**Лабораторна робота № 9.1**

**Створення проекту для web-розробки, який складається наборів контейнерів**

**з використанням Docker Compose та Dockerfile**

**Мета роботи** – ознайомитися та набути навичок:

- написання скрипта **Dockerfile**для створення контейнеру;

- встановлення **Docker Compose в Ubuntu 18.04**;

- створення контейнерів (сервісів): **Nginx+Php-Fpm+MySQL** для розроблення web-додатку для **Magento2**;

***Теоретичні відомості***

Для створення контейнерів необхідно вміти працювати з **Docker Compose** та **Dockerfile.**

***Dockerfile* та синтаксис для їх створення**

***Dockerfile*** - скрипт, який дозволяє автоматизувати процес побудови контейнерів шляхом виконання відповідних команд (дій) в *base* образі для формування нового образу.

Усі подібні файли починаються з позначення ***FROM*** так як і процес побудови нового контейнера, далі йдуть різні методи, команди, аргументи або умови, після застосування яких створиться Docker контейнер.

Розглянемо синтаксис *Dockerfile*. В Докер файлах міститься два типи основних блоків - *коментарі та команди з аргументами*. Причому для всіх команд передбачається певний порядок.

Нижче наведено типовий приклад синтаксису, де перший рядок є коментарем, а другий - командою.

*Print «Hello from User!»*

*RUN echo «Hello from User!!»*

Розглянемо усі можливі команди. Усі команди в Докерфайлах прийнято вказувати великими літерами - наприклад **RUN**, **CMD** і т.д.

* Команда ***ADD*** - бере два аргументи, шлях звідки скопіювати файл і шлях куди скопіювати файли у власну файлову систему контейнера. Якщо ж *source* шляхом є *URL* (тобто адреса веб-сторінки) - то вся сторінка буде скачана і поміщена в контейнер.

*Синтаксис команди:* ***ADD [вихідний шлях або URL] [шлях призначення]***

*ADD /my\_ friend\_app /my\_ friend\_app*

* Команда ***CMD,*** схожа на команду RUN, використовується для виконання певних програм, але, на відміну від RUN дана команда зазвичай застосовується для запуску/ініціації додатків або команд вже після їх установки за допомогою RUN в момент побудови контейнера.

*Синтаксис команди:* ***CMD %додаток% «аргумент», «аргумент»,*** ..

*CMD «echo» «Hello from User!»*

* Команда ***ENTRYPOINT*** встановлює конкретний додаток за замовчуванням, який використовується кожний раз в момент побудови контейнера за допомогою образу. Наприклад, якщо ви встановили певний додаток всередині образу і ви збираєтеся використовувати даний образ тільки для цього додатка, ви можете вказати це за допомогою ENTRYPOINT, і кожний раз, після створення контейнера з образу, ваш додаток буде сприймати команду CMD, наприклад. Тобто не буде потреби вказувати конкретний додаток, необхідно буде тільки вказати аргументи.

*Синтаксис команди*: ***ENTRYPOINT %додаток% «аргумент»***

Врахуйте, що аргументи опційні - вони можуть бути надані командою CMD або під час створення контейнера.

***ENTRYPOINT echo***

Синтаксис команди спільно з CMD:

***CMD «Hello from World!»***

***ENTRYPOINT echo***

* Команда ***ENV*** використовується для установки змінних середовища (однієї або багатьох). Дані змінні виглядають наступним чином «*ключ = значення*» і вони доступні всередині контейнера скриптів і різних додатків. Даний функціонал Докера, по суті, дуже сильно збільшує гнучкість щодо різних сценаріїв запуску додатків.

*Синтаксис команди*: ***ENV %ключ% %значення%***

***ENV BASH /bin/bash***

* Команда ***EXPOSE*** використовується для прив'язки певного порту для реалізації мережевої зв'язності між процесом всередині контейнера і зовнішнім світом - хостом.

*Синтаксис команди*: *EXPOSE %номер\_порту%*

***EXPOSE 8080***

* Команда ***FROM*** є однією з найнеобхідніших при створенні Докерфайла. Вона визначає базовий образ для початку процесу побудови контейнера. Це може бути будь-який образ, в тому числі і створені вами до цього. Якщо вказаний вами образ не знайдений на хості, Докер спробує знайти і завантажити його. **Ця команда в Докерфайлі завжди повинна бути вказана першою**.

*Синтаксис команди:* ***FROM %назва\_образу% FROM centos***

* Команда **MAINTAINER** не є виконуваною, вона визначає значення поля автора образу. Найкраще її вказувати відразу після команди FROM.

*Синтаксис команди*: ***MAINTAINER %ваше\_ім'я%***

MAINTAINER User Networks

* Команда ***RUN*** є ***основною командою*** для виконання команд при написанні Докерфайла. Вона бере команду як аргумент і запускає її з образу. На відміну від CMD дана команда використовується для побудови образу (можна запустити кілька RUN поспіль, на відміну від CMD).

*Синтаксис команди:* ***RUN % імя\_команди%***

*RUN yum install -y wget*

* Команда ***USER*** - використовується для установки *UID* або імені користувача, яке буде використовуватися в контейнері.

*Синтаксис команди*: ***USER %ID\_користувача%***

*USER 751*

* Команда ***VOLUME*** використовується для організації доступу вашого контейнера до директорії на хості (те ж саме, що і монтування директорії)

*Синтаксис команди:* ***VOLUME [«/ dir\_1», «/ dir2» ...]***

VOLUME [«/home»]

* Команда ***WORKDIR*** вказує директорію, з якої буде виконуватися команда *CMD*.

*Синтаксис команди*: ***WORKDIR /шлях***

WORKDIR ~/

***Приклад створення свого власного образу для встановлення Mongodb***

*MongoDB* - найбільш популярна нереляційна база даних.

Створимо порожній файл і відкриємо його за допомогою редактора *nano*:

*nano Dockerfile*

Надалі ми можемо вказати коментарями для чого даний *Dockerfile* буде використовуватися (це не обов'язково), але може бути корисно в подальшому. Про всяк випадок нагадаю - всі коментарі починаються з символу #.

########

# Dockerfile to build MongoDB container images

# Based on Ubuntu

########

Далі, вкажемо базовий образ:

***FROM ubuntu***

Після чого оновимо репозиторії та встановимо *gnupg2 (*вільна програма для шифрування інформації і створення електронних цифрових підписів*)*:

***RUN apt-get update && apt-get install -y gnupg2***

Після вкажемо команди і аргументи для скачування *MongoDB* (установку проводимо відповідно до плану на офіційному сайті):

***RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10***

***RUN echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' > tee /etc/apt/sources.list.d/mongodb.list***

***RUN apt-get update***

***RUN apt-get install -y mongodb***

***RUN mkdir -p /data/db***

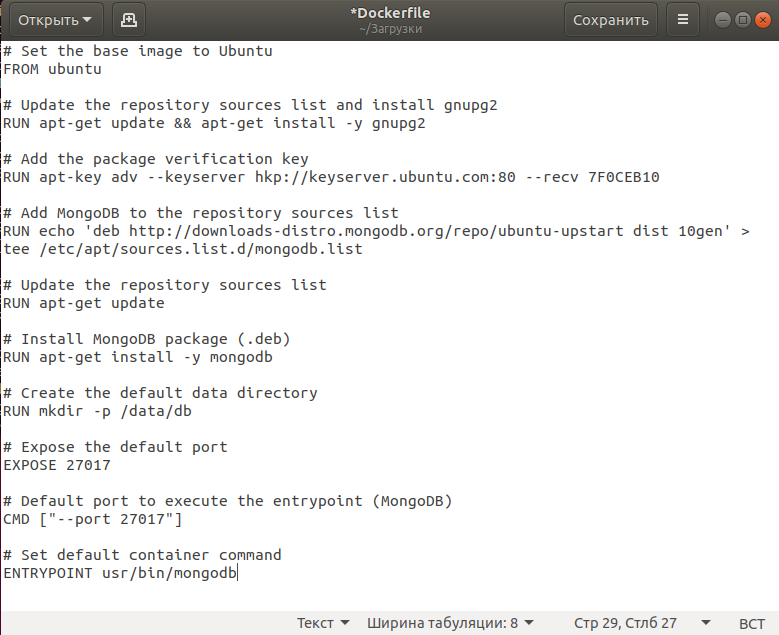
Після чого вкажемо дефолтний порт для MongoDB:

***EXPOSE 27017***

***CMD [«--port 27017»]***

***ENTRYPOINT usr/bin/mongod***

Ось як повинен виглядати у вас фінальний файл – перевірте, а потім можна зберегти зміни і закрити файл:



*#########*

*#* ***Dockerfile to build MongoDB container images***

***# Based on Ubuntu***

*#########*

*# Set the base image to Ubuntu*

*FROM ubuntu*

*# Update the repository sources list and install gnupg2*

*RUN apt-get update && apt-get install -y gnupg2*

*# Add the package verification key*

*RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10*

*# Add MongoDB to the repository sources list*

*RUN echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' > tee /etc/apt/sources.list.d/mongodb.list*

*# Update the repository sources list*

*RUN apt-get update*

*# Install MongoDB package (.deb)*

*RUN apt-get install -y mongodb*

*# Create the default data directory*

*RUN mkdir -p /data/db*

*# Expose the default port*

*EXPOSE 27017*

*# Default port to execute the entrypoint (MongoDB)*

*CMD ["--port 27017"]*

*# Set default container command*

*ENTRYPOINT usr/bin/mongodb*

***Запуск контейнера Docker***

Створити наш перший *MongoDB* образ за допомогою Docker!

***sudo docker build -t user\_mongodb .***

***-t*** та ім'я тут використовується для присвоювання тега образу.

А *точка в кінці* означає що Докерфайл знаходиться в тому ж поточному робочому каталозі, в якому виконується команда.

Для виведення всіх можливих ключів введіть ***sudo docker build -help***

*Запускаємо наш новий MongoDB* в контейнері!

***sudo docker run -name UserMongoDB -t -i user\_mongodb***

Ключ ***-name*** використовується для *присвоєння простого імені контейнеру*, в іншому випадку це буде досить довга цифро-буквена комбінація. Після запуску контейнера для того, щоб повернутися в систему хоста натисніть ***CTRL+P,*** а потім ***CTRL+Q***.

**Установка Docker Compose в Ubuntu 18.04**

*Docker* - це інструмент для автоматизації розгортання додатків *Linux* всередині контейнерів програмного забезпечення, але для використання всіх його можливостей необхідно, щоб кожний компонент додатка запускався у своєму власному контейнері. Для великих програм з великою кількістю компонентів, організація спільних - запуску, комунікації та зупинки всіх контейнерів може швидко стати дуже непростим і заплутаним завданням.

Спільнота *Docker* запропонувало популярне рішення, яке називається ***Fig*** і дозволяє вам використовувати єдиний файл з розширенням .***YAML або .YML*** для організації спільної роботи всіх ваших контейнерів і конфігурацій. Воно стало настільки популярним, що команда *Docker* вирішила створити *Docker* *Compose* на базі вихідного коду *Fig,* який в даний є застарілим інструментом і не підтримується.

*Docker Compose* спрощує організацію процесів контейнерів *Docker*, включаючи запуск, зупинку і настройку зв'язків і томів всередині контейнера. Це утиліта, яка полегшує збірку і запуск системи, що складається з декількох контейнерів, пов'язаних між собою.

**1.** ***Встановимо останню версію Docker Compose для управління додатками з декількома контейнерами.***

Можна встановити *Docker Compose* з офіційних репозиторіїв *Ubuntu*, але там не представлені найостанніші версії, тому ми будемо встановлювати *Docker Compose* зі сховищ *Docker на GitHub*. Команда нижче трохи відрізняється від команди, яку ви знайдете на сторінці *Releases*. Завдяки використанню прапорця «***-o»*** для вказівки файлу виведення замість перенаправлення виведення, цей синтаксис дозволяє уникнути помилки відсутності прав доступу, що виникає при використанні *sudo.*

Перевіряємо поточну версію, за необхідності оновимо її за допомогою наступної команди (***curl - утиліта доступу до сервісу)***:

***sudo curl -L "https://github.com/docker/compose/releases/download/1.25.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose***

Встановимо дозвіл (+x – на виконання / rwx), тобто зробимо файл виконуваним:

**sudo chmod +x /usr/local/bin/docker-compose**

Перевіримо чи установка пройшла успішно за допомогою перевірки версії:

**docker-compose --version**

В результаті повинна бути виведена встановлена нами версія:

*Output*

*docker-compose version 1.25.5, build 8a1c60f6*

Після встановлення Docker Compose можемо запустити приклад «Hello World».

**2). *Запуск контейнера за допомогою Docker Compose***

У загальнодоступному реєстрі *Docker*, *Docker Hu*b, міститься образ *Hello World*, який використовується для демонстрації та тестування. Він демонструє мінімальні параметри конфігурації, необхідні для запуску контейнера за допомогою *Docker Compose*: файл *YAML*, що викликає окремий образ:

Створимо директорію для файлу *YAML* і перейдемо в неї:

***mkdir hello-world***

***cd hello-world***

Створимо в цій директорії файл *YAML:*

***nano docker-compose.yml***

Помістіть у файл наступні дані, збережіть його і закрийте текстовий редактор:

***docker-compose.yml***

***my-test:***

***image: hello-world***

Перший рядок файлу *YAML* використовується в якості частини імені контейнера. Другий рядок вказує, який образ використовується для створення контейнера.

При запуску команди ***docker-compose up*** вона буде шукати локальний образ за вказаним іменем, тобто *hello-world*. Після цього можна зберегти і закрити файл.

Ми можемо вручну переглянути образи в нашій системі за допомогою команди *docker images:* ***docker images***

Коли локальні образи відсутні, будуть відображені тільки заголовки стовпців:

*Output*

*REPOSITORY TAG IMAGE ID CREATED SIZE*

Далі, перебуваючи в директорії *«~/hello-world*», виконаємо наступну команду:

***docker-compose up***

При першому запуску команди, якщо локальний образ з ім'ям *hello-world* відсутній, *Docker Compose* буде завантажувати його з відкритого сховища *Docker Hub*:

*Output*

*Pulling my-test (hello-world: latest) ...*

*latest: Pulling from library / hello-world*

*c04b14da8d14: Downloading [============================================== ====>] c04b14da8d14: Extracting [======================================== ==========>] c04b14da8d14: Extracting [================================== ================>] c04b14da8d14: Pull complete*

*Digest: sha256: 0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9*

*Status: Downloaded newer image for hello-world: latest*

*. . .*

Після завантаження образу *docker-compose* створює контейнер, поміщає в нього і запускає програму *hello*, що, в свою чергу, підтверджує, що установка, виконана успішно:

*Output*

*. . .*

*Creating helloworld\_my-test\_1...*

*Attaching to helloworld\_my-test\_1*

*my-test\_1 |*

*my-test\_1 | Hello from Docker.*

*my-test\_1 | This message shows that your installation appears to be working correctly.*

*my-test\_1 |*

*. . .*

Далі програма відображає пояснення того, що вона зробила:

*Output of docker-compose up*

*1. The Docker client contacted the Docker daemon.*

*2. The Docker daemon pulled the "hello-world" image from the Docker Hub.*

*3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.*

*4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.*

Контейнери *Docker* продовжують працювати, поки команда залишається активною, тому після завершення роботи *hello* контейнер зупиняється. Отже, коли ми переглядаємо активні процеси, заголовки стовпців будуть з'являтися, але контейнер hello-world НЕ буде з'являтися в списку, оскільки він не запущений.

***docker ps***

*Output*

*CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES*

Переглянемо інформацію контейнера, яка нам буде потрібна на наступному кроці, використовуючи ключ «***-a»***, за допомогою якого можна відобразити всі контейнери, а не тільки активні: ***docker ps -a***

*Output*

*CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES*

*06069fd5ca23 hello-world "/hello" 35 minutes ago Exited (0) 35 minutes ago*

*PORTS NAMES*

*drunk\_payne*

Можемо отримати інформацію, яка нам буде потрібна для видалення контейнера, коли ми закінчимо працювати з ним.

**3). *Видалення способу (необов'язково)***

Щоб уникнути необов'язкового використання дискового простору, ми видалимо локальний образ. Для цього нам треба видалити всі контейнери, які містять образ, за допомогою команди ***docker rm***, після якої слідує CONTAINER ID або NAME. Нижче ми використовуємо CONTAINER ID з команди ***docker ps -a***, яку ми тільки що запустили. Не забувайте замінювати ідентифікатор на ідентифікатор вашого контейнера:

***docker rm 06069fd5ca23***

Після видалення всіх контейнерів, які містять образ, ми можемо видалити образ:

***docker rmi hello-world***

***Створення наборів контейнерів для web-розробки***

***з використанням Docker Compose***

**Завдання:**

Створити три контейнери для розроблення web-додатків:

* веб-сервера та поштового проксі-сервера ***Nginx,*** який працює на UNIX-подібних операційних системах, з мінімальним налаштуванням для запуску проекту,
* сервера бази даних ***MySQL***,
* мови програмування для розробки web-додатків ***PHP 7.1.3-fpm***.

В якості проекту будемо розгортати - беремо ***Magento2 2.5*** (система управління інтернет-магазинами).

В Linux основним інструментом для управління службами на сервері є команда ***systemctl.*** Для автоматичного включення служби використовуємо параметр ***enable***  (для зупинки служби – ***disable***). Вмикаємо і запускаємо сервіс:

***systemctl enable docker.service***

***systemctl start docker.service***

Для створення однією командою нашої структури необхідно оновити *docker-compose.* Встановимо необхідні для нього компоненти:

***sudo apt install*** ***epel-release***

***sudo apt install -y python-pip***

***sudo apt-get upgrade python***

Створимо для цього проекту папку *mage* і перейдемо до неї:

*mkdir /mage*

*cd /mage*

Створюємо наступну структуру папок:

*mkdir MySQL Nginx PHP*

В папці *MySQL* будуть зберігатися бази. Зручно створювати резервні копії (*backup copy*) баз та їх переносити.

В папці *Nginx* будуть зберігатися лог-файли, файл конфігурації і наш проект.

В папкуPHP будемо складати *Dockerfile* з налаштуваннями і ***php.ini.***

В корені (це папка */mage*) буде лежати файл *docker-compose.yml.*

Створюємо конфігураційний файл для *Nginx:*

*cd /mage/Nginx/core*

*touch nginx.conf*

*nano nginx.conf*

Можна використовувати будь-який інший редактор (наприклад, *mc*). Якщо його немає - можна встановити за допомогою:

***sudo*** ***apt install nano***

І додаємо до конфігураційного файлу *nginx.conf* наступний код:

server {

listen 80;

index index.php index.html index.htm;

server\_name magento2.dev;

set $MAGE\_ROOT /var/www/magento2;

error\_log /var/log/nginx/error.log;

access\_log /var/log/nginx/access.log;

root $MAGE\_ROOT;

location ~\*\.php$ {

try\_files $uri $uri/ /index.php last;

fastcgi\_split\_path\_info (.+?\.php)(/.\*)$;

fastcgi\_pass php:9000;

fastcgi\_index index.php;

include fastcgi\_params;

fastcgi\_param SCRIPT\_FILENAME

$document\_root$fastcgi\_script\_name;

fastcgi\_param PATH\_INFO $fastcgi\_path\_info;

}

location ~\*.php/ {rewrite (.\*.php)/$1 last; }

}

Це мінімальна конфігурація для того, щоб все запрацювало.

У першому блоці описуємо який порт буде слухати, перелічуємо можливі *index* сторінки, називаємо та створюємо *alias* (псевдонім) для довгого шляху, де лежить *magento2*, пишемо – які потрібні логи і вказуємо де вони обов'язково повинні зберігаються, вказуємо папку там, де знаходиться *magento2* (у даному випадку наш *alias* $ MAGE\_ROOT) .

У другому блоці прописуємо параметри *fastcgi*.

Третій блок потрібен для вирішення проблеми відображення, в проекті з’явилася пуста сторінка. З документації написано, що *magento2* вимагає реврайтинга (rewriting - обробка початкових текстових матеріалів з метою їх подальшого використання). (В інших проектах таких проблем не виникало).

В папці ***www*** створюємо каталог для нашого проекту:

***cd /mage/Nginx/www***

***mkdir magento2***

Скачуємо з офійційного сайту *magento2* <https://magento.com/tech-resources/download>

Та витягуємо з архіву у папку ***/mage/Nginx/www/magento2***

*З налаштуваннями для Nginx ми закончили*.

*Переходимо до PHP:*

Починаємо з *Dockerfile*

***cd /mage/PHP***

***touch Dockerfile php.ini***

***nano Dockerfile***

*Збираємо самостійно:*

*FROM php:7.1.3-fpm*

*RUN apt-get update && apt-get install -y \*

*curl \*

*wget \*

*git \*

*libfreetype6-dev \*

*libjpeg62-turbo-dev \*

*libxslt-dev \*

*libicu-dev \*

*libmcrypt-dev \*

*libpng-dev \*

*libxml2-dev \*

*&& docker-php-ext-install -j$(nproc) iconv mcrypt mbstring mysqli pdo\_mysql zip \*

*&& docker-php-ext-configure gd --with-freetype-dir=/usr/include/ --with-jpeg-dir=/usr/include/ \*

*&& docker-php-ext-install -j$(nproc) gd*

*RUN docker-php-ext-configure intl*

*RUN docker-php-ext-install intl*

*RUN docker-php-ext-install xsl*

*RUN docker-php-ext-install soap*

*RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer*

*ADD php.ini /usr/local/etc/php/conf.d/40-custom.ini*

*WORKDIR /var/www/magento2*

*CMD ["php-fpm"]*

Налаштуємо *php.ini: nano php.ini*

*memory\_limit = 2G*

*always\_populate\_raw\_post\_data = -1*

*cgi.fix\_pathinfo = 1*

*fastcgi\_split\_path\_info = 1*

*max\_execution\_time = 18000*

*flag session.auto\_start = off*

*zlib.output\_compression = on*

*suhosin.session.cryptua = off*

*display\_errors = Off*

Налаштування PHP виконано.

Далі створюємо файл *docker-compose*, який нам усі складові збереже в одній зручній системі: ***cd /mage***

***touch docker-compose.yml***

***nano docker-compose.yml***

Розпишемо які сервіси і з якими налаштуваннями повинні запуститися:

# Пропишемо версію

*version: '3.3'*

# Перелічимо сервіси

***services:***

***nginx:***

# Пропишимо який образ ми хочемо використати

*image: nginx:latest*

# Дамо назву контейнеру

***container\_name: nginx***

# Перекидання портів

*ports:*

*- "80:80"*

*- "443: 443"*

# Перекидання папок

*volumes:*

*- ./Nginx/core:/etc/nginx/conf.d*

*- ./Nginx/www:/var/www/*

*- ./Nginx/Logs:/var/log/nginx/*

*- ./Nginx/html:/usr/share/nginx/html/*

# Вкажемо залежності

*links:*

*- php*

***mysql:***

*image: mysql:latest*

*ports:*

*- "3306: 3306"*

# Дамо назву контейнеру

***container\_name: mysql***

# Пропишемо налаштування, замість *mypassword* пропонується прописати більш складний пароль, який належить *root*

*environment*:

*- MYSQL\_ROOT\_PASSWORD=mypassword*

*- MYSQL\_DATABASE=magento2*

*- MYSQL\_USER=magento2*

*- MYSQL\_PASSWORD=magento2*

*volumes:*

*- ./MySQL:/var/lib/mysql*

***php:***

# Будуємо з підтримкою *dockerfile,* вказавши директорію, де він знаходиться

*build: ./PHP*

*container\_name: php-fpm*

*volumes*:

*- ./Nginx/www:/var/www*

*links*:

*- mysql*

*phpmyadmin*:

*image: phpmyadmin/phpmyadmin*

*container\_name: phpmyadmin*

*ports:*

*- 8090: 80*

*links:*

*- mysql:db*

На екрані з’являться рядки по ходу встановлення.

Після встановлення в папці *MySQL* створяться багато файлів і папок, з яких буде *Magento2*, а в папці Nginx/Logs з'являється 2 логи.

Відкривши браузер і набравши в ньому *localhost*, ви обов'язково побачите запрошення до установки *Magento2.*

Якщо щось у вас не вийшло, пропонується список рішень для усунення проблем:

1. Версія *docker-compose*- файлу не підійшла, тож потрібно поправити «версію: '3.3'», яка саме версія вам потрібна, перегляньте за посиланням:

<https://docs.docker.com/compose/compose-file/>

1. Все нормально запустилось, але браузер відкриває чисту сторінку, без єдиної помилки - допоможе рядок в *nginx.conf:*

*"location ~\* .php/ { rewrite (.\*.php)/ $1 last; }"*

1. Якщо після встановлення самої *Magento2* (у браузері) у вас не прорисовуються фрейми та все виглядає як текстовий варіант сайту, вам потрібно зробити наступне:

3.1. Зайти в *SQL* через *phpmyadmin localhost:8090,* логін *root,* пароль *mypassword,* вибрати базу *magento2* і ввести *sql* запит:

*insert into core\_config\_data (config\_id, scope, scope\_id, path, value) values (null, 'default', 0, 'dev/static/sign', 0)*

3.2. Підключитися до контейнеру c *PHP (php-fpm)* і набрати:

*php bin/magento cache:clean config*

*php bin/magento setup:static-content:deploy*

Цей контейнер повинен перечитати і все перевірити. Після цього все повинно коректно відображатися.

**Підготувати звіт**

1. Описати хід виконання поставлених завдань, надаючи покроковий знімок екрану (*screenshot*).
2. Висновки по роботі.

**Контрольні питання**

1. Що таке Docker Compose?
2. Що таке Dockerfile?
3. Які вам відомі команди для роботи з Dockerfile?
4. У чому полягає алгоритм створення проекту для розроблення web-застосування?

**Література**

1. Моэт Э. Использование Docker. Москва : ДМК Пресс, 2017. 354 с.
2. Сейерс Э. Х., Милл А. Docker на практике. Москва : ДМК. 2019. 516 с.
3. Парминдер Сингх Кочер. Микросервисы и контейнеры Docker. Москва : ДМК Пресс, 2019. 240 с.
4. Сайфан Джиджи. Осваиваем Kubernetes. Оркестрация контейнерных архитектур. Санкт-Петербург : Питер, 2016. 522 с.

1. <https://dker.ru/docs/> - Docker документація російською