Міністерство освіти й науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

ЗВІТ

Про виконання лабораторної роботи №9

З дисципліни «Архітектура системного програмного забезпечення»

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

**з використанням Docker Compose та Dockerfile**

TI-92 Черноусова Дениса

Перевірила д.т.н. Левченко Л.О.

Київ - 2021

**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**

***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 ~/

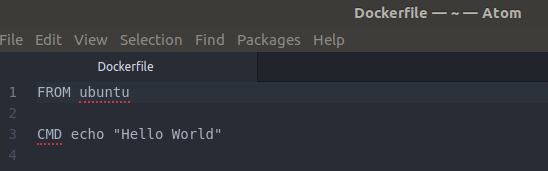
**Завдання :**

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

**Хід виконання роботи**

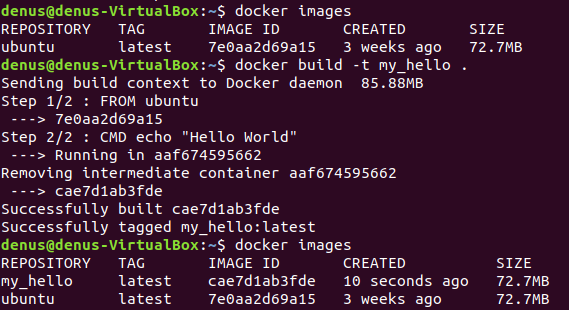


Створюємо Dockerfile за допомогою тектового редактора atom

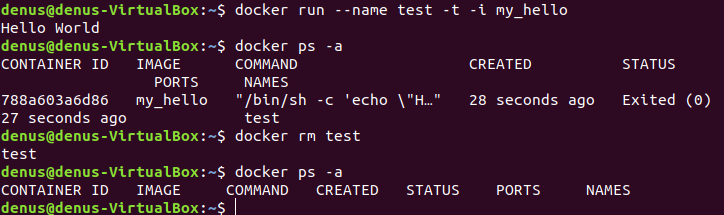


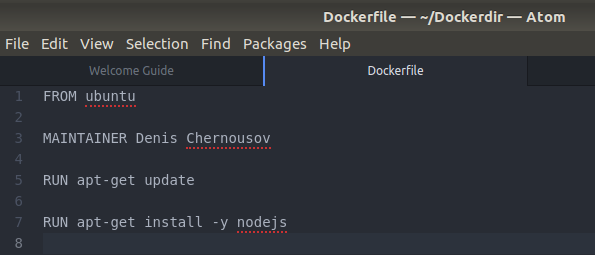
За основу контейнера беремо образ убунту

CMD – команда відбувається уже в контейнері

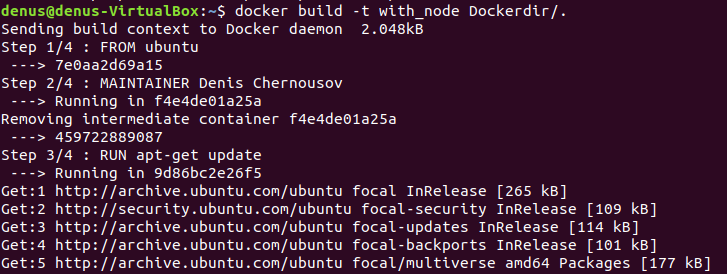


Створюємо новий образ з тегом my\_hello

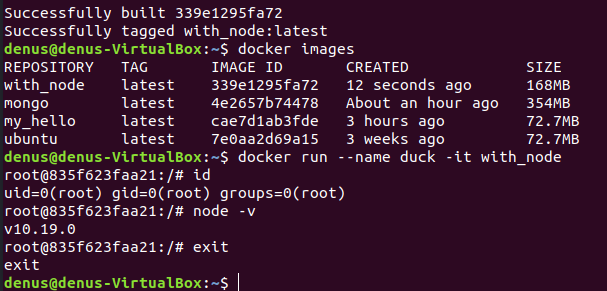
Створюємо, запускаємо контейнер в якому відбувається команда echo

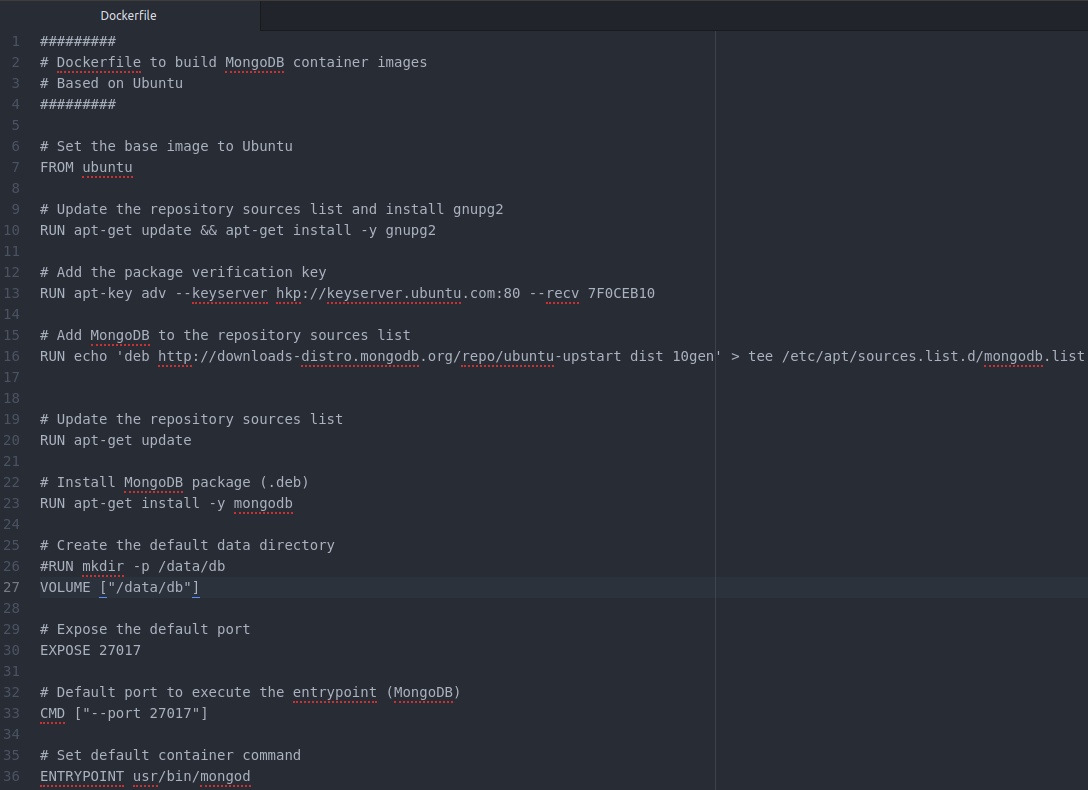


Створюємо новий докер файл з базовим образом убунту ,в якого вмонтований nodejs



RUN – спрацьовує під час створення образу

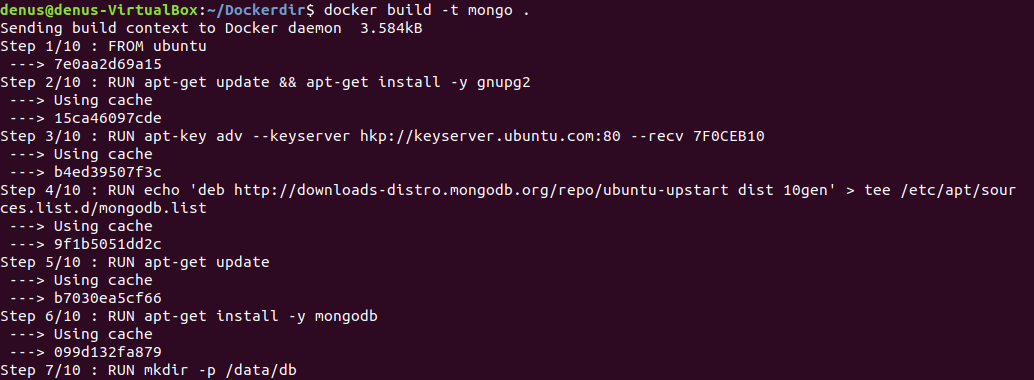


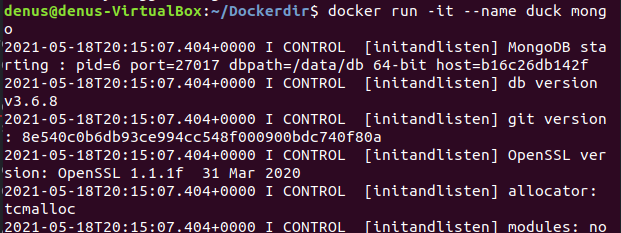
Nodejs присутній в контейнері

Створюємо Dockerfile. Контейнер убунту в якому присутній mongodb

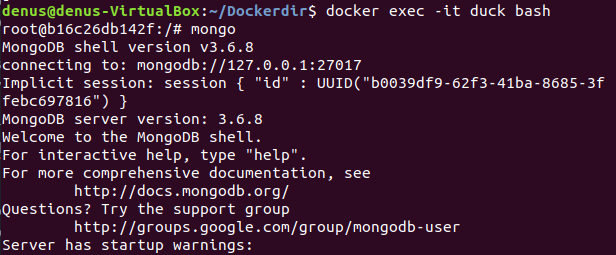
**ВАЖЛИВО** В Dockerfile була помилка в рядку “ENTRYPOINT usr/bin/mongodb”.

Я зайшов у контейнер і знайшов “usr/bin/mongod”, тож виправив так

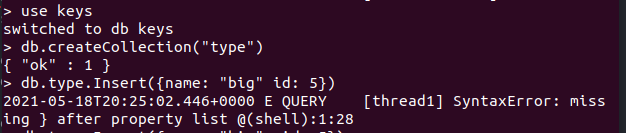
Його збудування

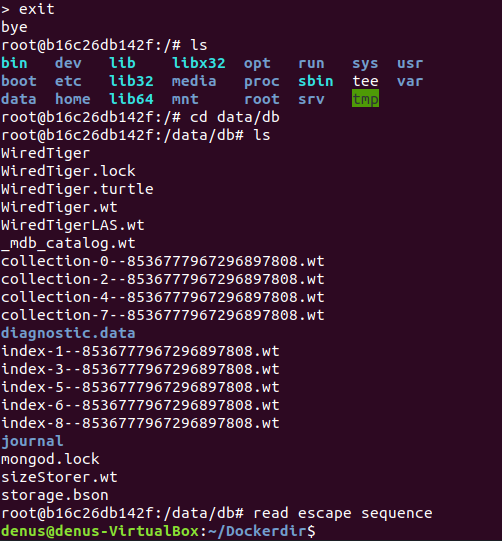


Запуск контейнера. Виходимо з нього не зупиняючи його Ctrl+P, Ctrl+Q

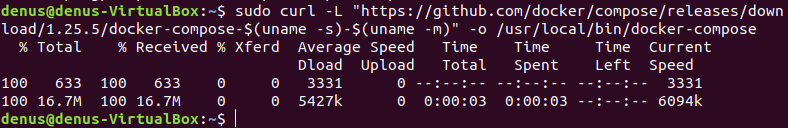


Заходимо в контейнер через bash і запускаємо mongodb

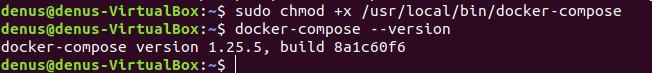
Mongodb працює



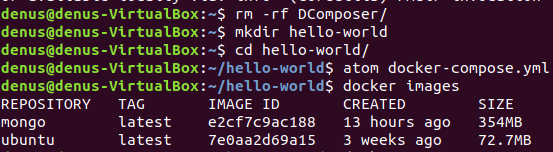
Через exit виходимо з mongo і надалі вивчаємо корінь ubuntu, а саме дивимся що в диреторії data/db знаходяться дані про бази даних створених через mongodb



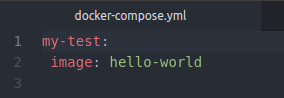
Скачуємо нову версію docker-compose

Робимо файл docker-compse виконуваним

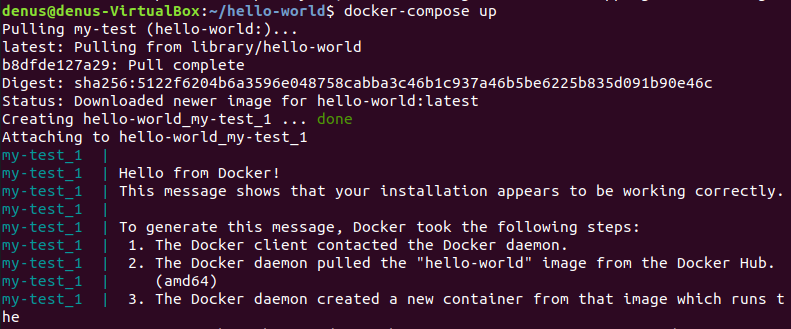
І виводимо версію компосера



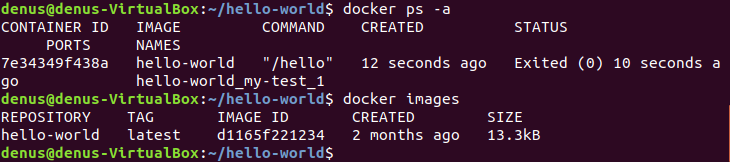
Створюємо в новій папці docker-compose.yml

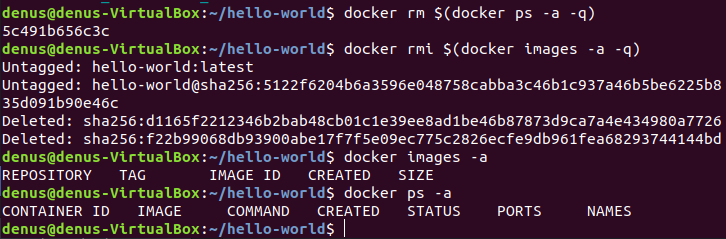


Прописуємо ім’я сервісу та образ який ми використовуємо

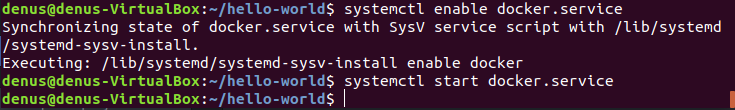


Компонуємо сервіси через файл docker-compose.yml

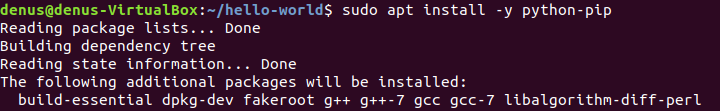
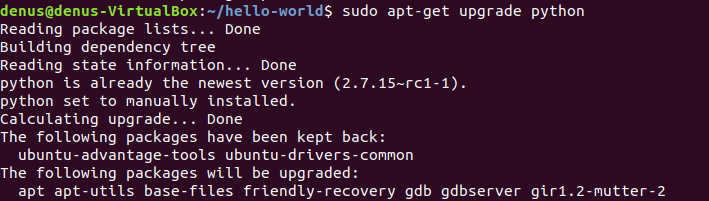
Контейнер та його образ



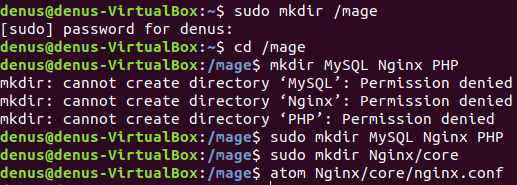
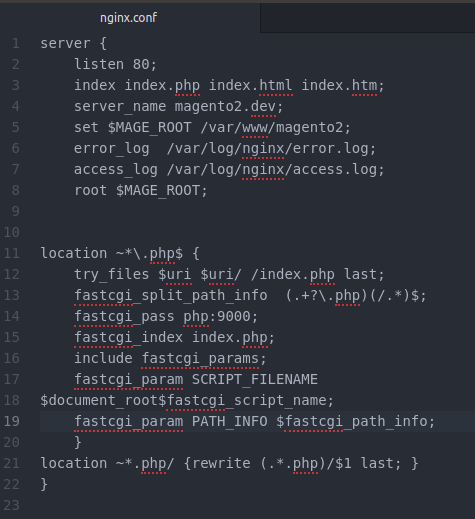
Команди для видалення усіх образів та контейнерів



Вмикаємо мережевий сервіс docker.service



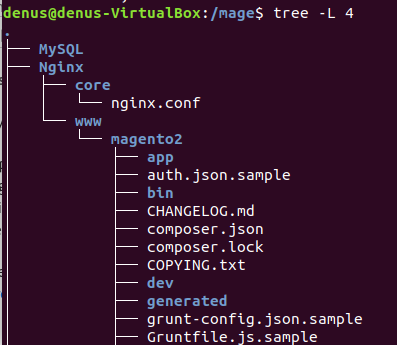
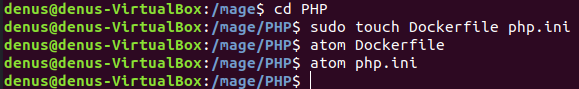
Встановлюємо необхідний компонент

створення необхідних папок та nginx.conf

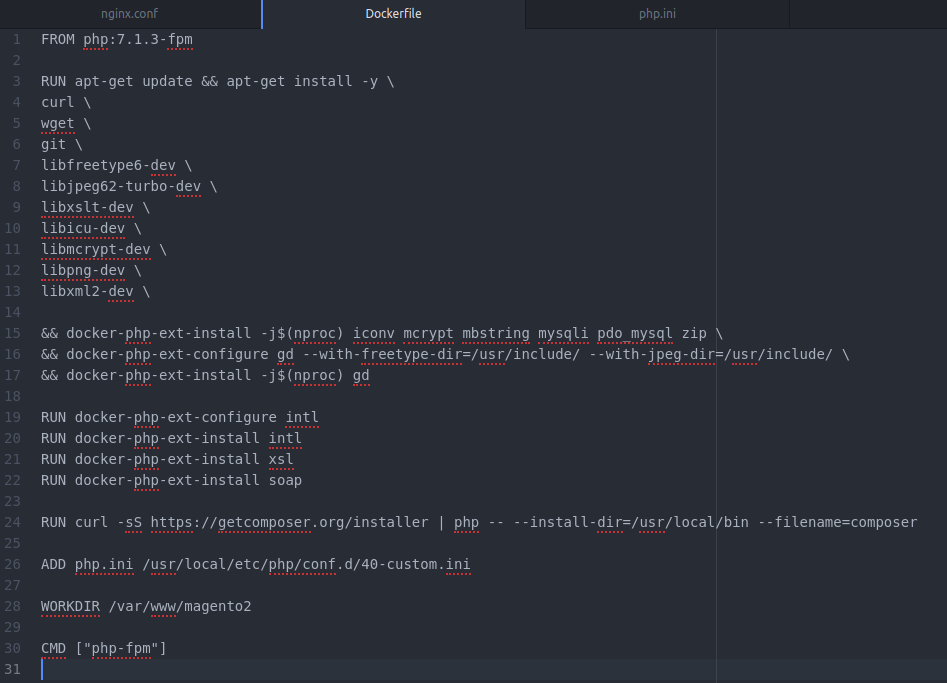
Конфігурація серверу на nginx



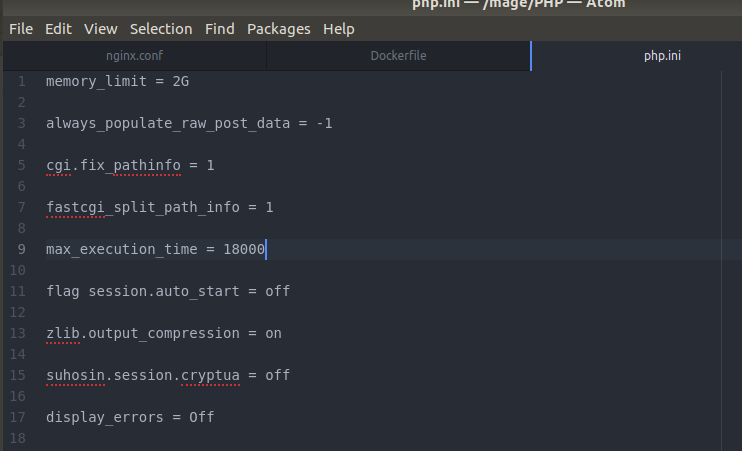
Створення папки серверу і розпаковка туди файлів фреймворка magento2

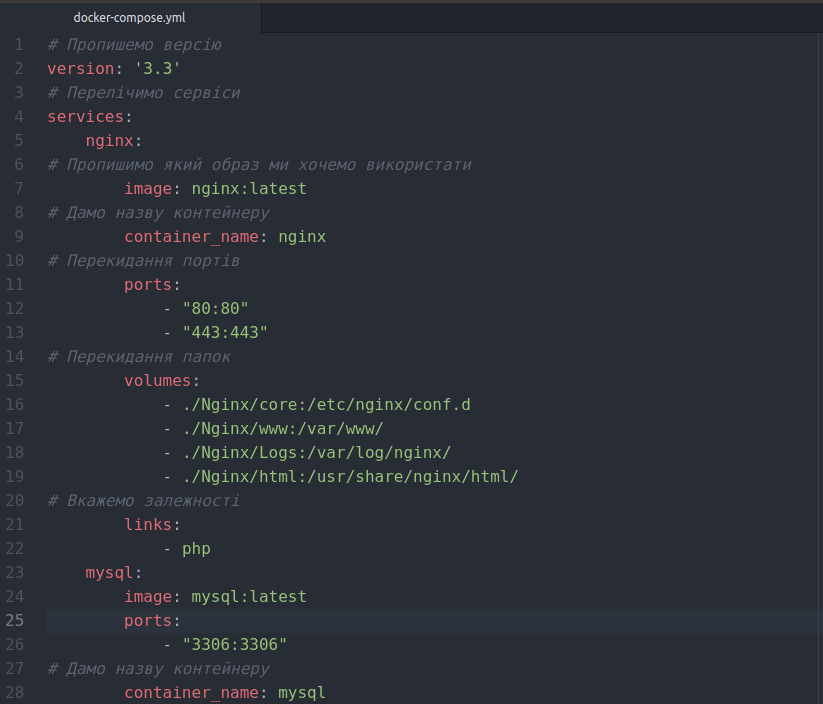
Архів успішно розпакований в корінь сайту

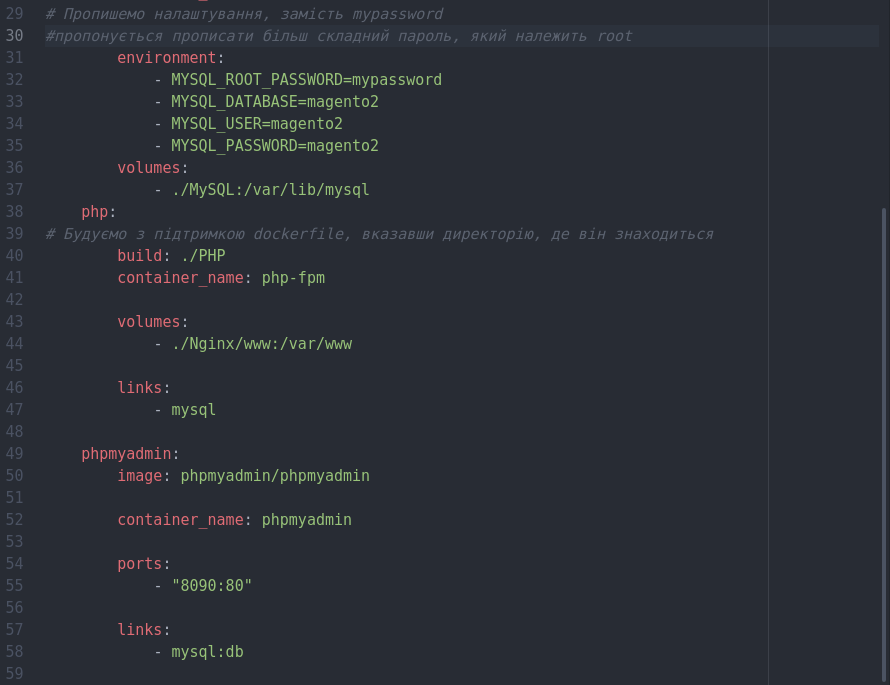
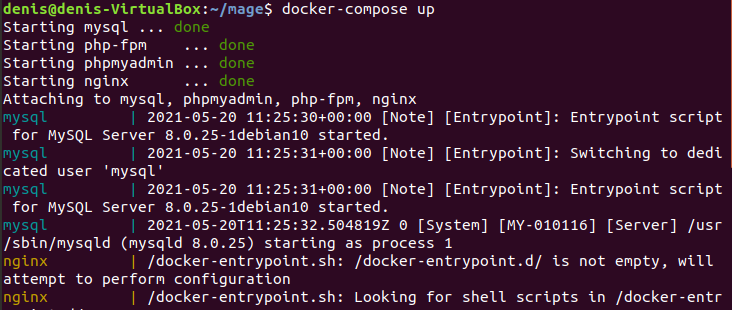
Створення Докерфайлу, через який буде створений контейнер з php-fpm



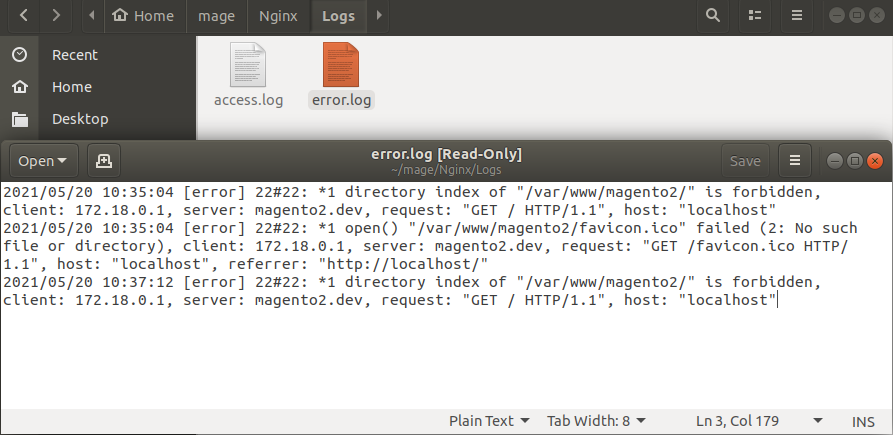
Опис образу php-fpm що буде збудований через Docker-compose

Налаштовуємо зміні середовища, які могли бути зразу прописаними в *Dockerfile через команду ENV, але прийшлось перероблювати команду ADD що копіює з хосту файл або посилання в контейнер за заданим адресом*



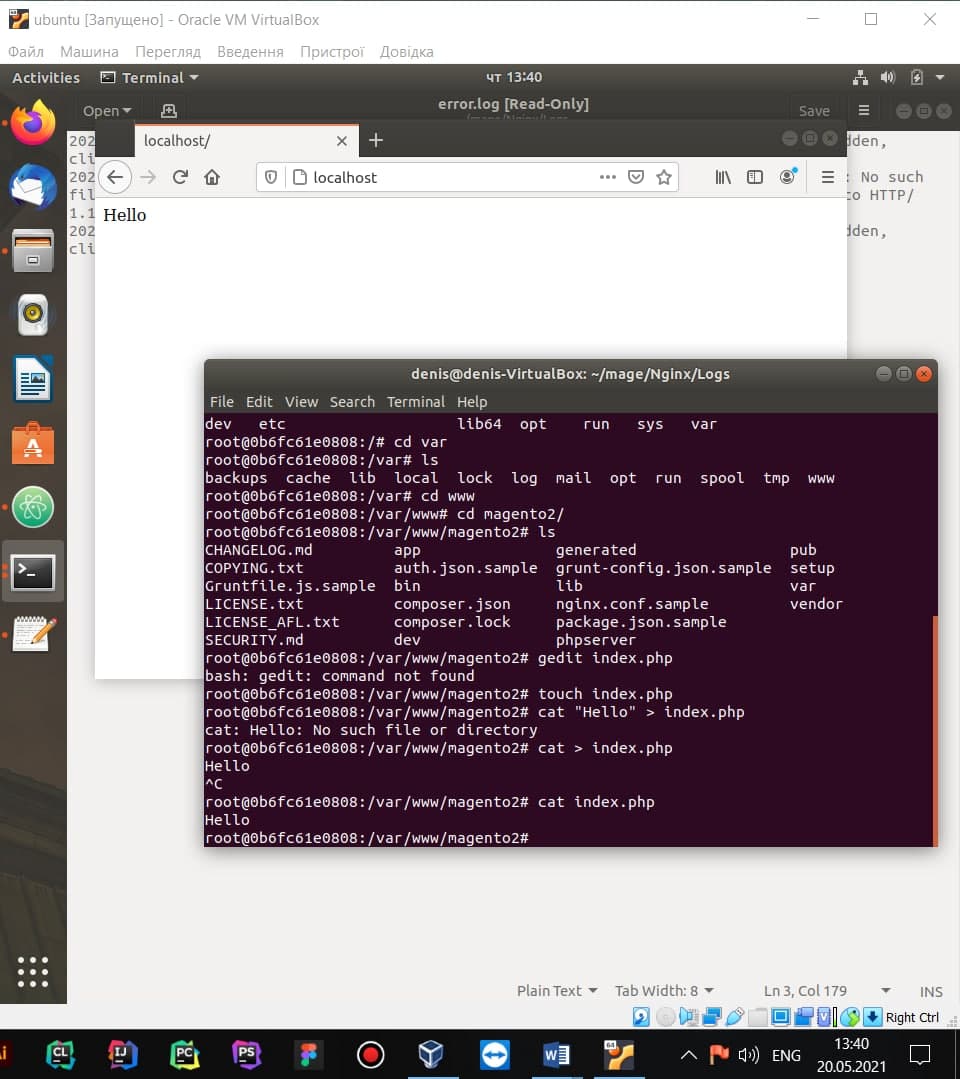
Так виглядає Docker-compose.yml – файл, що поєднує сервіси, передає кожному контейнеру(сервісу) папки з яким він буде працювати, інформацію через порти і вказує залежності від інших сервісів

Запускаємо сервер

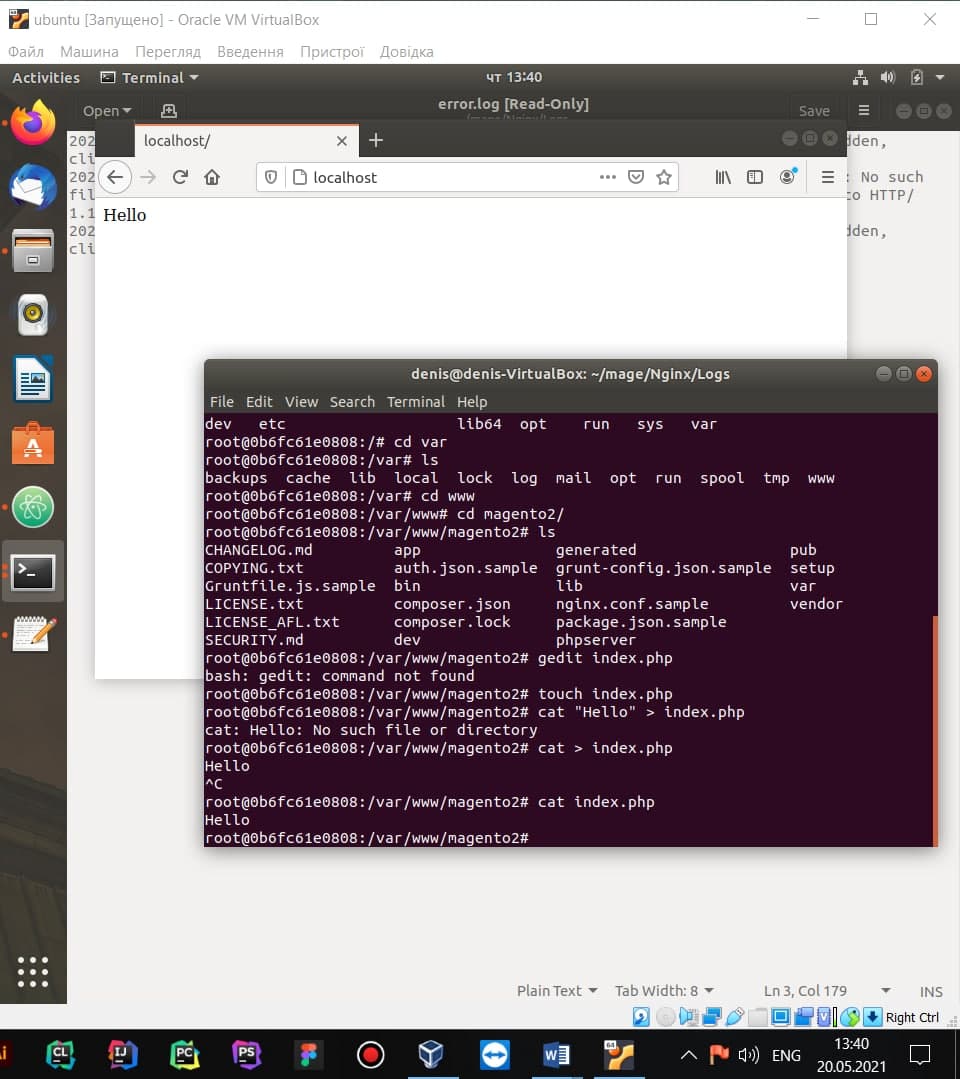
При заходжені на сервер через браузер “localhost/” показує помилка, бо система не може знайти файл index.php в певній директорії



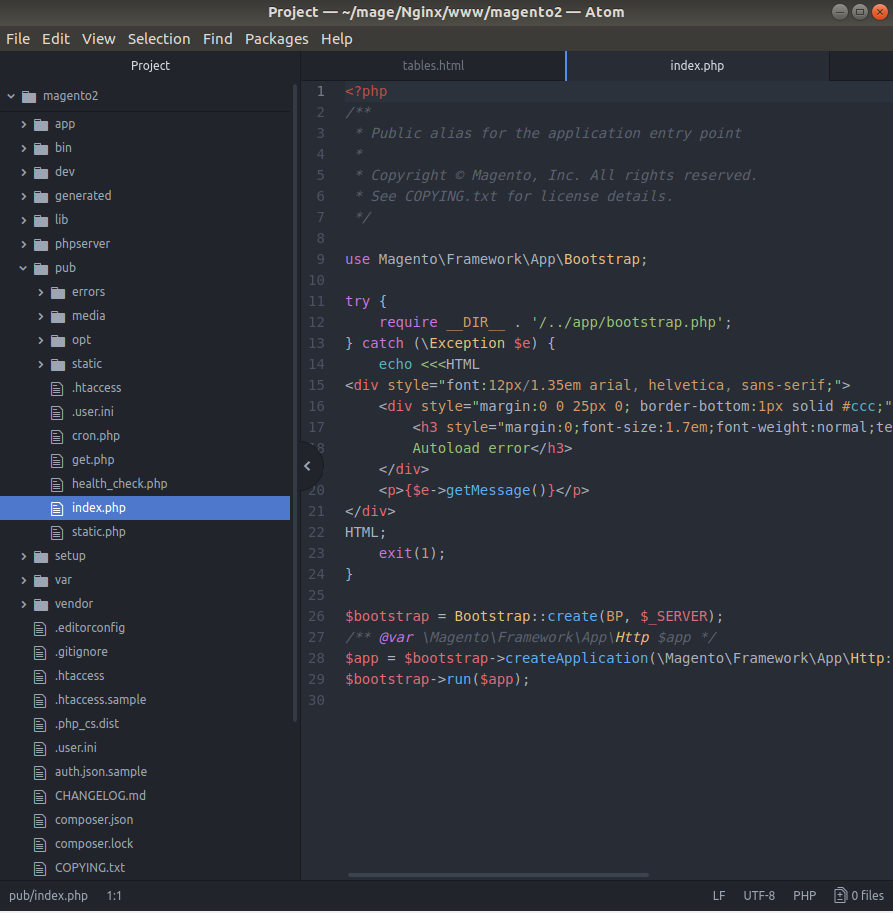
Заходимо через bash в контейнер серверу nginx



В каталозі /var/www/magento2 створюємо index.php і вносимо туди рядок “Hello”

Сайт

**Важливо!** Можна було не входити в контейнер nginx, а створити на хості в корені сайту той самий файл index.php. Дані би перенеслися в контейнер nginx автоматично через ***перекидання папок*** (див. docker-compose.yml)

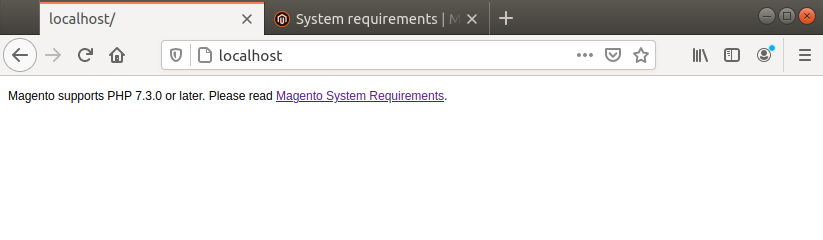


Відкриваємо наш сайт-проект по адресу mage/Nginx/www

Оскільки, у фреймворках зазвичай вхідний файл знаходиться в каталозі public (тут pub), заходимо туди і знаходимо вхідний файл index.php

В коментарі вказано, що цеentrypoint – вхідна точка

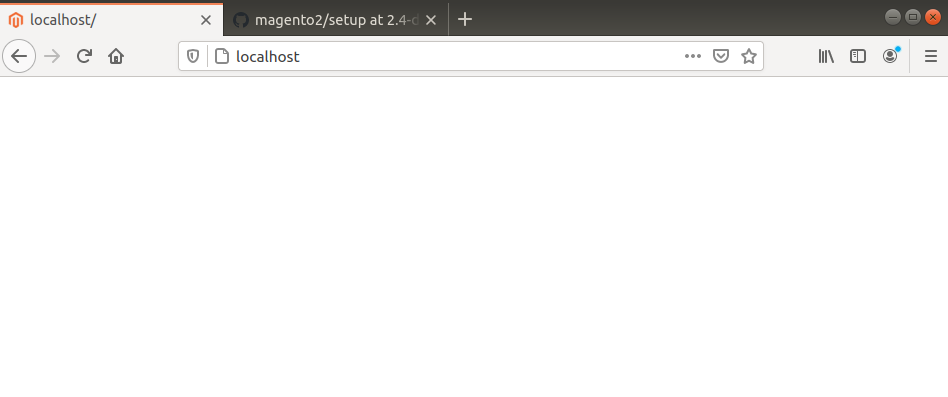
Налагоджуємо конфігурацію сервера, так щоб корінь сайту находився в каталозі pub



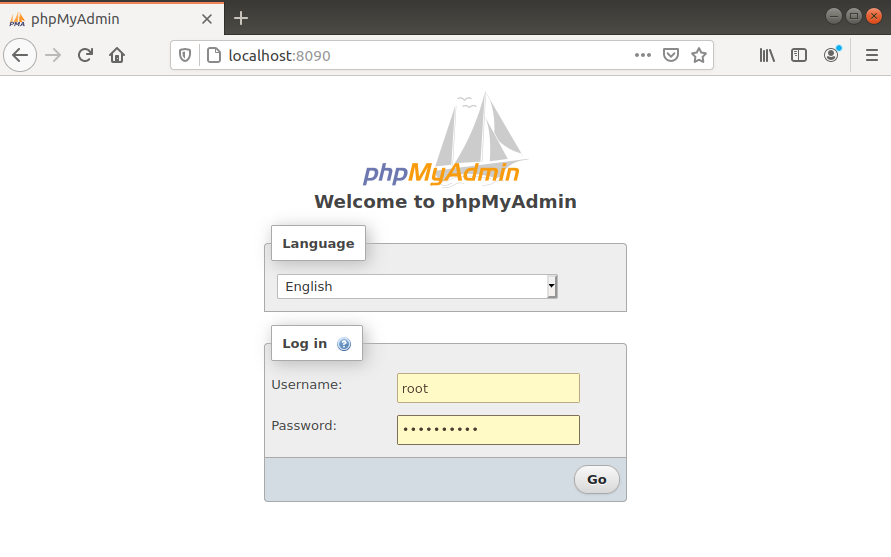
Magento2 потребує вищої версії PHP-fpm

Беремо новий образ PHP-fpm

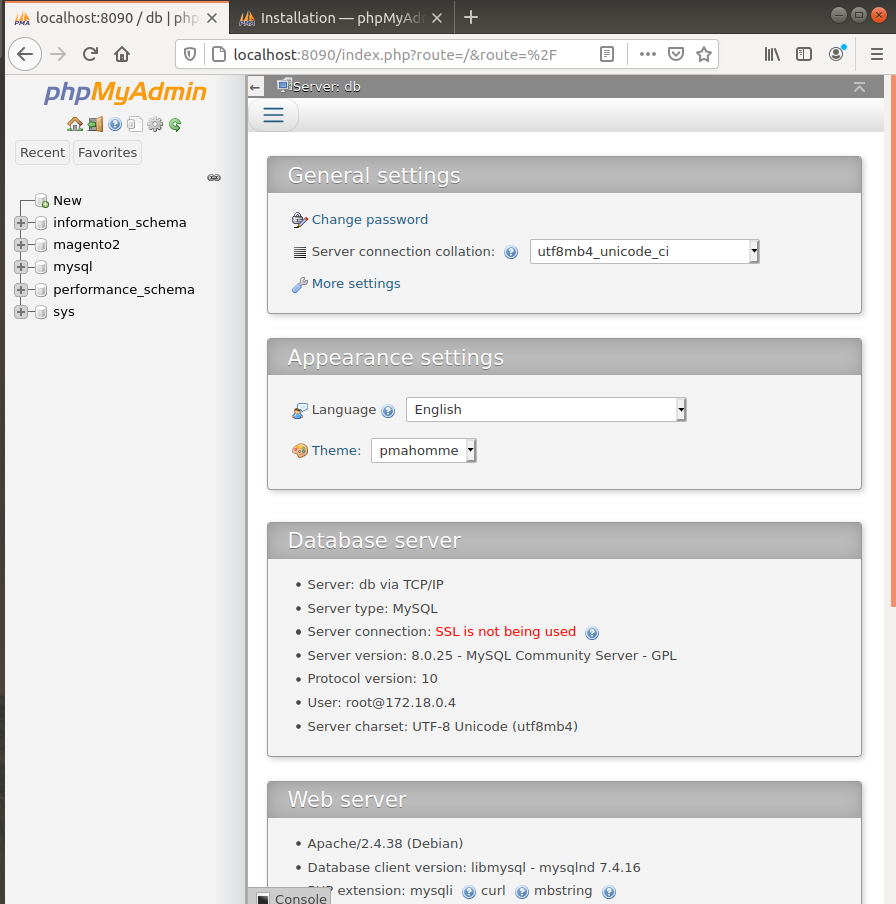
Методом виключення і багатократної компіляції знаходимо ті пакети, що потребує ця версія. Стара версія за коментована



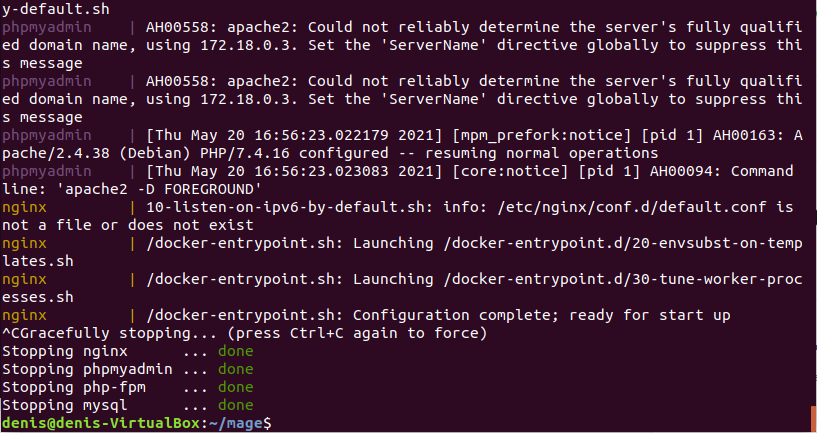
Сайт досі не працює



Входимо до phpMyAdmin через порт 8090

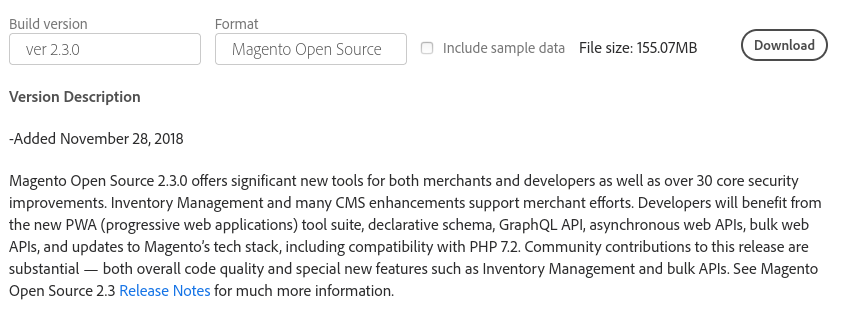


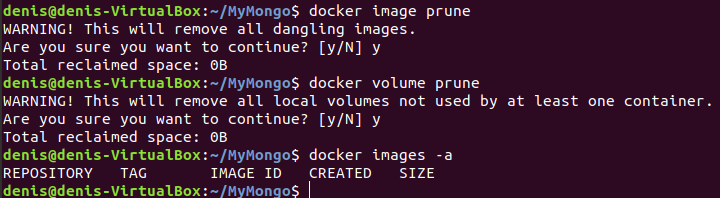
Готовий до роботи з базою даних

Відключення серверу



За допомогою пошуку в інтернеті дізнаємося, що ця інструкція встановлення magento2 була актуальна в 2018 році

 Я ще пробував встановити сервер через нижчу версію magento2, проте не вдалося



Важливі команди для остаточного видалення томів та образів

**Висновок**

Під час роботи над лабораторною роботою були здобуті керування Dockerfile та docker-compose.yml. Були вивчені команди для створення скриптів Dockerfile, що дозволило створити контейнер, в якому можна працювати з mongodb. Docker-compose допоміг розгорнути декілька контейнерів, передати їм необхідні дані для коректної співпраці сервісів.