Міністерство освіти і науки України

НТУУ «КПІ ім. Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

Лабораторна робота №9.2 з дисципліни «Операційна система UNIX»

**Створення проекту, що складається з контейтерів Django+PostgreSQL, з використанням Docker Compose та Dockerfile**

**Варіант: 16**

Виконав студент групи ТР-12

Каркушевський Владислав

Перевірила д.т.н., проф. Левченко Л. О.

КИЇВ 2022

**Мета роботи :**

- ознайомитися та набути навичок:

- написання скрипта Dockerfile для створення контейнеру;

- встановлення Docker Compose в Ubuntu 18.04;

- створення контейнерів (сервісів): Django+PostgreSQL для розроблення додатку.

**Теоретична частина**:

Docker – це стандартизоване пакетне програмне забезпечення, призначене для розробки, розгортання проектів та використання розроблених додатків, які є переносимими та самодостатніми, у той час як метою віртуальної машини є емуляція іншого операційного середовища.

Docker дозволяє відокремити ваш додаток від вашої інфраструктури і дозволяє запустити будь-який додаток, який безпечно ізольований у контейнері.

Docker використовується для створення, тестування і розгортання додатка. За допомогою Docker можна:

- зробити процес налаштування простіше, спростити налаштування на рівні інфраструктури;

- допомогти розробникам зосередитися виключно на коді, скорочуючи час розробки і збільшуючи продуктивність;

- посилити можливості налагодження з використанням вбудованих функцій;

- ізолювати додатки;

- поліпшити щільність використання серверів у формі контейнеризації;

- робити швидке розгортання на рівні операційної системи.

Життєвий цикл програмного забезпечення з використанням контейнера Docker полягає у проходженні наступних стадій:

docker create - створення контейнера без його запуску, може бути використана для зберігання і виведення ідентифікатора контейнера для майбутнього використання,

docker run - робота контейнера: створити і запустити контейнер,

docker stop - призупинення роботи контейнера,

docker start - відновлення роботи контейнера: запустити існуючий зупинений контейнер,

docker restart - перезапуск контейнера,

docker rm - видалення контейнера,

docker kill - відправити сигнал SIGKILL контейнеру про завершення роботи,

docker attach - підключитися до працюючого контейнера,

docker wait - блокувати команду і чекати, поки контейнер не зупиниться

Контейнери - це технологія упаковки і запуску додатків Windows, Linux, MacOS в різних локальних середовищах і в хмарі. Контейнер створюється і складається з образів. Образ - базовий елемент кожного контейнера. Кожному образу Docker відповідає файл, який називається Dockerfile.

Dockerfile -це звичайний текстовий файл, який являє собою сценарій, що складається з послідовності команд і аргументів, необхідних для створення Docker образу. Кожна команда, яка модифікує файлову систему створює новий шар.

Такі сценарії спрощують розгортання і процес підготовки програми до запуску. Спочатку Dockerfile визначає образ, на основі якого буде відбуватися збирання образу. Потім йде ряд методів, команд і аргументів, які створюють новий образ. Вміст Dockerfile передається демона Docker для збирання образу. Усі подібні файли починаються з позначення FROM (базового образу, наприклад, FROM ubuntu) так як і процес побудови нового контейнера, далі йдуть різні методи, команди, аргументи або умови, після застосування яких створиться

Docker контейнер.

Усі команди в Докерфайлах прийнято вказувати великими літерами - наприклад RUN, CMD і т.д. Наступні команди надають інструменти для створення образів та роботи з ними:

- 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%

- 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»

- 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% Наприклад, FROM centos

- Команда ADD - бере два аргументи, шлях звідки скопіювати файл і шлях куди скопіювати файли у власну файлову систему контейнера. Синтаксис команди: ADD [вихідний шлях або URL] [шлях призначення]

- Команда ENTRYPOINT встановлює конкретний додаток за замовчуванням, який використовується кожний раз в момент побудови контейнера за допомогою образу. Синтаксис команди: ENTRYPOINT %додаток% «аргумент»

- 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%

- 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%

- Команда MAINTAINER не є виконуваною, вона визначає значення поля автора образу. Найкраще її вказувати відразу після команди FROM. Синтаксис команди: MAINTAINER %ваше\_ім'я%

- Команда USER - використовується для установки UID або імені користувача, яке буде використовуватися в контейнері. Синтаксис команди: USER %ID\_користувача%

- 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» ...]

- Команда WORKDIR вказує директорію, з якої буде виконуватися команда CMD. Синтаксис команди: WORKDIR /шлях

Нові образи контейнерів створюються за допомогою файлів Dockerfile і команди docker build. Для команди docker build необхідно: Dockerfile і контекст створення образу (build context) (який може бути порожнім). Контекст створення - це набір локальних файлів і каталогів, до яких можна звертатися з інструкцій ADD і / або COPY в Dockerfile і які зазвичай визначаються як шлях до потрібного каталогу. Наприклад, в каталозі test створюється файл образ dockerfile: docker build -t test/dockerfile .При запуску команди docker build для створення нового образу вважається, що Dockerfile знаходиться у поточному робочому каталозі – визначено як крапка «.» (це і є контекст створення). Це означає, що усі файли і каталоги, розташовані по зазначеному шляху, формують контекст створення образу і передаються в демон Docker як частина процесу створення. Якщо цей файл знаходиться в якомусь іншому місці, його розташування можна вказати з використанням прапорця «-f»

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

Порядок роботи Compose.

При роботі з Compose найбільш часто використовуються наступні команди:

up - запуск всіх контейнерів, визначених у Compose-файлі. Виведення журнальних записів об'єднується в один потік. У більшості випадків використовується аргумент «-d» для запуску Compose у фоновому режимі;

build - Перетворення усіх образів, створених з файлів Dockerfile. Команда up буде створювати образи, тільки якщо вони не існували раніше, тому команду build слід використовувати за необхідності поновлення образів;

ps - виведення інформації про стан контейнерів, які керовані Compose;

run - одноразовий запуск контейнера з виконанням однієї команди (не в якості сервісу). Також запускаються всі контейнери, з якими повинні бути встановлені з'єднання, якщо не заданий аргумент «--no-deps». (Команди, що передаються через run, замінюють команди, певні в файлі конфігурації сервісу. Крім того, за замовчуванням команда run не створює портів, визначених у файлі конфігурації сервісу);

logs - виведення журнальних записів з кольоровою підсвіткою, об'єднаний для всіх контейнерів, керованих Compose;

stop - зупинка контейнерів без їх видалення;

rm - видалення зупинених контейнерів. Слід пам'ятати про те, що аргумент «-v» дозволяє видалити всі томи, керовані механізмом Docker.

**Завдання:**

Створити два контейнери для розроблення web-додатків:

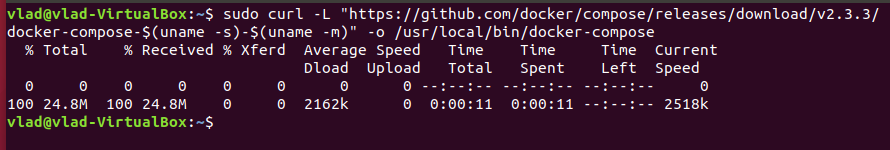
- повнофункціонального серверного веб-фреймворка Django, написаного на Python,

- вільної об'єктно-реляційної системи управління базами даних PostgreSQL.

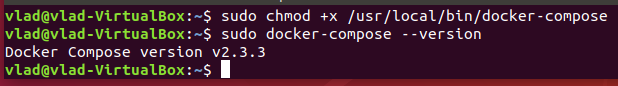
**Хід виконання роботи**

**Установка Docker Compose в Ubuntu 18.04**

Встановлюємо останню версію Docker Compose для управління додатками з декількома контейнерами. Перевіряємо поточну версію, за необхідності оновимо її за допомогою наступної команди

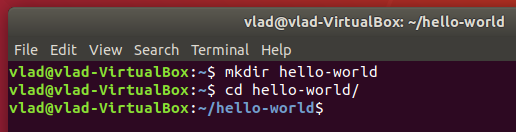


Встановимо дозвіл (+x – на виконання / rwx), тобто зробимо файл виконуваним та перевіримо чи установка пройшла успішно за допомогою перевірки версії

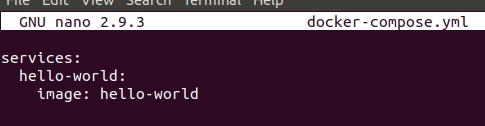


**Запуск контейнера за допомогою Docker Compose**

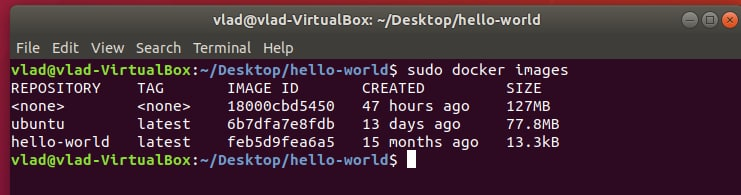
Створимо директорію hello-world для файлу YAML і перейдемо в неї



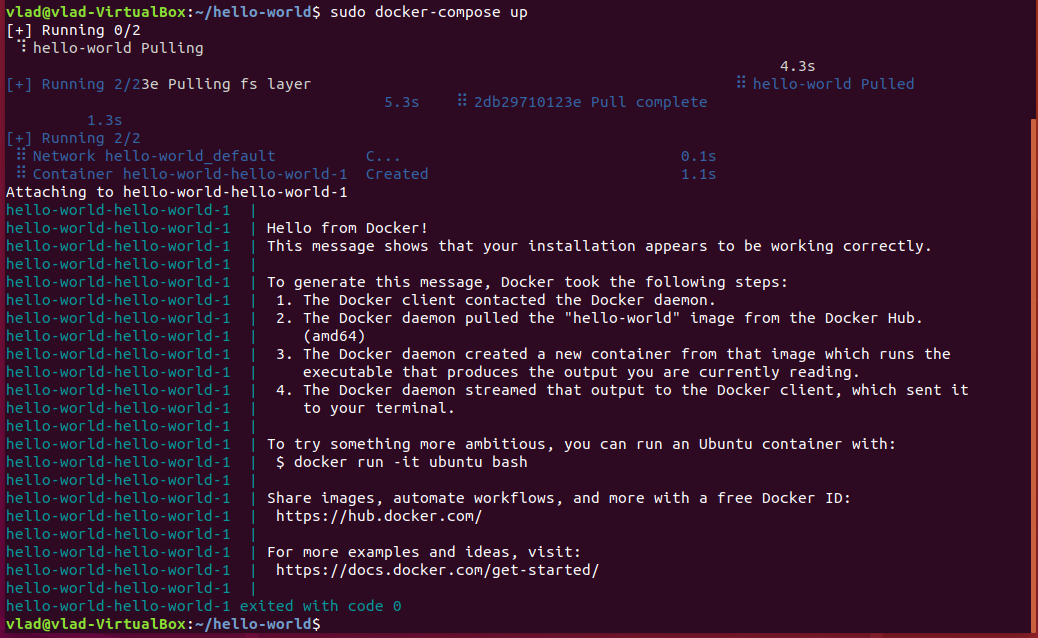
Створимо в цій директорії файл YAML та за допомогою команди nano перейдемо у цей файл. Пропишемо у ньому такий зміст



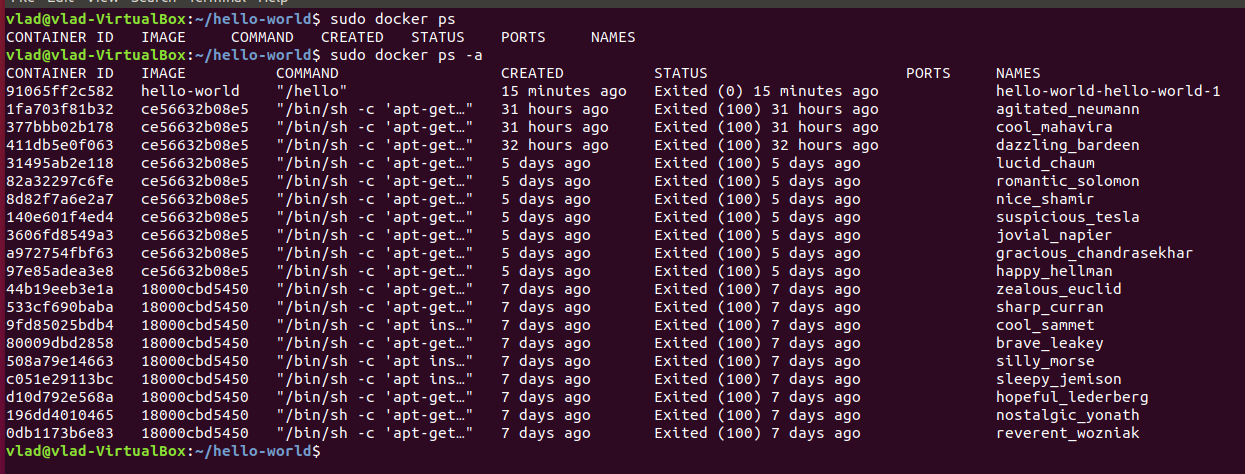
Ми можемо вручну переглянути образи в нашій системі за допомогою команди docker images



При запуску команди docker-compose up вона буде шукати локальний образ за вказаним іменем, тобто hello-world.

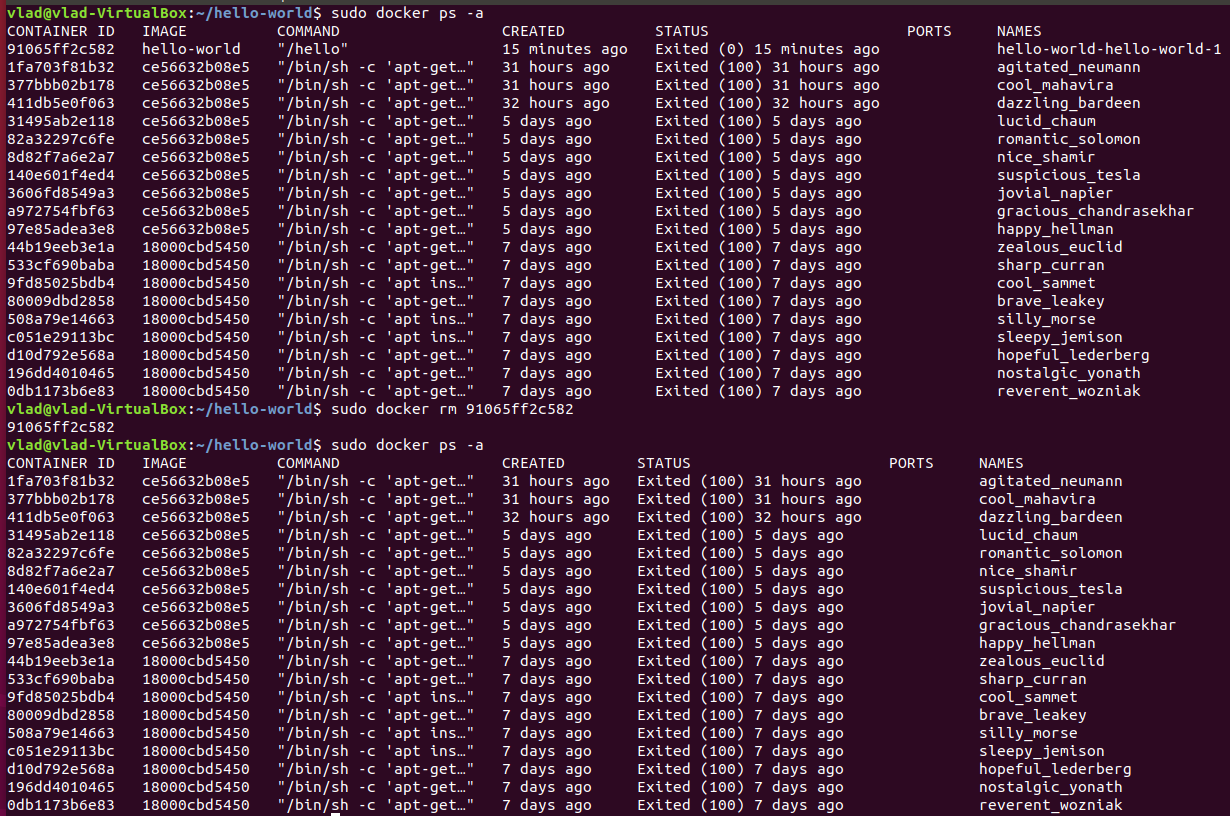


Контейнери Docker продовжують працювати, поки команда залишається активною, тому після завершення роботи hello контейнер зупиняється. Отже, коли ми переглядаємо активні процеси, заголовки стовпців будуть з'являтися, але контейнер hello-world НЕ буде з'являтися в списку, оскільки він не запущений. Переглянемо інформацію контейнера, яка нам буде потрібна на наступному кроці, використовуючи ключ «-a», за допомогою якого можна відобразити всі контейнери, а не тільки активні

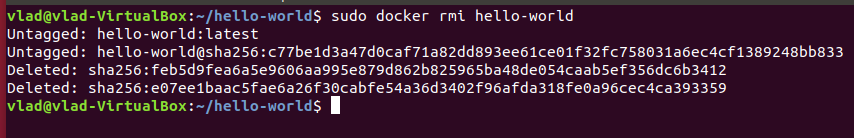


**Видалення способу**

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.

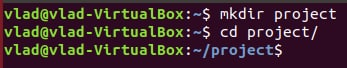


Після видалення всіх контейнерів, які містять образ, ми можемо видалити образ

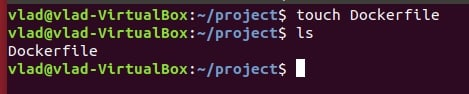


**Визначення компонентів проекту**

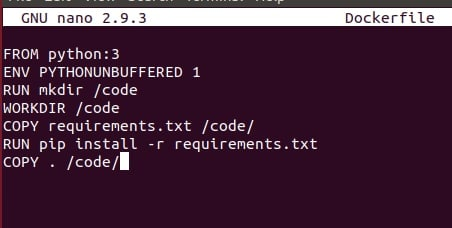
Створюємо порожній каталог для проекту з назвою project



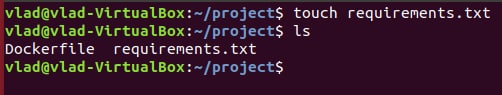
Створюємо новий файл під назвою Dockerfile у своєму проекті.



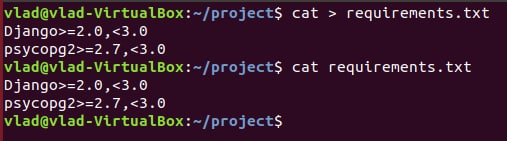
Додайте наступний вміст у файл Dockerfile та зберігаємо.



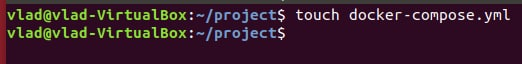
Створюємо файл вимог requirements.txt у каталозі проекту



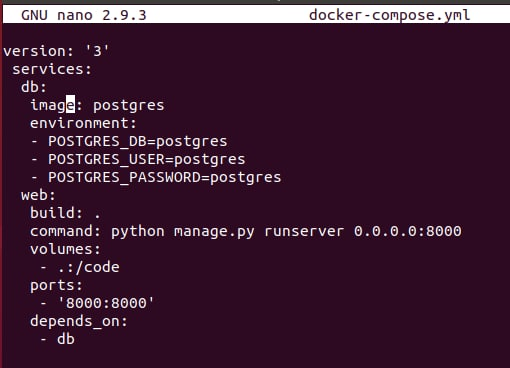
Додаємо необхідне програмне забезпечення у файл.



Створіть файл з назвою docker-compose.yml у каталозі проекту.



Додаємо у файл наступну конфігурацію

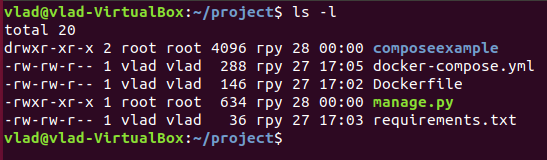


**Створити проект Django**

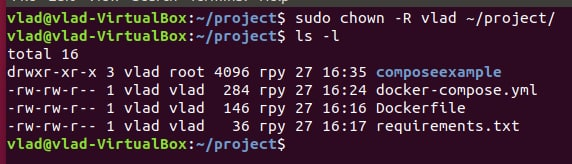
Створюємо проект Django, виконуючи команду run для docker-compose наступним чином



Після завершення команди Docker-Compose, виводимо список вмісту проекту

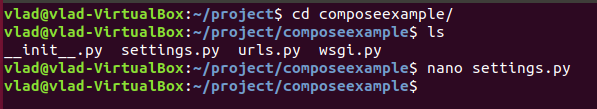


Оскільки Linux при роботі з Docker створені файли django-admin належать користувачу root тому треба змінити власника на нові файли



**Підключення бази даних**

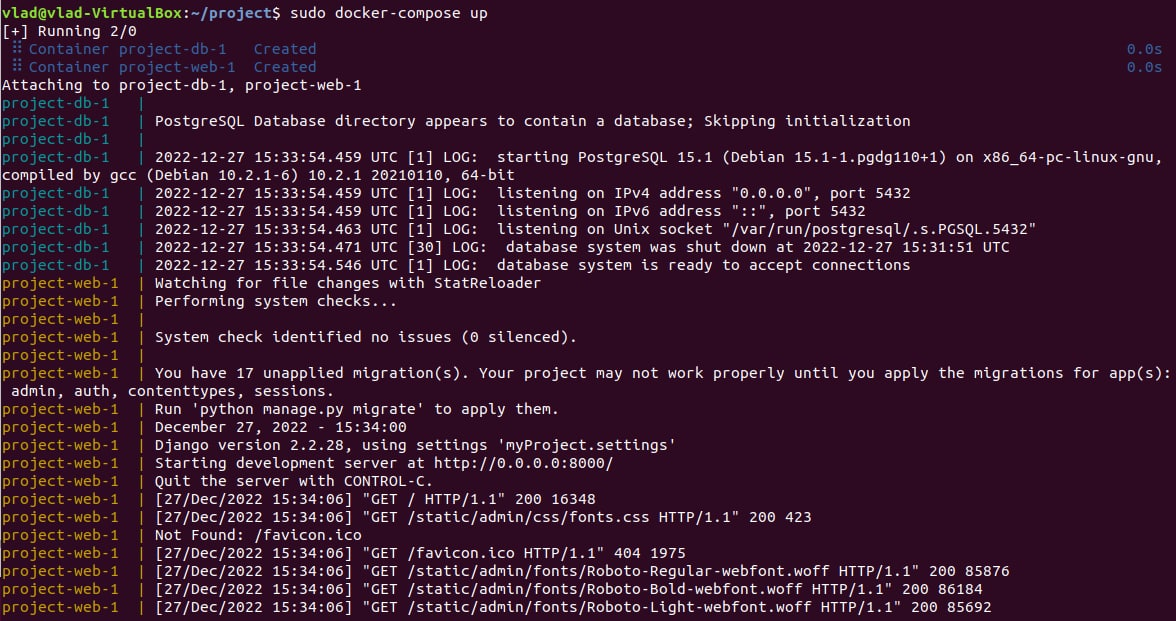
Відредагуємо файл composeexample/settings.py у каталозі проекту.



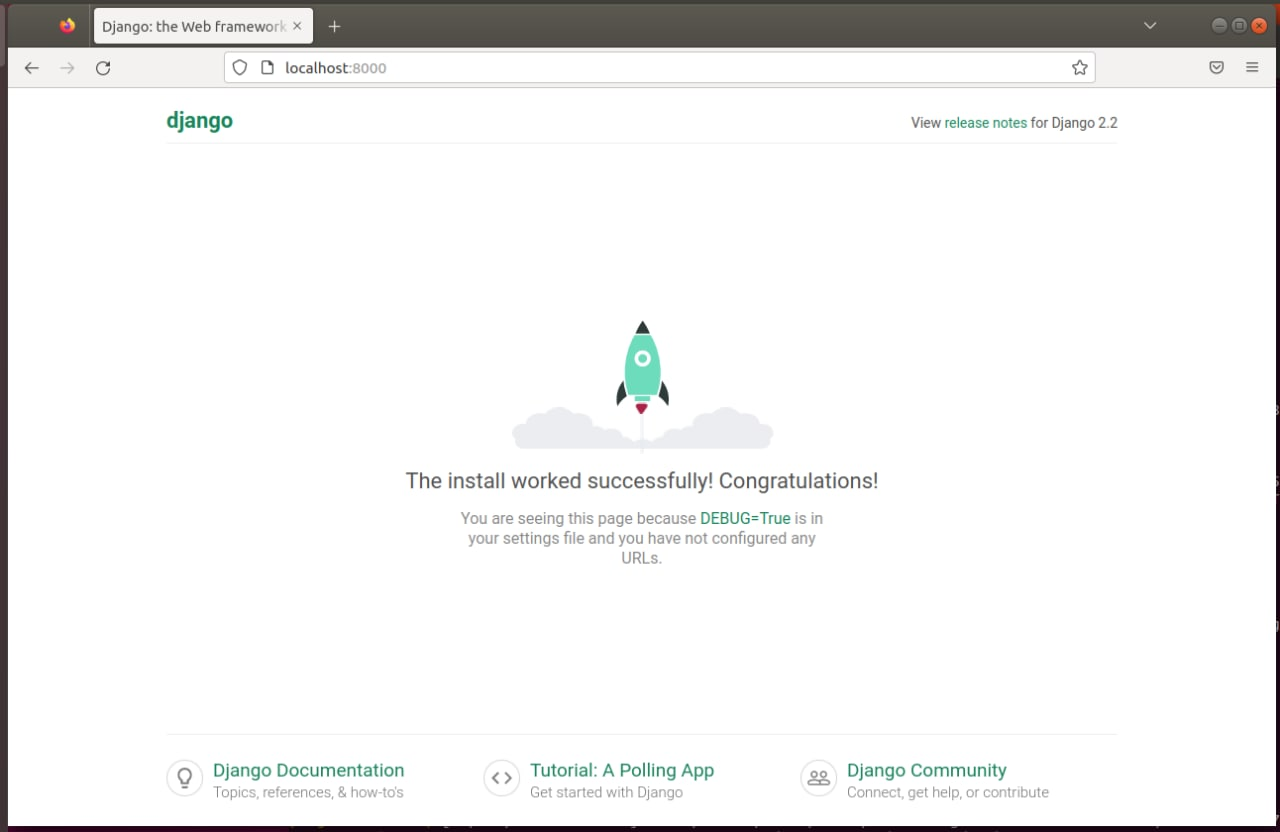
Замінимо DATABASES = ... на наступне:



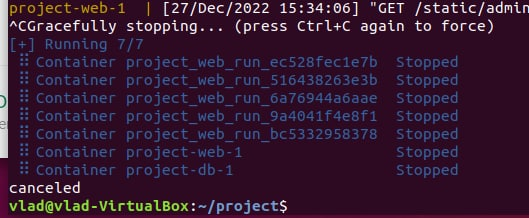
Запустимо команду docker-compose up з каталогу верхнього рівня.



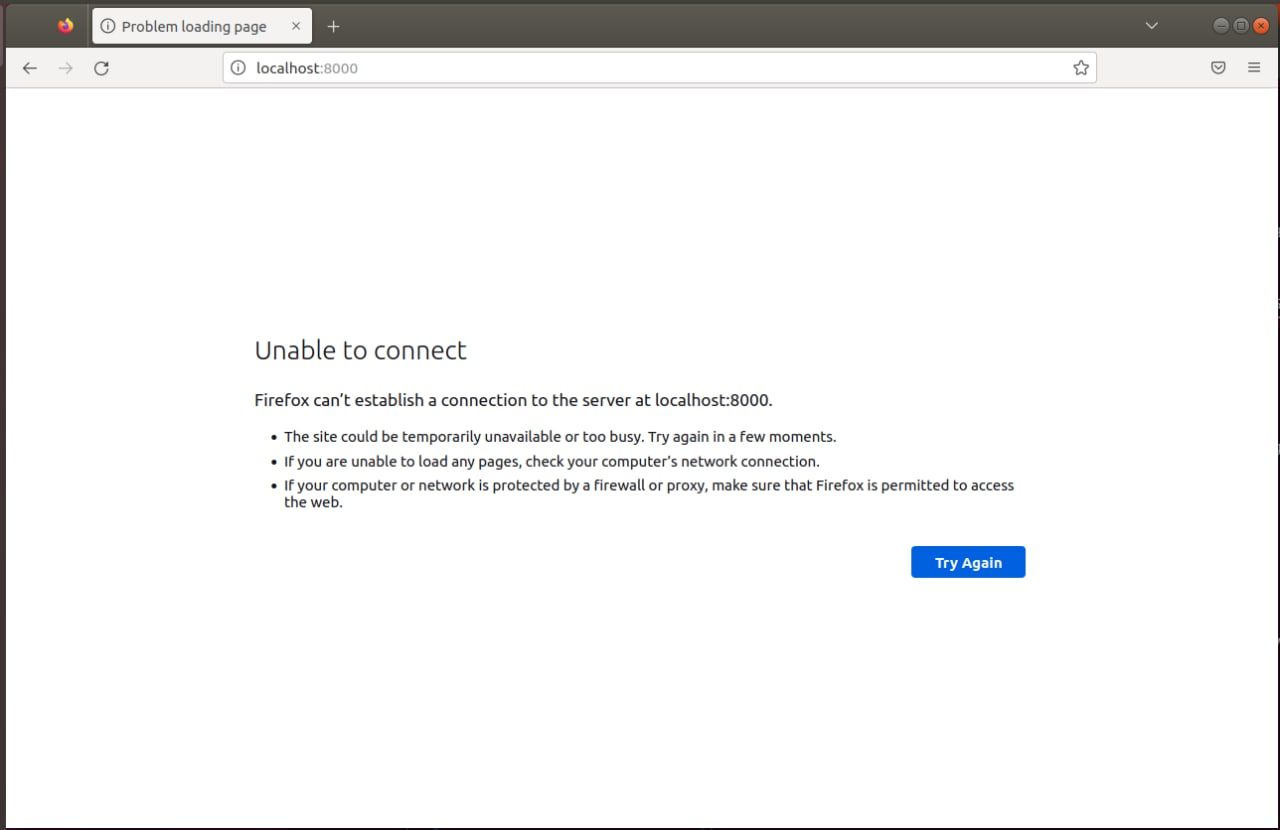
Перейдемо на localhost:8000 аби переглянути результат роботи контейнера. Перейшовши на веб-браузер http://localhost:8000 побачимо сторінку вітання Django.



Далі потрібно зупинити контейнер, натиснувши Ctrl+C



Та зайдемо на localhost:8000 , та перевіримо що контейнер не запущений.



**Висновок:** На цій лабораторній роботі було ознайомлено з теорію по лабораторній роботі. Встановлено Docker Compose в Ubuntu 18.04. Написано скрипт Dockerfile для створення контейнеру. Створено контейнери: Django+PostgreSQL для розроблення додатку