Docker 是一种容器化技术，可以独立使用来运行单个或少量的容器，核心组件包括 Docker 镜像（容器的模板）和 Docker 容器（运行中的实例）。

Kubernetes 是一个 容器编排平台，可以基于多种容器运行时（如 Docker、containerd、CRI-O）来部署和管理容器化应用，Kubernetes 提供了许多 Docker 无法提供的高级功能，比如：集群管理、自动扩展、负载均衡、自愈能力

两者的关系：Docker 是一个运行时，用来构建和运行容器。Kubernetes 是一个编排工具，用来管理多个容器的运行和协作。可以把 Docker 理解为创建和运行容器的引擎，而 Kubernetes 是负责管理调度这些容器的系统。

什么是Docker？

Docker是一种可以让你在不同机器和操作系统上以独立且一致的方式建立、部署和运行应用的工具。这个过程是通过容器来实现的。容器是一种轻量级虚拟化的环境，它将所有依赖和应用所需的代码打包到单个文本文件中，这个文件可以在任意机器上以相同的方式运行。

Docker主要用于在容器中打包和运行应用，但它并不仅限于此。Docker还可以用于创建和运行其他类型的容器，比如用于测试、开发、或实验的容器。

什么是Docker image？

它是一个轻量级的可执行包，包含应用运行所需的所有一切，比如代运行时环境、系统工具，库和依赖。

虽然它不能保证零出错，但因为应用的行为最终依赖很多镜像本身之外的因素，所以使用Docker可以减少不可预期错误的概率。

Docker镜像是从Dockerfile创建的，Dockerfile文件是一个简单的文本文件，它包含一套建立镜像的指令，每个指令都会在镜像中创建一个新的层。

什么是容器？

容器是将代码及其依赖打包使应用从一个计算环境到另一个计算环境快速可靠地运行的标准软件单元。

容器是将代码和依赖一起打包的在应用层面的抽象概念。多个容器可以运行在同一个机器上并与其他容器共享OS Kernel内核。每一个容器都在用户空间独立运行。容器比虚拟机（容器镜像通常是几十MBs大小）占用更少的空间，它可以处理更多的应用，并需求更少的虚拟机和操作系统。

**容器和虚拟机有相似的资源隔离和分配好处但功能不同，因为容器虚拟化操作系统而不是使用硬件。容器更便捷高效。**

什么是虚拟机？

虚拟机是把一个服务器变成多个服务器的物理硬件的抽象概念。虚拟机管理程序（hypervisor）允许在一个机器上运行多个虚拟机。每一个虚拟机都包含一整套完整的操作系统，应用，必要的二进制和库的拷贝。所以它占用几十GBs。所以虚拟机启动很慢。

为什么选择Docker？它解决了什么问题？

在前容器时代，当一个开发者开发了一套代码进行在机器上测试，但测不了，原因可能多种多样，可能是因为测试需要安装让代码能正常运作的依赖，也可能是某些环境变量需要添加但在测试的机器上却没有。这就意味着我们不得不解决这个问题。

所以Docker就应运而生。

但为什么不用虚拟机而用Docker呢？我们看看二者的区别。

|  |  |
| --- | --- |
| Virtual Machine | Docker |
| 占用大量内存空间 | 占用很少的内存空间 |
| 启动很慢 | 快速启动（因为它使用你正在用的内核） |
| 很难扩展 | 超级容易扩展 |
| 低效 | 高效 |
| 卷存储不能跨虚拟机共享 | 卷存储可以跨主机和容器共享 |

Virtual Machine | Virtual Machine | Virtual Machine

App A. | App B | App C

Guest OS. | Guest OS | Guest OS

Hypervisor

Infrastructure

**PID 1**

这是初始进程的标识符，系统启动时的第一个进程，它负责启动或停止系统中别的进程。在docker容器中也是，初始进程负责启动和停止容器中运行的应用。

但是dicker容器中的PID 1表现行为和基于Unix的系统中的初始进程不同（他们不是同一个东西！）

Daemon进程不是PID 1.它是持续运行在系统上并执行特定任务的背景进程。而PID 1 是在基于Unix 系统上的kernel内核启动的第一个进程，在系统里起特殊作用。

containerized applications

App A / App B / App C

Docker

Host Operating System

Infrastructure

Infrastructure - 就像CPU，硬件和其他物理组件的计算机组件。

Host Operating System - 就像Linux和Mac和windows的用于计算机运行的操作系统

Docker engineer - docker engineer时基础引擎，安装在主机上，用于建立和运行使用docker组件和服务的容器。

App - 与别的容器隔离的运行的容器。

Docker Engine

Docker CLI（Client） -> RESTful APIs -> Docker Daemon(Server)

Docker engine - a lightweight runtime and packaging tool that bundles your application and its dependencies into a single package

Docker engine包含Docker daemon（server端）和Docker client（client端）

Docker daemon是管理docker容器的背景守护进程

Docker client是命令行工具，可以让你和Docker daemon交互

Docker engine运作原理：Dockerfile（包含创建Docker image的指令的text文件）Docker image（轻量级独立的可执行的镜像包，它包含所有运行软件所需的应用代码，库，依赖，运行时）

Docker client建立Docker image并指明Dockerfile的路径 - **docker build**

Docker daemon读Dockerfile里的指令创建Docker image

Docker client运行Docker image并把它作为容器 - **docker run**

Docker daemon从Docker image里创建一个容器并运行容器里的应用

Docker engine提供安全独立的环境让应用运行，也负责管理如CPU，内存，容器的存储之类的

可以使用Docker client查看、停止、管理所有容器，也可以使用Docker client推送Docker image到注册状态registry，比如Docker Hub，这样它就可以和别的共享。

**Dockerfile**是一个包含创建Docker image指令的text file，它写明了要用的基础镜像，要安装的依赖和软件，其他应用运行要搭建环境的所有配置或脚本。

Dockerfile包含的一系列命令自动化配置和建立镜像，比如安装包，拷贝文件，设置环境变量。每个命令都会在镜像中创建一个新的层。

**Docker Compose**是一个简化部署和管理多容器Docker应用的强大工具。它提供了几大好处，包括简化定义相关服务、数据持久保存的卷、以及容器连通的网络等流程。用它可以轻松配置每个服务的设置，包括使用的镜像、公开的端口，以及要设置的环境变量…

总的来说，Docker Compose简化了开发流程，使构建和交付应用更加轻松高效。

Docker Compose有**3大部分**：

**服务**：Docker Compose中的工作单元，有名字，它定义了容器镜像，一套环境变量，一整需要公开给主机的端口。运行docker-compose up， Docker就会为每一个服务创建一个新容器。

**网络**：网络是容器相互连通的方式，在Docker Compose中创建一个网络，Docker将会创建一个让所有其他容器都能连通的新网络。这使容器相互连通无需知道彼此的IP，只需知道名字即可。

**卷**：卷是存储容器间共享数据的方式。在Docker Compose中创建一个卷，Docker就会创建一个所有容器都能访问的新卷（换句话说就是文件夹）。这使容器间可以共享数据无需在你每次想用数据时使用复制粘贴。

Docker Compose file是YAML文件，它定义了Docker容器应创建运行的多个Docker容器，定义组成应用的服务，用单行命令启动和停止所有容器

二者目标不同：前者用来创建单个docker image，后者用于定义运行多个docker容器作为一个单个的应用。

二者格式不同：前者是特定格式和符号的普通text文件，后者是用YAML写的

二者范围不同：前者焦点在于创建单一的image，后者侧重于定义运行多个容器作为一个单个的应用。

二者命令不同：前者用FROM, RUN, 和CMD，后者用services, volumes, and networks来定义容器和它们应该怎么建立和运行

FROM - 说明基础image用作创建Docker image的起始点，基础image提供这个image的基础层

RUN - 在容器终端执行命令。典型地用于安装应用所需的软件或库

CMD - 指明当容器从image中启动时应当运行的默认命令。用于指明当容器启动之初应当运行的主命令

services - 用于定义组成应用的所有服务。一个服务就是运行特定应用或组件的容器

volumes - 用于定义应用的持久存储。一个volume就是附着在容器上的一份存储，它用于存储当容器停止或移除时应当持久保留的数据。

networks - 用于定义容器应当连接的网络。这个网络是虚拟的，用于连接多个容器，允许它们互相通信。

Docker cli常用命令

docker build - 从Dockerfile里建立Docker image

docker run - 运行基于Docker image的Docker容器

docker pull - 从诸如Docker Hub之类的registry里拉取Docker image

docker push - 把Docker image推送到registry

docker ps - 在系统上罗列出正运行的Docker容器

docker stop - 停止正运行的Docker容器

docker rm - 移除Docker容器

docker rmi - 移除Docker image

docker exec - 在正在运行的Docker容器里执行命令

docker logs - 查看一个Docker容器的日志

docker network - 创建和管理网络。例子：创建新的bridge网络：docker network create my-network

docker volume - 创建管理volumes。例子：创建新的volume：docker volume create my-volume

把volume加载到容器里，启动容器时加-v：docker run -v my-volume:/var/lib/mysql mysql

Docker Compose是一个定义运行多容器Docker应用的工具。用YAML配置应用的服务。用单行命令创建启动配置中的所有服务

通过在一个单一的地方定义所有的服务切轻松启动停止它们，简化管理多容器应用的进程，它通过增减服务副本让扩展应用变得也很容易。

Docker Compose的常用命令：

up: 创建启动容器

down: 停止和移除容器，网络，images以及volumes

start: 启动已存在的容器

stop: 停止正运行的容器

restart: 重启正运行的容器

build: 建立images

ps: 罗列容器清单

logs: 从容器里查看结果

exec: 在正运行的容器中运行一个命令

pull: 从registry中拉取images

push: 把images推送到registry

Docker中的网络是定义容器之间连接网络的虚拟软件。它使得容器之间以及和外界之间可以通信，它在深层的网络基础之上提供了一个额外的抽象层级。

可在Docker中创建的几种网络：

Bridge：这是在安装Docker时默认的网络类型。它允许容器之间并且和主机之间通信，但不能和外界连通。

Host：用主机网络的栈的主机网络，它不提供主机和容器之间的隔离。

Overlay：它允许容器在不同的Docker主机上运行并相互通信

Macvlan：它允许容器在和主机相同的子网段内有自己的IP地址。

命令：docker network create my-network

Docker volume是一个持久存储的位置，用于存储来自容器的数据。即使容器被删除后也能持久存储这些数据，且这些数据可以在容器之间共享。

两种volumes：

Bind mount：主机上的文件或目录，它会被加载到容器里。对它所做的任何更改都会反映到主机上和所有加载相同文件目录的容器上

Named volume：Docker创建管理的volume，存储于主机的特定位置，不和主机的特定文件目录绑定。它对存储那些需要在容器之间共享的数据特别有用，因为它可以轻松地附加到容器上或从容器移除

命令例子：

docker run -v my-volume:/var/lib/mysql mysql

启动一个容器运行mysql image，把my-volume加载到/var/lib/mysql，任何写到容器这个位置的数据都会被永久保存，即使容器被删除。

也可以用Docker Compose创建管理volumes，例如：

version: '3'

services:

db:

image: mysql

volumes:

- db-data:/var/lib/mysql

volumes:

db-data:

上面就是定义了一个db-data volume并把它附加在db服务的/var/lib/mysql位置。写到这个位置的数据都会被永久保存。

MariaDB比MySql性能更好，安全更高，支持新的存储引擎和数据类型

安装：

1.Pull debian:buster (基础image)

2.更新package manager： apt-get update -y

3.安装mariadb服务器：apt-get install mariadb-server -y

4.进入/etc/mysql/mariadb.conf.d/50-server.cnf 把第28行的bind-address = 127.0.0.1改成bind-address = 0.0.0.0 这样所有网络都能连到mariadb了

5. service mysql start

6.创建数据库和用户，给他所有访问数据库的权限FLUSH PRIVILEGES

脚本部分：（这部分 Shell 脚本通常写在 Dockerfile、entrypoint 脚本 或 手动执行的 Shell 脚本中）

（比如把如下脚本存入init-db.sh，让Docker在启动时自动执行）

#!/bin/bash

#db\_name = Database Name

#db\_user = User

#db\_pwd = User Password

echo "CREATE DATABASE IF NOT EXISTS $db\_name ;" > db1.sql

echo "CREATE USER IF NOT EXISTS '$db\_user'@'%' IDENTIFIED BY '$db\_pwd' ;" >> db1.sql

echo "GRANT ALL PRIVILEGES ON $db\_name.\* TO '$db\_user'@'%' ;" >> db1.sql

echo "ALTER USER 'root'@'localhost' IDENTIFIED BY '12345' ;" >> db1.sql

echo "FLUSH PRIVILEGES;" >> db1.sql

mysql < db1.sql

保持这个容器running运行在Dockerfile里（/usr/bin/mysqld\_safe）CMD里的这个命令

当系统启动时，mysqld\_safe用于启动MySQL服务器，它也用于手动启动和停止MySQL服务器

解释：在Docker中，一个容器（Container）本质上是一个运行中的进程，它是基于某个镜像（Image）启动的。当运行一个容器时，它的生命周期依赖于它的主进程（PID 1 进程）。如果这个主进程退出，容器就会停止。在Dockerfile里，如果希望 容器一直保持运行（Running），就需要确保 CMD 或 ENTRYPOINT 启动的进程不会退出。

在MariaDB 或 MySQL 的Docker容器里，通常使用 mysqld\_safe 来确保数据库进程始终运行。

mysqld\_safe 是一个管理 MySQL / MariaDB 服务器的脚本，它的作用：

确保 MySQL 服务器自动重启（如果崩溃）

管理 MySQL 服务器的日志

作为 Docker 容器的主进程（PID 1），保持容器运行

在Dockerfile里，mysqld\_safe通常作为 CMD 命令：CMD ["/usr/bin/mysqld\_safe"]

（这意味着当 Docker 容器启动时，它会运行 mysqld\_safe 进程，进而启动 MySQL 服务器，并让容器一直保持 Running。）

完整的 Dockerfile 示例：：：

FROM mariadb:latest

# 设置数据库的环境变量

ENV MYSQL\_ROOT\_PASSWORD=root

ENV MYSQL\_DATABASE=mydb

ENV MYSQL\_USER=myuser

ENV MYSQL\_PASSWORD=mypassword

# 复制初始化 SQL 文件（可选）复制并执行数据库初始化脚本

COPY init-db.sh /docker-entrypoint-initdb.d/init-db.sh

RUN chmod +x /docker-entrypoint-initdb.d/init-db.sh

# 确保数据库服务启动并保持运行

CMD ["/usr/bin/mysqld\_safe"]

然后把上面那段shell脚本存入init-db.sh, Docker启动时会自动执行：

#!/bin/bash

# 创建数据库和用户

echo "CREATE DATABASE IF NOT EXISTS $MYSQL\_DATABASE;" > /docker-entrypoint-initdb.d/db1.sql

echo "CREATE USER IF NOT EXISTS '$MYSQL\_USER'@'%' IDENTIFIED BY '$MYSQL\_PASSWORD';" >> /docker-entrypoint-initdb.d/db1.sql

echo "GRANT ALL PRIVILEGES ON $MYSQL\_DATABASE.\* TO '$MYSQL\_USER'@'%';" >> /docker-entrypoint-initdb.d/db1.sql

echo "ALTER USER 'root'@'localhost' IDENTIFIED BY '$MYSQL\_ROOT\_PASSWORD';" >> /docker-entrypoint-initdb.d/db1.sql

echo "FLUSH PRIVILEGES;" >> /docker-entrypoint-initdb.d/db1.sql

# 执行 SQL 文件

mysql < /docker-entrypoint-initdb.d/db1.sql

**Wordpress**

是一个内容管理系统（content management system (CMS)），基于PHP和MySQL。开源平台，用于建立网站，blogs和应用。也是一个大型社区，用户和开发者参与，带来广泛的主题，插件和Wordpress网站功能性扩展的其他工具。

FastCGI

Fast Common Gateway Interface 快速通用网关接口，允许网络服务器和网络应用（比如PHP脚本）通信的协议（protocol）。在网络服务器执行脚本时它比传统的通用网关接口更高效，这个协议包含启动新进程执行每一个脚本。

**PHP-FPM**（FastCGI Process Manager）FastCGI进程管理器是专门设计来使用PHP的FastCGI协议的具体实现。它通过启动负责执行PHP脚本的工作者进程池来工作的。当网络服务器收到PHP脚本的请求，它将这个请求传给其中一个工作者进程，这个进程执行脚本并把结果返回给网络服务器。因为工作者进程可以复用来处理多个请求所以PHP脚本执行就更高效了。

PHP-FPM通常用作mod\_php的替代品，mod\_php是一个把PHP解释器直接嵌入到Apache网络服务器的Apache模块。用PHP-FPM可以提高PHP脚本的性能和扩展性，因为它允许网络服务器和PHP在各自独立的进程运行，它也允许在PHP环境更细颗粒度的控制，因为不同的工作者进程池可以用不同的设置来配置。

安装部分：

1. pull debian:buster (base image)

2. 更新package manager： apt-get -y update && apt-get -y upgrade && apt update -y && apt upgrade -y

3. apt install php-fpm php-mysql -y&& apt install curl -y

php-fpm是专门设计来使用PHP的FastCGI协议的具体实现。php-mysql是允许PHP和MySQL数据库通信的PHP扩展。**curl是用不同网络协议比如HTTP，HTTPS，FTP传输数据的命令行工具。**

source of info https://developer.wordpress.org/cli/commands/core/

脚本部分：

#!/bin/bash

# create directory to use in nginx container later and also to setup the wordpress conf

mkdir /var/www/

mkdir /var/www/html

cd /var/www/html

# remove all the wordpress files if there is something from the volumes to install it again

rm -rf \*

# The commands are for installing and using the WP-CLI tool.

# downloads the WP-CLI PHAR (PHP Archive) file from the GitHub repository. The -O flag tells curl to save the file with the same name as it has on the server.

curl -O https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar

# makes the WP-CLI PHAR file executable.

chmod +x wp-cli.phar

# moves the WP-CLI PHAR file to the /usr/local/bin directory, which is in the system's PATH, and renames it to wp. This allows you to run the wp command from any directory

mv wp-cli.phar /usr/local/bin/wp

# downloads the latest version of WordPress to the current directory. The --allow-root flag allows the command to be run as the root user, which is necessary if you are logged in as the root user or if you are using WP-CLI with a system-level installation of WordPress.

wp core download --allow-root

mv /var/www/html/wp-config-sample.php /var/www/html/wp-config.php

# change the those lines in wp-config.php file to connect with database

#line 23

sed -i -r "s/database/$db\_name/1" wp-config.php

#line 26

sed -i -r "s/database\_user/$db\_user/1" wp-config.php

#line 29

sed -i -r "s/passwod/$db\_pwd/1" wp-config.php

#line 32

sed -i -r "s/localhost/mariadb/1" wp-config.php (to connect with mariadb database)

# installs WordPress and sets up the basic configuration for the site. The --url option specifies the URL of the site, --title sets the site's title, --admin\_user and --admin\_password set the username and password for the site's administrator account, and --admin\_email sets the email address for the administrator. The --skip-email flag prevents WP-CLI from sending an email to the administrator with the login details.

wp core install --url=$DOMAIN\_NAME/ --title=$WP\_TITLE --admin\_user=$WP\_ADMIN\_USR --admin\_password=$WP\_ADMIN\_PWD --admin\_email=$WP\_ADMIN\_EMAIL --skip-email --allow-root

# creates a new user account with the specified username, email address, and password. The --role option sets the user's role to author, which gives the user the ability to publish and manage their own posts.

wp user create $WP\_USR $WP\_EMAIL --role=author --user\_pass=$WP\_PWD --allow-root

# installs the Astra theme and activates it for the site. The --activate flag tells WP-CLI to make the theme the active theme for the site.

wp theme install astra --activate --allow-root

wp plugin install redis-cache --activate --allow-root

# uses the sed command to modify the www.conf file in the /etc/php/7.3/fpm/pool.d directory. The s/listen = \/run\/php\/php7.3-fpm.sock/listen = 9000/g command substitutes the value 9000 for /run/php/php7.3-fpm.sock throughout the file. This changes the socket that PHP-FPM listens on from a Unix domain socket to a TCP port.

sed -i 's/listen = \/run\/php\/php7.3-fpm.sock/listen = 9000/g' /etc/php/7.3/fpm/pool.d/www.conf

# creates the /run/php directory, which is used by PHP-FPM to store Unix domain sockets.

mkdir /run/php

wp redis enable --allow-root

# starts the PHP-FPM service in the foreground. The -F flag tells PHP-FPM to run in the foreground, rather than as a daemon in the background.

/usr/sbin/php-fpm7.3 -F

什么是WP-CLI？

**WP-CLI**时用于WordPress的命令行界面。是允许用户和WordPress网站交互的工具。它有多种用途，比如自动化任务，调试问题，安装或删除插件，连带主题，管理用户和角色，导出或导入数据，运行数据库查询，等等…

为什么要用WP-CLI？

因为这个工具可以从终端和WordPress网站直接交互，比如可以大大节省手动安装插件和主题、管理用户及其角色、把新的WordPress网站部署到生产服务器等等的时间。

WP-CLI还有助于用最短的时间实现自动化，所以它是一个非常实用的工具。

Nginx

Nginx是可以用作反协议（reverse proxy）,加载平衡器（load balancer）,和HTTP cache的网络服务器。高性能、稳定性和低资源消耗。经常用来为网络应用处理服务器侧请求，也用来服务比如镜像和JavaScript文件的静态内容。除了它的网络服务器功能，它还被配置来处理其他网络协议类型，比如Secure Sockets Layer（SSL）和传输层安全Transport Layer Security (TLS)。它经常和其他软件比如数据库和内容管理系统（CMS）结合使用，用来强健的可扩展的网络应用。

TLS（Transport Layer Security）

传输层安全是一个安全协议，它被用于在互联网上两方之间建立安全的通信。它设计来阻止窃听eavesdropping, 篡改tampering, 以及信息伪造message forgery，并且它还提供验证和整合传输的数据。TLS用于保障广泛的基于互联网的应用的安全性，包括网页浏览、email、文件传输、VPNs（virtual private networks）和实时通讯系统。

TLS的运作是通过使用公钥加密在两方之间建立安全连接。当一个客户想用TLS和服务器通信，它和服务器交换一系列信息来建立安全连接。这个过程包括数字证书交换和加密关键字谈判（negotiation of encryption keys）。一旦连接建立，客户端和服务器就能安全通信了。

TLS继承于Secure Sockets Layer (SSL)协议，SSL开发于1990年代。TLS基于SSL，但它解决了SSL中发现的一些安全漏洞。TLS现在是互联网上安全通信的事实标准，被大量网站和应用用来保护敏感数据。

OpenSSL

这是SSL和TLS的开源实现。广泛用于SL和TLS的作业工具，很多操作系统都有。

OpenSSL可用于与SSL和TLS有关的任务，包括：

创建管理SSL/TLS证书和私钥、设置和配置启用SSL/TLS的服务器、作为客户端连接启用SSL/TLS的服务器、调试SSL/TLS连接、生成和签署数字证书

OpenSSL经常被系统管理员和开发者用来保护服务器和客户端安全通信，或在互联网上创建数据传输的安全通道，它也被用来为email、文件传输和其他互联网通信创建安全加密的连接。

安装：

1. pull debian:buster (base image)

2. 更新package manager apt update -y && apt upgrade -y

3. apt install -y nginx && apt install openssl -y 这将安装NGINX web server and the OpenSSL tool.

4. openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/nginx-selfsigned.key -out /etc/ssl/certs/nginx-selfsigned.crt -subj "/C=MO/L=KH/O=1337/OU=student/CN=[sahafid.1337.ma](http://sahafid.42.ma/)"

这个命令用OpenSSL生成自签名的SSL/TLS证书和私钥。

Req命令用来生成证书签名请求certificate signing request (CSR)或自签名证书self-signed certificate.

-x509选项告诉OpenSSL生成自签名证书而不是CSR。

-nodes选项告诉OpenSSL不要用passphrase加密私钥。这意味着，私钥将不用密码保护并且它会被以plaintext的形式保存。通常在生产环境不推荐使用。因为这易使key被未验证访问攻击。

-days选项指明证书有效的天数。这里有效期是365天

-newkey选项指明应生成新的密钥。rsa:2048参数告诉OpenSSL用2048位长生成一个RSA密钥（Rivest-Shamir-Adleman Key）（非对称加密算法asymmetric encryption algorithm，用于数据加密和数字签名）。

-keyout选项指明私钥应存储在哪个文件，-out选项表明证书应存储在哪个文件。

-subj选项表明证书主题。包括使用该证书的组织信息，安装证书的服务器信息。这里主题包括country (C=MO), the location (L=KH), the organization (O=1337), the organizational unit (OU=student), and the common name (CN=sahafid.42.ma)

运行上面的命令后，一个自签名SSL/TLS证书和密钥将被生成并存储在特定的文件。可用这些文件配置Nginx服务器并使用SSL/TLS加密。注意：大多数网页浏览器都不信任自签名证书，所以在生产环境需要从可信任的证书权威certificate authority（CA）获取证书。

server {

# The server listens for incoming connections on port 443, which is the default port for HTTPS traffic. The server listens for both IPv4 and IPv6 connections

listen 443 ssl;

listen [::]:443 ssl;

# replace login with your own loggin

server\_name www.login.42.fr login.42.fr;

# The ssl\_certificate and ssl\_certificate\_key directives specify the locations of the SSL/TLS certificate and private key, respectively, that will be used to encrypt the traffic. The ssl\_protocols directive specifies the TLS protocols that the server should support.

ssl\_certificate /etc/ssl/certs/nginx-selfsigned.crt;

ssl\_certificate\_key /etc/ssl/private/nginx-selfsigned.key;

# We are using version 1.3 of TLS

ssl\_protocols TLSv1.3;

# The index directive specifies the default file that should be served when a client requests a directory on the server. The root directive specifies the root directory that should be used to search for files.

index index.php;

root /var/www/html;

# The location directive defines a block of configuration that applies to a specific location, which is specified using a regular expression. In this case, the regular expression ~ [^/]\\.php(/|$) matches any request that ends in .php and is not preceded by a / character.

location ~ [^/]\\.php(/|$) {

# The try\_files directive attempts to serve the requested file, and if it does not exist, it will return a 404 error.

try\_files $uri =404;

#The fastcgi\_pass directive passes the request to a FastCGI server for processing.

fastcgi\_pass wordpress:9000;

# The include directive includes a file with FastCGI parameters.

include fastcgi\_params;

#The fastcgi\_param directive sets a FastCGI parameter. The SCRIPT\_FILENAME parameter specifies the path to the PHP script that should be executed.

fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;

}

}

简洁版：

server {

listen 443 ssl;

listen [::]:443 ssl;

server\_name www.login.1337.ma login.1337.ma;

ssl\_certificate /etc/ssl/certs/nginx-selfsigned.crt;

ssl\_certificate\_key /etc/ssl/private/nginx-selfsigned.key;

ssl\_protocols TLSv1.3;

index index.php;

root /var/www/html;

location ~ [^/]\\.php(/|$) {

try\_files $uri =404;

fastcgi\_pass wordpress:9000;

include fastcgi\_params;

fastcgi\_param SCRIPT\_FILENAME $document\_root$fastcgi\_script\_name;

}

}

可以copy这段配置到文件：/etc/nginx/sites-available/default

为了保持容器一直运行，需使用命令： nginx -g daemon off;

nginx -g daemon off;命令用来在前台foreground启动Nginx网络服务器，并禁用守护进程（daemon）模式。

在Nginx中，daemon指令用于启用和禁用守护进程模式，这决定了Nginx进程如何运行。daemon启用，Nginx进程在后台运行并与终端分离，daemon禁用时，Nginx进程在前台运行保持和终端连接。

**Adminer**

是管理数据库系统的开源工具。通过它的用户友好界面，可以轻松地查看、编辑、创建、修改数据库。它是为协同数据库工作提供用户友好界面的PHP文件。用Adminer可以创建和删除数据库，创建拷贝更改和删除数据表，删除编辑和增加字段，执行SQL命令，管理用户和权限。它广泛支持各种数据库系统，如MySQL，MariaDB，PostgreSQL，SQLite，Oracle和MS SQL。它是开源软件，可在任何安装PHP的服务器上使用。它是不需要安装的单一文件应用。

安装：

1. pull debian:buster (base image)

2. 更新package manager： apt update -y

3. apt install wget -y && apt-get install php php-mysql -y

wget包是一个从互联网下载文件的实用程序。

PHP是编程语言，常用于开发动态网络应用。PHP MySQL模块允许PHP和MySQL数据库通信。运行上面的命令后，就有可以和MySQL数据库交互的PHP网络应用了。

**FTP**（File Transfer Protocol）

它是一个标准网络协议，用于从一个主机到另一个基于TCP网络（比如互联网）的主机之间传输文件。FTP是建立在客户-服务器的架构上的，在客户端和服务器端使用单独的控制和数据连接。它为用户提供了在远程服务器上传、下载和管理文件的健壮机制。

FTP在2个主机之间（客户端和服务器）开放两种连接，一种是两个客户端之间发送的命令和回应，一种是处理数据传输的连接。

它使客户端可以上传下载文件，创建删除目录，罗列远程服务器上的目录内容清单。FTP客户端可用于所有主流的操作系统和大多数支持FTP的现代网页浏览器。

FTP的主动和被动模式连接的区别：

主动模式下，客户端发起连接服务器端传输文件。被动模式下，客户端发起连接服务器请求数据传输，服务器发起连接到客户端传递数据。

被动模式常用语客户端处于防火墙或NAT(Network Address Translation) 设备之后，因为客户端的防火墙或NAT会拦截从服务器端发起的连接。被动模式允许客户端建立和服务器的连接，然后服务器端能建立回客户端的连接并传递数据。

在使用FTP之前，需要安装FileZilla或者其他可选的工具。

安装：

1. pull debian:buster (base image)

2. 更新package manager： apt update -y

3. apt install vsftpd -y

vsftpd是系统上的vsftpd包。它代表“Very Secure FTP Daemon”，它是一个用于Unix之类系统的流行的FTP服务器软件。一旦安装，就可以用vsftp在系统上建立和管理FTP服务器了。

脚本：

#!/bin/bash

service vsftpd start

# Add the USER, change his password and declare him as the owner of wordpress folder and all subfolders

adduser $ftp\_user --disabled-password

echo "$ftp\_user:$ftp\_pwd" | /usr/sbin/chpasswd &> /dev/null

echo "$ftp\_user" | tee -a /etc/vsftpd.userlist &> /dev/null

mkdir /home/$ftp\_user/ftp

chown nobody:nogroup /home/$ftp\_user/ftp

chmod a-w /home/$ftp\_user/ftp

mkdir /home/$ftp\_user/ftp/files

chown $ftp\_user:$ftp\_user /home/$ftp\_user/ftp/files

sed -i -r "s/#write\_enable=YES/write\_enable=YES/1" /etc/vsftpd.conf

sed -i -r "s/#chroot\_local\_user=YES/chroot\_local\_user=YES/1" /etc/vsftpd.conf

echo "

local\_enable=YES

allow\_writeable\_chroot=YES

pasv\_enable=YES

local\_root=/home/sami/ftp

pasv\_min\_port=40000

pasv\_max\_port=40005

userlist\_file=/etc/vsftpd.userlist" >> /etc/vsftpd.conf

service vsftpd stop

/usr/sbin/vsftpd

1.脚本用service命令启动vsftp服务

2.用adduser命令添加新用户，通过$ftp\_user变量指明用户名。-disabled-password标识表明用户不能用密码登陆

3.然后脚本将$ftp\_pwd变量设置为用户密码

4.把新用户名添加到/etc/vsftpd.userlist文件中，该文件通过vsftpd表明哪些用户可以登录到FTP服务器。

5.脚本在/home/$ftp\_user/ftp创建了一个新目录，并把所有者设置为nobody:nogroup。然后设置改目录的权限，这样它就不能被写入。

6.脚本在/home/$ftp\_user/ftp/files创建了另一个目录，并把所有者设为$ftp\_user:$ftp\_user

7.脚本修改了/etc/vsftpd.conf文件，为本地用户启用了writing写入和chroot（比如jailing），还启动了被动模式，写明了被动连接模式下可用的端口范围。

8.local\_enable=YES：这个指令告诉vsftp允许本地用户登陆FTP服务器。

9.allow\_writeable\_chroot=YES：这个指令告诉vsftp允许本地用户对它们的home目录有写入权限。

10. pasv\_enable=YES：这个指令告诉vsftp允许被动连接模式。

11. local\_root=/home/sami/ftp：这个指令表明本地用户的根目录。所有本地用户将会被jailed（比如限制restricted）访问这个目录和它的子目录。

12. pasv\_min\_port和pasv\_max\_port：这俩个指令表明在被动模式下服务器可以与客户端建立起回连接的端口范围。

13. userlist\_file=/etc/vsftpd.userlist：这个指令表明包含允许登陆FTP服务器的用户列表的文件。

14. 脚本停止vsftp服务，并使用/usr/sbin/vsftpd命令重启它。

**REDIS cache** （REDIS Object Cache）

Redis是开源的、内存数据结构存储，可以用作数据库，缓存和信息代理。它是EorPress用于提高网站性能的插件，它在内存内存储要访问的数据，而不是每次要用到的时候查询数据库。它支持大量的数据结构，比如字符串、哈希、链表、集合等等。它的一个主要特征是它能够在内存中缓存数据，这使其能够实现非常快速的读写速度。Redis能通过缩短从数据库或其他慢速存储层访问数据所需的时间，从而提高网页应用性能，有更好的用户体验有助于在搜索引擎中提高等级。另外因为它的缓存能力，Redis也提供发布和订阅发送信息、交易和支持多种数据结构，它是一个多种用途的多功能工具。

安装：

1. pull debian:buster (base image)

2. 更新package manager： apt update -y

3.

echo "maxmemory 256mb" >> /etc/redis/redis.conf

echo "maxmemory-policy allkeys-lru" >> /etc/redis/redis.conf

sed -i -r "s/bind 127.0.0.1/#bind 127.0.0.0/" /etc/redis/redis.conf

这些命令将在系统上安装Redis服务器，并配置使用最大256MB的内存。maxmemory指令设置Redis允许使用的最大内存。

maxmemory-policy指令表明Redis在达到最大内存限制时应使用的政策。这里allkeys-lru政策将使Redis移除最近最少使用的键以释放内存。

因为用太大的内存会导致性能问题甚至系统崩溃，所以仔细设置最大内存特别重要，还应考虑Redis实例将处理的负载类型，因为不同的负载可能会有不同的内存需求。

最后命令更改了redis.conf配置文件，并注释掉bind指令，该指令表明Redis服务器应监听的IP地址。通过注释掉这个指令，Redis服务器将监听所有可得的网络接口。

为了保持Redis CMD ["redis-server", "--protected-mode", "no"]

redis-server命令用于启用Redis服务器，它是可用于存储诸如字符串、哈希表、链表和集合的数据结构的永久性键值对存储。

--protected-mode选项表明Redis服务器是否应该在保护模式下运行。保护模式是Redis 4.0版本引入的安全特性。它可以防止Redis服务器接受不在同一主机上运行的客户端的连接。

通过no参数，--protected-mode选项禁用了保护模式，并允许Redis服务器接受运行在所有主机上的客户端的连接。

把这些命令行添加到wp-config.php：

define( 'WP\_REDIS\_HOST', 'redis' ); define( 'WP\_REDIS\_PORT', 6379 ); define('WP\_CACHE', true);

CADVISOR（Extra service）

（Container Advisor）是Google开发的用于监管和分析资源使用和容器性能的开源工具。它提供了关于单个容器资源使用和主机本身的资源使用的信息。

cAdvisor在主机上作为守护进程运行，从操作系统和容器收集资源使用和性能数据。它通过基于网页的界面和一套RESTful APIs公开这些数据，这可以实时地监管和分析容器的资源使用情况。

cAdvisor在生产环境中对监管和优化容器应用的性能特别有用。有助于识别资源限制，查找问题，优化资源分配以改善容器性能。

安装：

1. pull debian:buster (base image)

2. 更新package manager： apt update -y

3. apt install wget -y (wget包, 是从网页下载文件的实用程序)

4. wget [https://github.com/google/cadvisor/releases/download/v0.37.0/cadvisor](https://github.com/google/cadvisor/releases/download/v0.37.0/cadvisor) (从github下载cadvisor)

5. chmod +x cadvisor 更改可执行文件的权限

6. 运行这个可执行文件 ./cadvisor 并在网页中找到stats状态。

注意：确保在docker-compose文件中公开8080端口，并连接正确的卷volumes，你可以参考上面的docker-compose文件如何写的。

RUN 是在**构建镜像时**执行的命令，CMD 是**运行容器时**执行的。

**Portainer** 是一个 **Docker 容器管理工具**，它提供了 **Web 界面**，可以用来 **可视化管理 Docker 容器、镜像、网络、卷** 等。

50-server.cnf 是 **MariaDB/MySQL 服务器的配置文件**，用于定义数据库服务 (mysqld) 的行为，比如数据存储路径、端口、网络访问权限等。它属于 **MySQL/MariaDB 的配置系统**，通常位于 /etc/mysql/mariadb.conf.d/ 或 /etc/mysql/mysql.conf.d/ 目录下。

**.cnf 后缀的含义**

* .cnf 是 **“configuration”**（配置）的缩写，专门用于 **MySQL/MariaDB** 的配置文件。
* MariaDB/MySQL **默认只会读取 .cnf 后缀的配置文件**，而不会读取 .conf 文件。
* 其他服务（如 nginx、apache）通常使用 .conf，但 **MariaDB/MySQL 采用 .cnf 作为约定**。

MariaDB/MySQL 的配置文件通常遵循 **数字开头的命名规则**：

** 00-xxx.cnf → 最先加载**

** 50-server.cnf → MariaDB 服务器核心配置**

** 99-custom.cnf → 用户自定义配置，最后加载**

**50-server.cnf 的文件结构**：

[server]

# 这里的配置会应用到所有MariaDB服务器组件

[mysqld]

datadir = /var/lib/mysql # 数据存储位置

socket = /run/mysqld/mysqld.sock # Unix socket 文件路径

bind-address = \* # 允许所有IP访问（默认只允许本机访问）

port = 3306 # 监听端口

user = mysql # 运行MariaDB的用户

# user = mysql 指的是 **Linux 系统的 mysql 用户，而不是数据库用户**。

# 这个用户 **在 MariaDB 安装时自动创建**，主要作用是 **以低权限身份运行 MariaDB 进程，提高安全性**。

# 相关配置写在 **MariaDB 服务器的 .cnf 配置文件** 中，比如 /etc/mysql/mariadb.conf.d/50-server.cnf。

[embedded]

# 用于嵌入式数据库模式，一般不修改

[mariadb]

# 特定于MariaDB的配置

**Portainer** 是一个 **Docker 容器管理工具**，它提供了 **Web 界面**，可以用来 **可视化管理 Docker 容器、镜像、网络、卷** 等。

作用：

 通过 **Web 界面** 管理 Docker（不需要手动输入 docker 命令）。

 监控容器状态（启动、停止、删除）。

 轻松创建、管理 **Docker Compose** 项目。

 管理 **Docker 卷、网络、镜像** 等。

假设你在 docker-compose.yml 里配置了：

ports:

- "9443:9443"

那么在浏览器里访问：

https://localhost:9443

就可以打开 Portainer 的管理界面。

Portainer 主要用于哪种情况？不想每次都输入 docker ps、docker run 这些命令，而是想要一个 **Web 面板** 来管理所有容器。

curl 是一个 **命令行工具**，用于发送 HTTP 请求。

作用：

 访问 Web 服务器的 API 或网页。

 测试 HTTP 响应。

 下载文件。

**基本用法**： 访问一个网页：curl http://example.com

获取 HTTP 头信息：curl -I http://example.com

下载文件：curl -O https://example.com/file.zip

**示例（测试 Nginx 是否正常工作）**： 如果你的 Nginx 运行在 localhost:8082，可以用 curl 测试：curl http://localhost:8082

如果 Nginx 正常，会返回 HTML 代码。

[www]

user = www-data

group = www-data

listen = 0.0.0.0:9000

listen.owner = www-data

listen.group = www-data

pm = dynamic #**作用**：指定 PHP-FPM 进程管理（Process Manager）模式。pm 选项有 3 种模式：**static**：固定数量的 PHP 进程（适合高流量网站）。**dynamic**（**默认**）：动态管理进程数量（适合一般情况）。**ondemand**：只有在请求到来时才创建进程（适合低流量服务器）。

pm.max\_children = 25 #如果请求数超过 25，新的请求必须等待，或者服务器返回 502 错误

pm.start\_servers = 5 #启动 PHP-FPM 时，**默认创建 5 个 PHP 进程**。

pm.min\_spare\_servers = 2 # 当 PHP-FPM 进程 **空闲时，最少保留 2 个 PHP 进程**。

pm.max\_spare\_servers = 10 # 当 PHP-FPM 进程 **空闲时，最多允许 10 个 PHP 进程**。超过 10 个空闲进程时，会自动销毁多余的进程。

clear\_env = no # 设为 no，表示 PHP-FPM **保留环境变量**，例如 .env 里的变量，否则 PHP 代码中 getenv() 可能无法获取 .env 的值。

这段 www.conf 配置的作用是：

 **设置 PHP-FPM 运行的用户**（www-data）。

 **监听 9000 端口**（或者 Unix Socket）。

 **动态管理 PHP 进程**，根据流量调整进程数量，保证性能和资源利用率。

 **优化 PHP-FPM 进程数量**，避免服务器过载或资源浪费。

ftp的Dockerfile里EXPOSE 2121 21100-21110解读：

在 Dockerfile 中，EXPOSE 指定容器**在运行时监听的端口**，但 **不会自动映射到宿主机**。

EXPOSE <port> [<port>/<protocol>...]

 <port>：容器内部要开放的端口（仅供其他容器访问）。

 /<protocol>（可选）：协议类型，默认为 TCP，例如 80/tcp 或 161/udp。

EXPOSE 2121 👉 **FTP 主控制端口**（用于 FTP 客户端与服务器的通信）。

EXPOSE 21100-21110 👉 **FTP 被动模式的端口范围**（FTP 数据传输使用的端口）。

作用：

 **告诉 Docker** 容器会监听 2121 和 21100-21110 端口。

 **不会自动开放到宿主机**，要在 docker-compose.yml 中用 ports: 手动映射。

**为什么 FTP 需要多个端口？**

FTP **有两种模式**：

 **主动模式**（Active Mode）：客户端指定数据端口，服务器主动连接过去。

 **被动模式**（Passive Mode）：服务器提供一个端口范围（如 21100-21110），客户端在这些端口上建立连接。

为了支持**被动模式**，我们需要：

 **EXPOSE 21100-21110**（告诉 Docker FTP 服务器会使用这些端口）。

 **在 docker-compose.yml 里映射到宿主机**：

ports:

- "2121:2121"

- "21100-21110:21100-21110"

EXPOSE vs ports:

|  |  |  |
| --- | --- | --- |
| 指令directives | 作用target | 是否映射到宿主机 |
| EXPOSE | 仅声明容器监听的端口 | ❌ **不会**，只影响容器间通信 |
| ports: (在 docker-compose.yml) | 将端口映射到宿主机 | ✅ **会**，宿主机可以访问 |