronym that describes a **L**inux operating system, with an (**E**)Nginx web server. The backend data is stored in a **M**ariaDB database and the dynamic processing is handled by **P**HP. Although this software stack typically includes **MySQL** as the database management system, some Linux distributions — including Debian — use MariaDB as a drop-in replacement for MySQL.

> My thoughts in the beginning of this project: It's completely different from previous projects where we re-wrote library functions. Now, we need to learn about new technologies like Docker which is very exciting but unfamiliar at the same time.

Prerequisites for the Project

Now that you know the usecase for Docker, you must start your project. There are a few requirements that are not listed in the subject that you should know of:

- Your Docker container must configure / download / install all the services when you run the `docker build` command, you ought not configure ANY services or change ANY files after using `docker run` as this mitigates the entire point of Docker.
- This means wordpress MUST be installed during `docker build` and not by means of wp_setup.php after running your container.
- Wordpress should work in its entirety, make sure you check for mail posting / uploading files / uploading themes / installing themes / creating blog posts / installing modules. You should also verify whether you have the correct php extensions installed.
- PHPMyAdmin should work in its entirty, so make sure you can create a database, download a database, create users, remove users, update tables, etc.
- Make sure you have the CORRECT file permissions across your system. This is INSANELY important as it is the main thing that can guarantee that your docker container will not become the victim of e.g. remote code execution.
- If chrome tells you 'Your connection is not private', just type thisisunsafe and hit enter to see your website.

HOW TO BUILD FT_SERVER > STEPS EXPLAINED

Step 1 — Create a Dockerfile and install Debian Operating System

Docker builds images automatically by reading the instructions from a `Dockerfile` -- a text file that contains all commands, in order, needed to build a given image. A `Dockerfile` adheres to a specific format and set of instructions.

A Docker image consists of read-only layers each of which represents a Dockerfile instruction. The layers are stacked and each one is a delta of the changes from the previous layer.

`FROM` creates a layer from the debian:buster Docker image. By using From you can install any OS in your virtual environment.

```
FROM debian:buster
```


All of the software you will be using for this procedure will come directly from **Debian**'s (an OS we are working with) default package repositories. This means you can use the `apt` package management suite to complete the installation.

Step 2 — Update Software Packages

Before install other stacks, update software packages in debian. **wget** command will be used to get .tar file of phpmyadmin and wordpress, so install it in advance. also unzip command to deal with zip files.

```
RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get -y install wget
RUN apt-get -y install unzip
```

`RUN` > This docker instruction is used during the build process. It runs any commands on top of the new Docker Image and creates a new layer with the results of command execution. As a result, the new Image will be used for Dockerfile's next step.

`APT` > provides a high-level commandline interface for the package management system. It is intended as an end user interface and enables some options better suited for interactive
usage by default compared to more specialized APT tools like apt-get(8) and apt-cache(8).

Step 3 — Installing the Nginx Web Server

In order to serve web pages to your site visitors, we are going to employ Nginx, a popular web server which is well known for its overall performance and stability.

`NGINX:` Picture this - you've created a web application and are now searching for the right web server to host it from. NGINX is a web server that also acts as an email proxy, reverse proxy, and load balancer. The software's structure is asynchronous and event-driven; which enables the processing of many requests at the same time. NGINX is highly scalable as well, meaning that its service grows along with its clients' traffic.

```
RUN apt-get -y install nginx
```

Step 4 — Create a shell script to initiate your container

In order to start your installed packages and create database for Wordpress and configure MySQL easier, create a init.sh file like below. Don't for get to add a command to your Dockerfile (at the very end of the file) to run your shell script like this one.

```
FROM debian:buster

# Update software packages in debian
RUN apt-get update
RUN apt-get upgrade -y

# Install necessary packages & unzip&wget command
RUN apt-get -y install wget
RUN apt-get -y install unzip
RUN apt-get -y install nginx

ENTRYPOINT ["sh","init.sh"]
```

Ultimately, both ENTRYPOINT and CMD give you a way to identify which executable should be run when a container is started from your image. In fact, if you want your image to be runnable (without additional `docker run` command

line arguments) you **must** specify an ENTRYPOINT or CMD. Trying to run an image which doesn't have an ENTRYPOINT or CMD declared will result in an error.

init.sh > To start, you should create a website folder which would initially hold all of other paths. It's also nice to create a simple index.html (you can find it inside srcs foler) file with a title and a few buttons to link you to wordpress, php, phpmyadmin (which will be installed in further steps). Remember to move your html file to its own path in your webpage.

```
# Generate website folder & move html file to the folder
mkdir /var/www/localhost
mv ./tmp/index.html /var/www/localhost/index.html

# Boot up nginx
service nginx start

bash
```

Step 5 — Configure Nginx

The way nginx and its modules work is determined in the **configuration file**. By default, the configuration file is named nginx.conf and placed in the directory /usr/local/nginx/conf, /etc/nginx, or /usr/local/etc/nginx. To access config file of nginx, run the docker by using below command in the directory which your dockerfile locates. Build the image first, and then run it. Build the image with name 'nginx' or whatever name you prefer. And run the container with port 80:80 and 443:443. rm option will delete the container automatically after quitting this container.

```
docker build -t nginx .
docker run -p 80:80 -p 443:443 -it --rm nginx
```

WHAT RUN DID

- Looked for image called nginx in image cache
- If not found in cache, it looks to the default image repo on Dockerhub
- Pulled it down (latest version), stored in the image cache
- Started it in a new container
- We specified to take port 80- on the host and forward to port 80 on the container
- We could do "$ docker container run --publish 8000:80 --detach nginx" to use port 8000
- We can specify versions like "nginx:1.09"

Configure Nginx

It will run in bash form. Since nginx does routing from default file in /etc/nginx/sites-available, you'll need to go to the directory and copy&change it.

This is how you access default nginx config file inside your container

```
cd /etc/nginx/sites-available
cat default
```

Create your own configuration file

```
server {
  listen 80;
  listen [::]:80;
  server_name localhost;
  return 301 https://$server_name$request_uri;
}

server {
  listen 443 ssl ;
  listen [::]:443 ssl ;

  server_name localhost;

  ssl_certificate /etc/nginx/ssl/localhost.pem;
```

```
  ssl_certificate_key /etc/nginx/ssl/localhost.key;

  root /var/www/localhost;

  index index.html index.htm index.nginx-debian.html index.php;

  location / {
    autoindex on;
    try_files $uri $uri/ =404;
  }

  location ~ \.php$ {
    include snippets/fastcgi-php.conf;
    fastcgi_pass unix:/var/run/php/php7.3-fpm.sock;
  }
}
```

Explanation

1. First Server block

- **[::]:** =>According to nginx docs, these square brackets [::]: are for IPv6 .
- **server_name ⇒** routing following domain.
- **return 301** => HTTP status code 301 makes redirection to HTTPS automatically.

2. Second Server block

- Setting **SSL** Key and **autoindex** is optional.
- **root ⇒** if server_name domain exists, you can set the root folder.
- **index ⇒** set following files as index files.
- **first location block⇒** Since Nginx doesn't support static file hosting as default, looking for file in the folder followed by uri within root folder. If cannot find this file, it shows 404(page not found) error.
- **second location block⇒** This enables php program connected with Nginx. php-fpm stands for PHP FastCGI Process Manager.

Now that you have you config file, you want to copy the file from your local directory to container layer built when docker build command is called. To use these files during initialisation, copy them to the ./tmp directory. Also, remember to copy your init.sh file to be able to bash it.

```
FROM debian:buster

# Update software packages in debian
RUN apt-get update
RUN apt-get upgrade -y

# Install necessary packages & unzip&wget command
RUN apt-get -y install wget
RUN apt-get -y install unzip
RUN apt-get -y install nginx

# Copy necessary files from your host to your image
COPY ./srcs/init.sh ./
COPY ./srcs/nginx-conf ./tmp/nginx-conf
COPY ./srcs/index.html ./tmp/index.html

ENTRYPOINT ["sh","init.sh"]
```

Inside your shell script (init.sh), move the config file inside nginx directory, make a symbolic link from your `sites-available/localhost` to `sites-enabled`. Otherwise nginx doesn't know that you want to use the non-default configuration. Lastly, remove the default config file.

```
# Generate website folder & move html file to the folder
mkdir /var/www/localhost
mv ./tmp/index.html /var/www/localhost/index.html

# Configure nginx
mv ./tmp/nginx-conf /etc/nginx/sites-available/localhost
ln -s /etc/nginx/sites-available/localhost /etc/nginx/sites-enabled/localhost
rm -rf /etc/nginx/sites-enabled/default

# Boot up nginx
```

```
service nginx start

bash
```

Step 6 — Installing MariaDB

Now that you have a web server up and running, you need to install the database system to be able to store and manage data for your site.

In Debian 10, the metapackage mysql-server, which was traditionally used to install the MySQL server, was replaced by default-mysql-server. This metapackage references MariaDB, a community fork of the original MySQL server by Oracle, and it's currently the default MySQL-compatible database server available on debian-based package manager repositories. For longer term compatibility, however, it's recommended that instead of using the metapackage you install MariaDB using the program's actual package, mariadb-server.

Put commands below to your dockerfile. Then build the image and run it. Shell script will be excecuted.

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

In your shell script, add mysql configuration.

```
# Generate website folder & move html file to the folder
mkdir /var/www/localhost
mv ./tmp/index.html /var/www/localhost/index.html

# Configure nginx
mv ./tmp/nginx-conf /etc/nginx/sites-available/localhost
ln -s /etc/nginx/sites-available/localhost /etc/nginx/sites-enabled/localhost
rm -rf /etc/nginx/sites-enabled/default

# Boot up mysql & Configure a wordpress database
service mysql start
echo "CREATE DATABASE wordpress;"| mysql -u root --skip-password
echo "GRANT ALL PRIVILEGES ON wordpress.* TO 'root'@'localhost' WITH GRANT OPTION;"| mysql -u root --skip-password
echo "FLUSH PRIVILEGES;"| mysql -u root --skip-password
echo "update mysql.user set plugin='' where user='root';"| mysql -u root --skip-password

# Boot up nginx
service nginx start

bash
```

MySQL config explanation

1. CREATE DATABASE wordpress;

> it's just simply creating schema named with 'wordpress'.

2. GRANT ALL PRIVILEGES ON '***Schema name***'.'***table name***' TO '***account name***'@'***host***' identified by '***account password***' WITH GRANT OPTION;

> Making account which can access to certain schema. In our case, we are making root account which can access to all tables in wordpress schema. And we skipped password by using **mysql -u root — skip-password** option from MySQL Docs.

3. FLUSH PRIVILEGES;

> If using `INSERT`, `UPDATE`, or `DELETE` statements, you should tell the server to reload the grant tables, perform a flush-privileges operation. Look for this MySQL documentation and stackoverflow link.

4. update mysql.user set plugin='' where user='root';

> Start from MySQL Server 5.7, if we do not provide a password to root user during the installation, it will use auth_socket plugin for authentication. With this configuration, MySQL won't care about your input password, it will check the user is connecting using a UNIX socket and then compares the username. But it matters when login to mysql root from other normal linux user account. That 's why we add this line.

Step 7 — Installing PhpMyAdmin for Processing

You have Nginx installed to serve your content and MySQL installed to store and manage your data. Now you can install phpMyAdmin to process code and generate dynamic content for the web server. phpMyAdmin helps the system

administrator to perform databases activities such as creating, deleting, querying, database, tables, columns, etc.

Before installing PhpMyAdmin you should install php first.

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

Then in your shell script, install phpmyadmin by using 'wget' and unzip the phpmyadmin file. Also, move the installed file and config file to its relevant web path. Don't forget to boot up php at the very end!

```
# Generate website folder & create index.php & move html file
mkdir /var/www/localhost && touch /var/www/localhost/index.php
echo "<?php phpinfo(); ?>" >> /var/www/localhost/index.php
mv ./tmp/index.html /var/www/localhost/index.html

# Config Access
chown -R www-data /var/www/*
chmod -R 755 /var/www/*

# SSL
mkdir /etc/nginx/ssl
openssl req -newkey rsa:4096 -x509 -sha256 -days 365 -nodes -out /etc/nginx/ssl/localhost.pem -keyout /etc/nginx/ssl/localhost.
openssl rsa -noout -text -in /etc/nginx/ssl/localhost.key

# Configure nginx
mv ./tmp/nginx-conf /etc/nginx/sites-available/localhost
ln -s /etc/nginx/sites-available/localhost /etc/nginx/sites-enabled/localhost
rm -rf /etc/nginx/sites-enabled/default

# Configure mysql
service mysql start
echo "CREATE DATABASE wordpress;" | mysql -u root --skip-password
echo "GRANT ALL PRIVILEGES ON wordpress.* TO 'root'@'localhost' WITH GRANT OPTION;" | mysql -u root --skip-password
echo "update mysql.user set plugin='mysql_native_password' where user='root';" | mysql -u root --skip-password
echo "FLUSH PRIVILEGES;" | mysql -u root --skip-password

# Install & Configure phpmyadmin
wget https://files.phpmyadmin.net/phpMyAdmin/4.9.4/phpMyAdmin-4.9.4-all-languages.zip
unzip phpMyAdmin-4.9.4-all-languages.zip -d /var/www/localhost/
mv /var/www/localhost/phpMyAdmin-4.9.4-all-languages /var/www/localhost/phpmyadmin
mv ./tmp/config.inc.php /var/www/localhost/phpmyadmin/config.inc.php

# Boot up php & nginx
service php7.3-fpm start
service nginx start

bash
```

For php configuration file > Again, build the docker image and run it. cd to phpmyadmin. You will find config.sample.inc.php file in this directory. This is the file you have to configure. Create a file called config.inc.php and copy the below configuration. This is the file moved to its directory in your shell script.

PhpMyAdmin config explanation

- I filled alphanum in blowfish_secret part(you can use any random string and number). It is a value that will be unique to your instance and use of PhpMyAdmin.
- Also set AllowNoPassword as true, so you can access PhpMyAdmin without password.

```
<?php
/* vim: set expandtab sw=4 ts=4 sts=4: */
/**
 * phpMyAdmin sample configuration, you can use it as base for
 * manual configuration. For easier setup you can use setup/
 *
 * All directives are explained in documentation in the doc/ folder
 * or at <https://docs.phpmyadmin.net/>.
 *
 * @package PhpMyAdmin
 */
// declare(strict_types=1);

/**
 * This is needed for cookie based authentication to encrypt password in
 * cookie. Needs to be 32 chars long.
 */
$cfg['blowfish_secret'] = 'abcdefghijklmnopqrstuvwxyz0123456789'; /* YOU MUST FILL IN THIS FOR COOKIE AUTH! */
```

```
/**
 * Servers configuration
 */
$i = 0;

/**
 * First server
 */
$i++;
/* Authentication type */
$cfg['Servers'][$i]['auth_type'] = 'cookie';
/* Server parameters */
$cfg['Servers'][$i]['host'] = 'localhost';
$cfg['Servers'][$i]['compress'] = false;
$cfg['Servers'][$i]['AllowNoPassword'] = true;
/**
 * Directories for saving/loading files from server
 */
$cfg['UploadDir'] = '';
$cfg['SaveDir'] = '';
```

Step 8 — Installing WordPress

You have the database set up and now you can install WordPress is a free and open-source CMS. In your shell script, install wordpress by using ‘wget’ and unzip the wordpress file. Also, move the installed file and config file to its relevant web path.

```
# Generate website folder & create index.php & move html file
mkdir /var/www/localhost && touch /var/www/localhost/index.php
echo "<?php phpinfo(); ?>" >> /var/www/localhost/index.php
mv ./tmp/index.html /var/www/localhost/index.html

# Configure nginx
mv ./tmp/nginx-conf /etc/nginx/sites-available/localhost
ln -s /etc/nginx/sites-available/localhost /etc/nginx/sites-enabled/localhost
rm -rf /etc/nginx/sites-enabled/default

# Configure mysql
service mysql start
echo "CREATE DATABASE wordpress;" | mysql -u root --skip-password
echo "GRANT ALL PRIVILEGES ON wordpress.* TO 'root'@'localhost' WITH GRANT OPTION;" | mysql -u root --skip-password
echo "update mysql.user set plugin='mysql_native_password' where user='root';" | mysql -u root --skip-password
echo "FLUSH PRIVILEGES;" | mysql -u root --skip-password

# Install & Configure phpmyadmin
wget https://files.phpmyadmin.net/phpMyAdmin/4.9.4/phpMyAdmin-4.9.4-all-languages.zip
unzip phpMyAdmin-4.9.4-all-languages.zip -d /var/www/localhost/
mv /var/www/localhost/phpMyAdmin-4.9.4-all-languages /var/www/localhost/phpmyadmin
mv ./tmp/config.inc.php /var/www/localhost/phpmyadmin/config.inc.php

# Install & Configure wordpress
wget -c https://wordpress.org/latest.tar.gz
tar -xvzf latest.tar.gz
mv wordpress/ /var/www/localhost
mv ./tmp/wp-config.php /var/www/localhost/wordpress/

# Boot up php & nginx
service php7.3-fpm start
service nginx start

bash
```

Configure WordPress

I configure wordpress as below. All I did was put DB_NAME, DB_USER and DB_PASSWORD on the config.

```
<?php

// ** MySQL settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', 'wordpress' );

/** MySQL database username */
define( 'DB_USER', 'root' );

/** MySQL database password */
define( 'DB_PASSWORD', '' );

/** MySQL hostname */
define( 'DB_HOST', 'localhost' );
```

```
/** Database Charset to use in creating database tables. */
define( 'DB_CHARSET', 'utf8mb4' );

/** The Database Collate type. Don't change this if in doubt. */
define( 'DB_COLLATE', '' );

/**#@+
 * Authentication Unique Keys and Salts.
 *
 * Change these to different unique phrases!
 * You can generate these using the {@link https://api.wordpress.org/secret-key/1.1/salt/ WordPress.org secret-key service}
 * You can change these at any point in time to invalidate all existing cookies. This will force all users to have to log in ag
 *
 * @since 2.6.0
 */
define( 'AUTH_KEY',         'put your unique phrase here' );
define( 'SECURE_AUTH_KEY',  'put your unique phrase here' );
define( 'LOGGED_IN_KEY',    'put your unique phrase here' );
define( 'NONCE_KEY',        'put your unique phrase here' );
define( 'AUTH_SALT',        'put your unique phrase here' );
define( 'SECURE_AUTH_SALT', 'put your unique phrase here' );
define( 'LOGGED_IN_SALT',   'put your unique phrase here' );
define( 'NONCE_SALT',       'put your unique phrase here' );

/**#@-*/

/**
 * WordPress Database Table prefix.
 *
 * You can have multiple installations in one database if you give each
 * a unique prefix. Only numbers, letters, and underscores please!
 */
$table_prefix = 'wp_';

/**
 * For developers: WordPress debugging mode.
 *
 * Change this to true to enable the display of notices during development.
 * It is strongly recommended that plugin and theme developers use WP_DEBUG
 * in their development environments.
 *
 * For information on other constants that can be used for debugging,
 * visit the Codex.
 *
 * @link https://codex.wordpress.org/Debugging_in_WordPress
 */
define( 'WP_DEBUG', false );

/* That's all, stop editing! Happy publishing. */

/** Absolute path to the WordPress directory. */
if ( ! defined( 'ABSPATH' ) ) {
  define( 'ABSPATH', dirname( __FILE__ ) . '/' );
}

/** Sets up WordPress vars and included files. */
require_once( ABSPATH . 'wp-settings.php' );
```

There you go. You have successfully installed WordPress 5 with Nginx on Debian 10 Buster.

Step 9 — Create your Wordpress & have a working Wordpress site at localhost/wordpress

We can then go to localhost / wordpress which normally leads to the install menu of wordpress. We will be asked for the name of the db, the name of the sql user, the password, the name of the server etc: 'wordpress', 'wordpress', 'password', 'localhost', etc.

Wordpress will either create or ask us to create (and copy the content into) the 'wp-config.php' file. It should be in the 'wordpress /' folder. If we are not redirected automatically, we can now go to:

- localhost / wordpress: the home page of our wordpress site
- localhost / wordpress / wp-admin: the dashboard of our wordpress site

Wordpress is now installed thanks to the wp-config file. But if you leave the container, this file is lost and you will have to redo the installation of wordpress each time you relaunch a container. It is therefore necessary to copy the wp-config file obtained (copy / paste for example) and place it in the wordpress folder of the repo so that Wordpress is always installed after a rebuild / rerun.

On the other hand, the database structure created by and for Wordpress in the 'wordpress' database will be lost. The solution is to make a backup file of the state of the database and import it at each rerun. The base will therefore be recreated and reimported at each rerun (this takes more or less time depending on the weight of the base) which will keep the installation complete. For that we will use phpMyAdmin.

For this project, keep in mind that when you run your docker container, you should not have to do ANYTHING and you should just be able to visit https://localhost/wordpress/ and you should see a working Wordpress site, not the wordpress installer page. So, it's necessary to make sure your wordpress data is not lost.

Now, go to 'localhost/phpMyAdmin' to access the PMA login screen, just log in with the mysql wordpress user: root and no password. We can view our databases in the database tab. You can view all the tables in the wordpress database as well as their content. If we post a comment on wordpress, we can also see it on phpMyAdmin. We will be able to recover the backup file of the state of the wordpress database. Just click on the 'Export' tab, and retrieve the 'wordpress.sql' file (the browser downloads it).

We can now place this file in our sources then add an instruction to the run script to import the database.

src /init.sh line: 38: `mysql wordpress -u root --password=  < /var/www/wordpress.sql`

Don't forget to add this file to your sources and add `COPY ./srcs/wordpress.sql /var/www/` to your Dockerfile!

When you build and run, you should be able to see a working wordpress instead of the wordpress install page.

Step 10 — Configure SSL for Docker

We configured nginx port 80(HTTP) and also 443(HTTPS). Since config setting for 443 port exists, we have to set SSL key in the shell script.

```
# Generate website folder & create index.php & move html file
mkdir /var/www/localhost && touch /var/www/localhost/index.php
echo "<?php phpinfo(); ?>" >> /var/www/localhost/index.php
mv ./tmp/index.html /var/www/localhost/index.html

# Config Access
chown -R www-data /var/www/*
chmod -R 755 /var/www/*

# SSL
mkdir /etc/nginx/ssl
openssl req -newkey rsa:4096 -x509 -sha256 -days 365 -nodes -out /etc/nginx/ssl/localhost.pem -keyout /etc/nginx/ssl/localhost.
openssl rsa -noout -text -in /etc/nginx/ssl/localhost.key

# Configure nginx
mv ./tmp/nginx-conf /etc/nginx/sites-available/localhost
ln -s /etc/nginx/sites-available/localhost /etc/nginx/sites-enabled/localhost
rm -rf /etc/nginx/sites-enabled/default

# Configure mysql
service mysql start
echo "CREATE DATABASE wordpress;" | mysql -u root --skip-password
echo "GRANT ALL PRIVILEGES ON wordpress.* TO 'root'@'localhost' WITH GRANT OPTION;" | mysql -u root --skip-password
echo "update mysql.user set plugin='mysql_native_password' where user='root';" | mysql -u root --skip-password
echo "FLUSH PRIVILEGES;" | mysql -u root --skip-password
mysql wordpress -u root --password=  < /var/www/wordpress.sql

# Install & Configure phpmyadmin
wget https://files.phpmyadmin.net/phpMyAdmin/4.9.4/phpMyAdmin-4.9.4-all-languages.zip
unzip phpMyAdmin-4.9.4-all-languages.zip -d /var/www/localhost/
mv /var/www/localhost/phpMyAdmin-4.9.4-all-languages /var/www/localhost/phpmyadmin
mv ./tmp/config.inc.php /var/www/localhost/phpmyadmin/config.inc.php

# Install & Configure wordpress
wget -c https://wordpress.org/latest.tar.gz
tar -xvzf latest.tar.gz
mv wordpress/ /var/www/localhost
mv ./tmp/wp-config.php /var/www/localhost/wordpress/

# Boot up php & nginx
service php7.3-fpm start
service nginx start

bash
```

Step 11 — Change Authorisation and Init bash

```
# Generate website folder & create index.php & move html file
mkdir /var/www/localhost && touch /var/www/localhost/index.php
echo "<?php phpinfo(); ?>" >> /var/www/localhost/index.php
mv ./tmp/index.html /var/www/localhost/index.html

# Config Access
chown -R www-data /var/www/*
chmod -R 755 /var/www/*

# SSL
mkdir /etc/nginx/ssl
openssl req -newkey rsa:4096 -x509 -sha256 -days 365 -nodes -out /etc/nginx/ssl/localhost.pem -keyout /etc/nginx/ssl/localhost.
openssl rsa -noout -text -in /etc/nginx/ssl/localhost.key

# Configure nginx
mv ./tmp/nginx-conf /etc/nginx/sites-available/localhost
ln -s /etc/nginx/sites-available/localhost /etc/nginx/sites-enabled/localhost
rm -rf /etc/nginx/sites-enabled/default

# Configure mysql
service mysql start
echo "CREATE DATABASE wordpress;" | mysql -u root --skip-password
echo "GRANT ALL PRIVILEGES ON wordpress.* TO 'root'@'localhost' WITH GRANT OPTION;" | mysql -u root --skip-password
echo "update mysql.user set plugin='mysql_native_password' where user='root';" | mysql -u root --skip-password
echo "FLUSH PRIVILEGES;" | mysql -u root --skip-password

# Install & Configure phpmyadmin
wget https://files.phpmyadmin.net/phpMyAdmin/4.9.4/phpMyAdmin-4.9.4-all-languages.zip
unzip phpMyAdmin-4.9.4-all-languages.zip -d /var/www/localhost/
mv /var/www/localhost/phpMyAdmin-4.9.4-all-languages /var/www/localhost/phpmyadmin
mv ./tmp/config.inc.php /var/www/localhost/phpmyadmin/config.inc.php

# Install & Configure wordpress
wget -c https://wordpress.org/latest.tar.gz
tar -xvzf latest.tar.gz
mv wordpress/ /var/www/localhost
mv ./tmp/wp-config.php /var/www/localhost/wordpress/

# Boot up php & nginx
service php7.3-fpm start
service nginx start

bash
```

References

https://www.techopedia.com/definition/22441/system-administration

https://docs.docker.com/get-started/overview/

https://medium.com/dev-genius/docker-explained-in-simple-terms-178748e28e99

https://gist.github.com/bradtraversy/89fad226dc058a41b596d586022a9bd3

https://harm-smits.github.io/42docs/projects/ft_server

https://www.digitalocean.com/community/tutorials/how-to-install-linux-nginx-mariadb-php-lemp-stack-on-debian-10

https://forhjy.medium.com/how-to-install-lemp-wordpress-on-debian-buster-by-using-dockerfile-1-75ddf3ede861

https://www.ctl.io/developers/blog/post/dockerfile-entrypoint-vs-cmd/