

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»

Тема «Створення проекту, що складається з контейнерів
Django + PostgreSQL, з використанням Docker Compose та
Dockerfile»

Варіант №22

Студента 2-го курсу НН ІАТЕ гр. ТР-12

Ковальова Олександра

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

Мета роботи. Ознайомитися та набути навичок написання скрипта Dockerfile для створення контейнеру, встановлення Docker Compose, створення контейнерів (сервісів) Django + PostgreSQL для розробки web-додатку.

Теоретична частина.

Docker-compose – це утиліта від авторів оригінального Docker, яка дозволяє об'єднати процес створення контейнерів. Для цього використовуються yaml файли з назвою docker-compose.yml. Compose використовує файли YAML (YAML Ain't Markup Language – мова серіалізації даних) для зберігання конфігурації груп контейнерів. На початку треба вказати версію утиліти, яку можна дізнатись за допомогою команди docker-compose –v. Після цього треба вказати ключове слово version, і в лапках версію (після двокрапки). Потім, після ключового слова services можна перелічувати контейнери, та вказувати їм певні властивості.

Повний опис усіх команд Compose знаходиться на сайті Docker <https://docs.docker.com/compose/reference/>.

Docker-compose створює один образ (base) на основі всіх, що входять в нього.

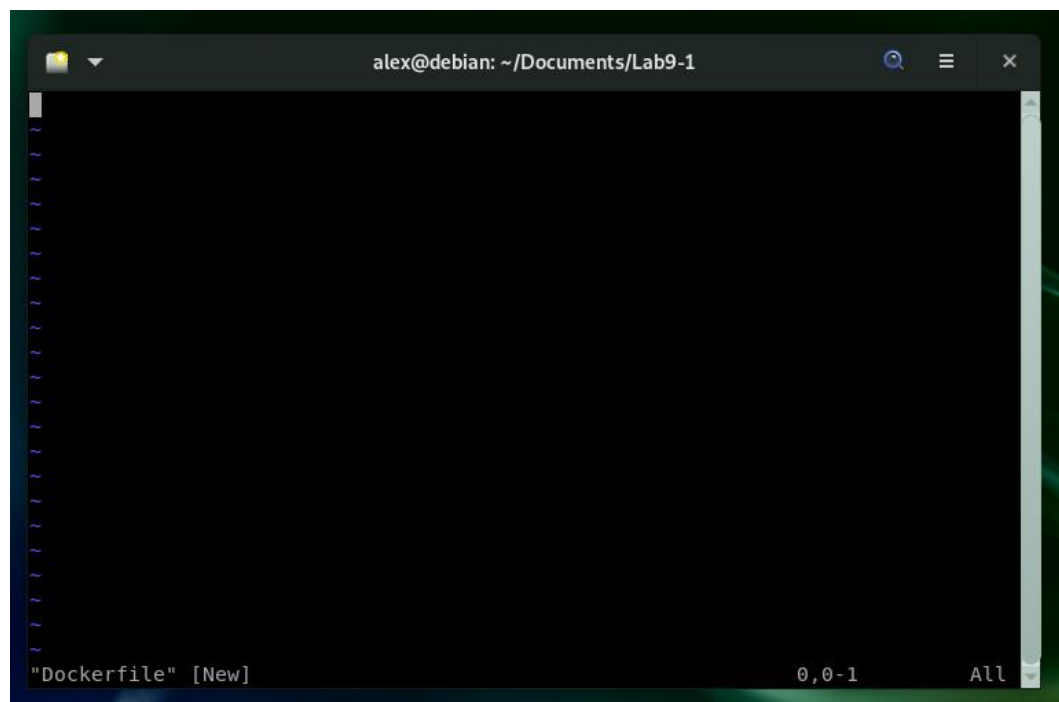
Також, утиліта має механізм Docker з'єднання (links) – найпростіший спосіб забезпечення обміну інформацією між контейнерами на одному хості. Тобто передається інформація про IP-адресу та відкриті порти з одного контейнера у другий. Це дуже спрощує роботу.

Теоретична частина (на практиці).

MongoDB – найбільш популярна нереляційна база даних. Нереляційна – тобто, не використовує стандартну схему таблиць та стовбців [1]. В цих базах даних використовується модель зберігання даних, яка оптимізована під конкретні вимоги типу збережених файлів.

Задача: створити власний образ для встановлення MongoDB.

Створимо порожній файл з назвою Dockerfile (без розширення) за допомогою утиліти vim. Команда – vim Dockerfile:



Коментарі в Dockerfile треба писати після символу решітки. Образ будемо створювати на основі Debian. Запишемо це:

```
alex@debian: ~/Documents/Lab9-1
#####
# Dockerfile to build MongoDB container images
# Based on Debian
#####
```

Для того щоб вказати Dockerfile'у що хочемо створити базовий образ на основі певного, треба вказати ключове слово From. Цей рядок обов'язковий:

```
alex@debian: ~/Documents/Lab9-1
#####
# Dockerfile to build MongoDB container images
# Based on Debian
#####

# Set the base image to Debian
FROM debian
```

Після цього використовуємо інструкцію RUN – вона запускає певні команди в процесі створення образу. Зараз вона потрібна для того, щоб оновити список пакетів та встановити пакет gnupg – він відповідає за створення та підписання електронних цифрових підписів. Використовуємо ключ -y, який означає «На всі питання Так чи Ні відповідати Так». Тобто, якщо оболонка запитає, чи можна скачати п мегабайт для пакету – не треба вказувати свою відповідь, вона автоматично буде «Так». Окрім цього, доцільно було б встановити пакет software-properties-common. За допомогою нього можна додавати репозиторії в список без редагування файлу sources.list, використовуючи команду add-apt-repository. Також, знадобиться команда wget – для неї потрібно завантажити пакет. Виведення займає багато місця, тому переправляємо звичайне виведення (STDIN, дескриптор каналу 1) в пустоту. Будуть виводитись на екран лише помилки та попередження.

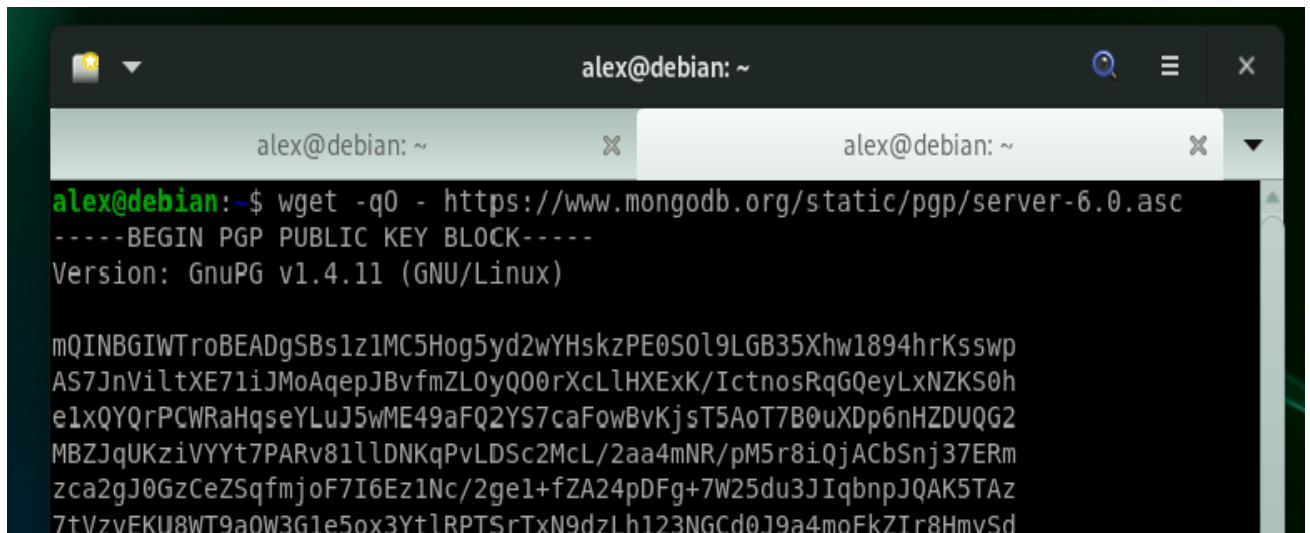
```
# Update the repository sources list and install gnupg2, software-properties-common, wget
RUN apt-get update 1> $VOID

RUN apt-get install -y gnupg 1> $VOID
RUN apt-get install -y software-properties-common 1> $VOID
RUN apt-get install -y wget 1> $VOID
```

В якості пустоти використовується змінна VOID. Вона записана за допомогою ключового слова ENV. В неї записана адреса /dev/null. Це спеціальний файл, який є символьним пристроєм, представляє собою «пустоту». Таким чином можна подавити виведення, переправляючи все в цей файл.

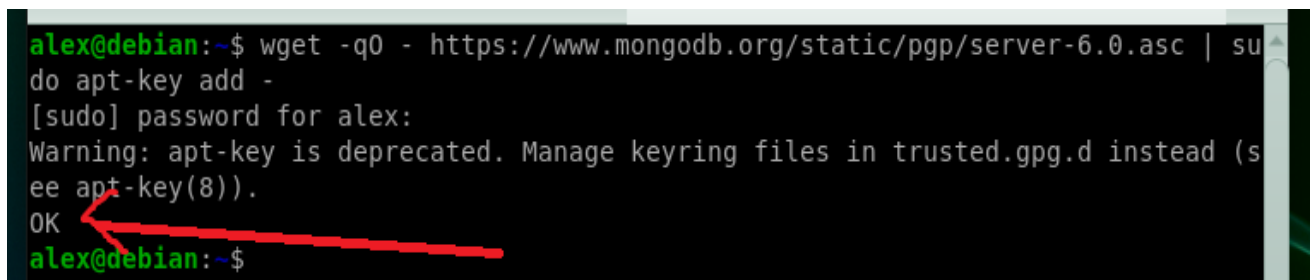
```
ENV VOID /dev/null
```

Додамо ключ верифікації пакету за допомогою утиліти apt-key. Для початку, ключ потрібно отримати – це можна зробити програмою wget, яка відповідає за завантаження певних даних [2]. Ключ -q0 означає, що помилки не треба виводити. Також додаємо посилання на офіційний сайт mongodb, звідки буде завантажуватись ключ. Запустимо команду та отримаємо результат:



```
alex@debian: ~  
alex@debian:~$ wget -q0 - https://www.mongodb.org/static/pgp/server-6.0.asc  
-----BEGIN PGP PUBLIC KEY BLOCK-----  
Version: GnuPG v1.4.11 (GNU/Linux)  
  
mQINBGIWTroBEADgSBs1z1MC5Hog5yd2wYHskzPE0S0l9LGB35Xhw1894hrKsswp  
AS7JnViltXE71iJMoAqepJBvfmZLOyQ00rXcLlHXExK/IctnosRqGQeyLxNZKS0h  
e1xQYQrPCWRaHqseYLuJ5wME49aFQ2YS7caFowBvKjsT5AoT7B0uXDp6nHZDUQG2  
MBZJqUKziVYYt7PARv81llDNKqPvLDSc2McL/2aa4mNR/pM5r8iQjACbSnj37ERm  
zca2gJ0GzCeZSqfmjoF7I6Ez1Nc/2ge1+fZA24pDFg+7W25du3JIqbnpJQAK5TAz  
7tVzvEKU8WT9aQW3G1e5ox3YtlRPTSrTxN9dzLh123NGCd0J9a4moFkZIr8HmySd
```

Цей ключ передаємо за допомогою конвеєра утиліті apt-key, яка додає його до бази своїх ключів.

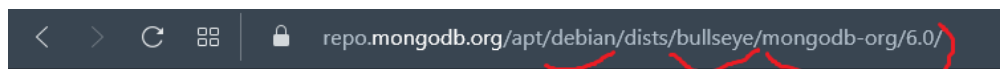


```
alex@debian:~$ wget -q0 - https://www.mongodb.org/static/pgp/server-6.0.asc | su  
do apt-key add -  
[sudo] password for alex:  
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (s  
ee apt-key(8)).  
OK  
alex@debian:~$
```

Додамо цю команду в Dockerfile.

```
#####  
# Dockerfile to build MongoDB container images  
# Based on Debian  
#####  
  
# Set the base image to Debian  
FROM debian  
  
ENV VOID /dev/null  
  
# Update the repository sources list and install gnupg2, software-properties-common, wget  
RUN apt-get update 1> $VOID  
  
RUN apt-get install -y gnupg 1> $VOID  
RUN apt-get install -y software-properties-common 1> $VOID  
RUN apt-get install -y wget 1> $VOID  
  
# Add the package verification key  
RUN wget -q0 - https://www.mongodb.org/static/pgp/server-6.0.asc | apt-key add -
```

За допомогою команди `add-apt-repository` з пакету `software-properties-common` додаємо репозиторій в список репозиторіїв apt – файл `sources.list`, який знаходиться за адресою `/etc/apt/sources.list`. Але – вказане зеркало з дистрибутивами (<http://downloads-distro.mongodb.org/repo/ubuntu-upstart>) вже не працює [3]. На заміну йому прийшов новий (<https://repo.mongodb.org>). Заходимо на нього, та обираємо потрібну версію:



Index of 6.0

[Parent Directory](#)

[Release](#)

[Release.gpg](#)

[main](#)

repo.mongodb.org

Додаємо це посилання в список репозиторіїв. Оновлюємо список пакетів. Встановлюємо пакет `mongodb-org`.

```
# Add MongoDB to the repository sources list
RUN add-apt-repository "deb http://repo.mongodb.org/apt/debian bullseye/mongodb-org/6.0 main"

# Update the repository sources list
RUN apt-get update

# Install MongoDB package (.deb)
RUN apt-get install -y mongodb-org 1> $VOID
```

Створюємо каталог для збереження даних.

```
# Create the default data directory
RUN mkdir -p /data/db
```

Даємо знати контейнеру, що зв'язок буде налаштований через порт 27017:

```
# Expose the default port
EXPOSE 27017
```

Відкриваємо для «зовнішнього світу» порт 27017, який є стандартним (дефолтним) для MongoDB:

```
# Default port to execute the entrypoint
CMD ["--port 27017"]
```

Встановлюємо додаток за замовчуванням: демон (службу) mongod.

```
# Set default container command
ENTRYPOINT usr/bin/mongod
```

Створюємо докер образ за допомогою команди build.

Команда: `sudo docker build -t alex-mongodb .`

Команда `build` використовується для збирання контейнеру в образ. Ключ `-t` означає, що ми хочемо присвоїти ім'я (tag) образу. Крапка в кінці означає «контекст» – тобто, в якому каталогі ми працюємо, в якому каталогі знаходиться `Dockerfile`.

```
alex@debian: ~/Documents/Lab9-1/Mongo
alex@debian:~/Documents/Lab9-1/Mongo$ sudo docker build -t alex-mongodb .
[sudo] password for alex:
Sending build context to Docker daemon  2.56kB
Step 1/14 : FROM debian
latest: Pulling from library/debian
32de3c850997: Pull complete
Digest: sha256:c66c0e5dc607baefefda1d9e64a3b3a317e4189c540c8eac0c1a06186fe353a1
Status: Downloaded newer image for debian:latest
--> 446440c01886
Step 2/14 : ENV VOID /dev/null
--> Running in 8e914433cd4b
Removing intermediate container 8e914433cd4b
--> 351f0ce99ef0
Step 3/14 : RUN apt-get update 1> $VOID
--> Running in 65e51c397f21
```

Можемо бачити, що всі команди виконуються покроково і окремо. Весь образ ділиться на шари, які можемо бачити на скріншоті вище.

При завантаженні є деякі попередження. Наприклад, те що продемонстроване нижче – це попередження пов'язане з утилітою `apt-get`. У неї є консольний інтерфейс, і через це `Docker` викидує дане попередження. Для того щоб це полагодити, треба встановити пакет `apt-utils`, та встановити значення для декількох змінних. Але, це подавить навіть попередження – тому, залишаємо як є. Це `safe-ignored` попередження, тобто його можна ігнорувати [4].

```
--> 83e895130302
Step 4/14 : RUN apt-get install -y gnupg 1> $VOID
--> Running in fbc9b452e0d7
debconf: delaying package configuration, since apt-utils is not installed
Removing intermediate container fbc9b452e0d7
--> b732daf5283d
Step 5/14 : RUN apt-get install -y software-properties-common 1> $VOID
```

Також є попередження щодо використання утиліти `apt-key`. Вона застарівша, і хорошою практикою вважається використання `gpg`. Але – це довше (код) та потребує додаткових пакетів, не дуже добре навантажувати образ зайвими пакетами.

```
--> 2be1fc98ec4c
Step 7/14 : RUN wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc | apt-key add -
--> Running in 79f8c73a815a
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
Removing intermediate container 79f8c73a815a
--> c3b98e471156
```

Образ успішно зібрався:

```
---> Running in 16458df6c36a
Removing intermediate container 16458df6c36a
---> d059a2c64e5e
Step 14/14 : ENTRYPOINT usr/bin/mongod
---> Running in 6be7a89e005a
Removing intermediate container 6be7a89e005a
---> 45c94c735662
Successfully built 45c94c735662
Successfully tagged alex-mongodb:latest
alex@debian:~/Documents/Lab9-1/Mongo$
```

За допомогою команди `docker images` переглянемо список образів. Як бачимо, образ підписаний. Також є базовий образ `debian`.

```
---> 45c94c735662
Successfully built 45c94c735662
Successfully tagged alex-mongodb:latest
alex@debian:~/Documents/Lab9-1/Mongo$ docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
alex-mongodb         latest          45c94c735662   21 minutes ago  913MB
debian               latest          446440c01886   6 days ago     124MB
alex@debian:~/Documents/Lab9-1/Mongo$
```

Запускаємо контейнер за допомогою команди `docker run` з ключами `-it` (interactive, terminal) та `--name` для того щоб призначити ім'я новому контейнеру та запустити його в інтерактивному режимі. Створюємо його на основі нашого образу, який щойно був згенерований. Можемо бачити багато інформації в json форматі – це означає, що все працює добре.

```
alex@debian: ~/Documents/Lab9-1/Mongo
alex@debian:~/Documents/Lab9-1/Mongo$ sudo docker run -it --name AlexMongoDB alex-mongodb
{"t":{"$date":"2022-12-27T19:11:05.156+00:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"-",
"msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"
{"t":{"$date":"2022-12-27T19:11:05.161+00:00"},"s":"I", "c":"NETWORK", "id":4915701, "ctx":"mai
n", "msg":"Initialized wire specification. Supported: ['compression', 'incomingExternalClient', 'incomingVncsi
```

Docker Compose

Завантажуємо останню версію програми з GitHub за допомогою утиліти `curl`. Робимо файл виконуваним використовуючи `chmod`. Дивимось версію програми:

```
alex@debian: /usr/local/bin
alex@debian:~/Documents/Lab9-1/Mongo$ sudo curl -L "https://github.com/docker/compose/releases/download/v2.14.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
% Total    % Received % Xferd  Average Speed   Time    Time     Time    Current
           Dload  Upload   Total   Spent    Left   Speed
  0     0    0     0    0     0      0      0  --:--:-- --:--:-- --:--:--    0
100 42.8M 100 42.8M    0     0 6974k    0  0:00:06 0:00:06 --:--:-- 10.2M
alex@debian:~/Documents/Lab9-1/Mongo$ sudo chmod +x /usr/local/bin/docker-compose
alex@debian:~/Documents/Lab9-1/Mongo$ docker-compose --version
Docker Compose version v2.14.2
alex@debian:~/Documents/Lab9-1/Mongo$
```


Створимо файл docker-compose.yml для запуску контейнера hello-world:

```
alex@debian: ~/Documents/Lab9-1/hello-world$ cat docker-compose.yml
services:
  my-test:
    image: hello-world
```

Спочатку треба написати ключове слово “services”. Після нього повинен йти перелік сервісів та дії з ними.

Наш сервіс називається my-test та базується на образі hello-world. Запустимо командою docker compose up:

```
alex@debian: ~/Documents/Lab9-1/hello-world$ docker-compose up
[+] Running 2/2
  # my-test Pulled                                3.3s
  # 2db29710123e Pull complete                    0.8s
[+] Running 1/1
  # Container hello-world-my-test-1 Created        0.8s
Attaching to hello-world-my-test-1
hello-world-my-test-1 |
hello-world-my-test-1 | Hello from Docker!
hello-world-my-test-1 | This message shows that your installation appears to be working correctly.
hello-world-my-test-1 |
hello-world-my-test-1 | To generate this message, Docker took the following steps:
hello-world-my-test-1 | 1. The Docker client contacted the Docker daemon.
hello-world-my-test-1 | 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
hello-world-my-test-1 |    (amd64)
hello-world-my-test-1 | 3. The Docker daemon created a new container from that image which runs the
hello-world-my-test-1 |    executable that produces the output you are currently reading.
hello-world-my-test-1 | 4. The Docker daemon streamed that output to the Docker client, which sent
hello-world-my-test-1 |    it
hello-world-my-test-1 |    to your terminal.
hello-world-my-test-1 |
hello-world-my-test-1 | To try something more ambitious, you can run an Ubuntu container with:
hello-world-my-test-1 | $ docker run -it ubuntu bash
hello-world-my-test-1 |
```

Переглянемо список запущених сервісів через docker-compose, docker. Також переглянемо список образів докер.

```
alex@debian: ~/Documents/Lab9-1/hello-world$ docker-compose ps -a
NAME                IMAGE              COMMAND             SERVICE    CREATED          STATUS          PORTS
hello-world-my-test-1 hello-world        "/hello"            my-test    5 minutes ago   Exited (0) 5 minutes ago

alex@debian: ~/Documents/Lab9-1/hello-world$ docker ps -a
CONTAINER ID   IMAGE      COMMAND             CREATED          STATUS          PORTS          NAMES
3bbcd5a2f794   hello-world "/hello"            5 minutes ago   Exited (0) 5 minutes ago           hello-world-my-test-1
518716aa4e89   alex-mongodb "/bin/sh -c usr/bin/" 2 hours ago     Exited (130) 2 hours ago           AlexMongoDB

alex@debian: ~/Documents/Lab9-1/hello-world$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
alex-mongodb   latest   45c94c735662   2 hours ago     913MB
debian         latest   446440c01886   6 days ago      124MB
hello-world    latest   feb5d9fea6a5   15 months ago   13.3kB
```


Видалимо контейнер, а потім образ:

```
alex@debian: ~/Documents/Lab9-1/hello-world
alex@debian:~/Documents/Lab9-1/hello-world$ docker rm hello-world-my-test-1
hello-world-my-test-1
alex@debian:~/Documents/Lab9-1/hello-world$ docker image rm hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:c77be1d3a47d0caf71a82dd893ee61ce01f32fc758031a6ec4cf1389248bb833
Deleted: sha256:feb5d9fea6a5e9606aa995e879d862b825965ba48de054caab5ef356dc6b3412
Deleted: sha256:e07ee1baac5fae6a26f30cabfe54a36d3402f96afda318fe0a96cec4ca393359
alex@debian:~/Documents/Lab9-1/hello-world$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
518716aa4e89   alex-mongodb   "/bin/sh -c usr/bin/..." 2 hours ago    Exited (130) 2 hours ago          AlexMongoDB
alex@debian:~/Documents/Lab9-1/hello-world$ docker images
REPOSITORY    TAG          IMAGE ID      CREATED        SIZE
alex-mongodb   latest       45c94c735662  3 hours ago    913MB
debian         latest       446440c01886  6 days ago     124MB
alex@debian:~/Documents/Lab9-1/hello-world$
```

Хід роботи

Завданням є формування робочої збірки контейнерів Django + PostgreSQL. Всі сервіси потрібно розкидати по контейнерах та скомпонувати за допомогою docker-compose.

Django – це повнофункціональний вільний веб-фреймворк, написаний мовою програмування Python (та використовується в зв'язці з нею). Використовує шаблон проектування MVC (Model View Controller).

PostgreSQL – вільна об'єктно-реляційна система управління базами даних. Реляційна – тобто використовує стандартний підхід з таблицями та стовбцями. Об'єктна – використовує деякі концепції об'єктно-орієнтованого програмування, наприклад, наслідування, об'єкти, класи.

Створимо порожній каталог з назвою Website. В ньому потрібно створити файл Dockerfile (обов'язково без розширення) – це файл-інструкція, по якому Docker буде будувати образ та створювати з нього контейнер.

```
alex@debian: ~/Documents/Lab9-2/Website
alex@debian:~/Documents/Lab9-2$ mkdir Website
alex@debian:~/Documents/Lab9-2$ cd Website/
alex@debian:~/Documents/Lab9-2/Website$ touch Dockerfile
alex@debian:~/Documents/Lab9-2/Website$ vim Dockerfile
```

На початку використовуємо ключове слово FROM. Воно обов'язково повинно бути на початку файлу, бо без нього в коді немає сенсу. Це ключове слово визначає який образ є батьківським. В нашому випадку, це python версії 3 (Вказується через двокрапку)

```
FROM python:3
ENV PYTHONUNBUFFERED 1
RUN mkdir /code
WORKDIR /code
COPY requirements.txt /code/
RUN pip install -r requirements.txt
COPY . /code/
```

Команда ENV використовується для установки змінних середовища. Нам потрібна лише одна змінна – PYTHONUNBUFFERED, якій присвоюємо значення 1. Це потрібно для того, щоб все що йде в потоки STDOUT та STDERR, які прикріплені до терміналу, не буферизувалось, тобто щоб все йшло в режимі реального часу на екран.

Наступна команда – RUN. Після неї повинні йти Linux команди. Використовуємо mkdir для створення каталогу /code.

За допомогою команди WORKDIR призначаємо цей каталог робочим. Тобто, для команди CMD поточним каталогом буде саме /code.

COPY – скопіювати щось з одного каталогу в інший. Копіюємо файл з залежностями пакетів (створимо наступним кроком) в /code.

PIP – інструмент для встановки пакетів Python. Власне, його назва і є аббревіатурою від Python Install Packages. Ключ -r для команди install використовується, щоб вказати файл з залежностями. Встановлюємо всі потрібні пакети зі списку (ще не створеного)

Копіюємо все з поточного каталогу в /code.

Створимо файл вимог requirements.txt. Нам потрібна версія Django між другою включно та третьою. Також, потрібен пакет psycopg2 – це адаптер між Django та PostgreSQL:

```
alex@debian: ~/Documents/Lab9-2/Website
Django>=2.0,<3.0
psycopg2>=2.7,<3.0
```

Створимо файл docker-compose.yml. В цьому файлі описуються сервіси, які створюють додаток, як вони будуть взаємодіяти між собою, які порти їм можуть знадобитися. Також, описуються відкриті порти сервісів.

```
version: '2.14.2'

services:
  db:
    image: postgres
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres

  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - ./code
    ports:
      - "8000:8000"
    depends_on:
      - db
```

На початку обов'язково вказується версія docker-compose. Її можна перевірити за допомогою команди `docker-compose -v`.

Після цього йде перелік сервісів. Першим є сервіс `db` – Database. Параметр `image` відповідає за те, з якого образу буде сформований новий контейнер. Після цього йдуть змінні, які передаються контейнеру (ключове слово `environment`). `POSTGRES_DB` – параметр, який відповідає за назву бази даних. `POSTGRES_USER`, `POSTGRES_PASSWORD` – логін та пароль відповідно.

Наступний сервіс – `web`. Батьківського образу тут немає, так як будувати будемо на основі `Dockerfile`. Встановлюємо поточний каталог `(.)` параметру `build`. Параметр `command` відповідає за те, які команди будуть запущені при ввімкненні контейнеру. У нашому випадку буде відбуватись запуск файлу `manage.py` (спеціальний файл Django) утилітою `Python`, яка запускає `Python` код. Передаємо аргументи `“runserver 0.0.0.0:8000”`, тобто відкрити сервер на порту 8000.

Створюємо контейнеру том, до якого монтуємо наш каталог `code`. Прокидуємо порт 8000. `depends_on` вказує, який контейнер повинен бути запущений перед ним, в нашому випадку це сервіс `db`.

Запускаємо `docker-compose`:

```
alex@debian:~/Documents/Lab9-2/Website$ sudo docker-compose run web django-admin.py
startproject composeexample .
[+] Running 8/14
  db Pulling                                     31.4s
    #3f4ca61aafcd Pull complete                  8.8s
    #048d3078d446 Pull complete                  11.6s
    #c6d23b4fe6c1 Pull complete                  12.2s
    #d846f6946dd5 Pull complete                  13.5s
    #76f7157f330d Pull complete                  18.0s
    #4eacfb0464b2 Pull complete                  21.7s
    #5c197e2b597b Pull complete                  22.6s
    #2c4576649951 Pull complete                  22.8s
    #1ae267d32d50 Extracting                     77.99MB/92.22MB 28.9s
    #03048c1132b5 Download complete              28.9s
    #bdee410b6909 Download complete              28.9s
    #d3354a8bfb14 Download complete              28.9s
    #0105a87d8ff9 Download complete              28.9s
```

`Docker-compose` за допомогою команди `run` запускає сервіс `web`, якому ми передали аргументами команду `«django-admin startproject composeexample .»`, тобто створення проекту `django` з назвою `composeexample`. Так як ще немає образу `web`, то він будується з поточної директорії.

Бачимо, що `Docker` створив новий контейнер `website-db-1` та запустив його.

```
# 0105a87d8ff9 Pull complete
[+] Running 1/1
  Container website-db-1 Created                  5.1s
[+] Running 1/1
  Container website-db-1 Started                  6.1s
[+] Building 211.1s (10/11)
=> [internal] load build definition from Dockerfile 1.1s
=> => transferring dockerfile: 185B 0.2s
=> [internal] load .dockerignore 0.9s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/python:3 9.5s
=> [1/6] FROM docker.io/library/python:3@sha256:250990a809a15bb6a3e307 200.1s
=> => resolve docker.io/library/python:3@sha256:250990a809a15bb6a3e307fe 0.3s
```

Переглянемо список файлів за допомогою команди `ls -l` (`ls` – list, ключ `-l` = long):

```
alex@debian:~/Documents/Lab9-2/Website$ ls -l
total 20
drwxr-xr-x 2 root root 4096 Dec 30 23:51 composeexample
-rw-r--r-- 1 alex alex 329 Dec 30 23:48 docker-compose.yml
-rw-r--r-- 1 alex alex 146 Dec 30 23:43 Dockerfile
-rwxr-xr-x 1 root root 634 Dec 30 23:51 manage.py
-rw-r--r-- 1 alex alex 36 Dec 30 23:44 requirements.txt
alex@debian:~/Documents/Lab9-2/Website$
```

Бачимо, що створився проект Django в директорії `composeexample`. Перша літера `-d` – означає що це директорія. Відсутність букви `-` – те, що це файл. Після цього йдуть 3 блоки по 3 літери – властивості файлу (Чи можна прочитати, записати в нього, та виконати) для власника, групи власника та для інших користувачів. Після цього число посилок на файл, потім ім'я власника, і після цього група. Змінимо власника з `root` на себе за допомогою команди `chown -R $USER:$USER .`, де ключ `-R` означає, що каталог потрібно рекурсивно, `$USER` – змінна оболонки (в цьому місці вказується ім'я та група), `«.»` – поточний каталог:

```
alex@debian:~/Documents/Lab9-2/Website$ sudo chown -R $USER:$USER .
alex@debian:~/Documents/Lab9-2/Website$ ls -l
total 20
drwxr-xr-x 2 alex alex 4096 Dec 30 23:51 composeexample
-rw-r--r-- 1 alex alex 329 Dec 30 23:48 docker-compose.yml
-rw-r--r-- 1 alex alex 146 Dec 30 23:43 Dockerfile
-rwxr-xr-x 1 alex alex 634 Dec 30 23:51 manage.py
-rw-r--r-- 1 alex alex 36 Dec 30 23:44 requirements.txt
alex@debian:~/Documents/Lab9-2/Website$
```

Налаштуємо підключення до бази даних. Для цього потрібно відредагувати файл `settings.py`:

```
alex@debian:~/Documents/Lab9-2/Website$ tree .
.
├── composeexample
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── docker-compose.yml
├── Dockerfile
├── manage.py
└── requirements.txt

1 directory, 8 files
alex@debian:~/Documents/Lab9-2/Website$ vim composeexample/settings.py
```

В цьому файлі зберігаються різноманітні налаштування. Нам потрібен список `DATABASES`:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

На початку в ньому вказаний тільки «рушій» який відповідає бази даних MySQL та назву БД. Нам потрібно змінити ці параметри та додати нові. MySQL змінюємо на postgresql, назву бази даних, ім'я, пароль – на postgres, хост – на db (наш сервіс), порт на 5432, який є стандартним для PostgreSQL. Зберігаємо файл:

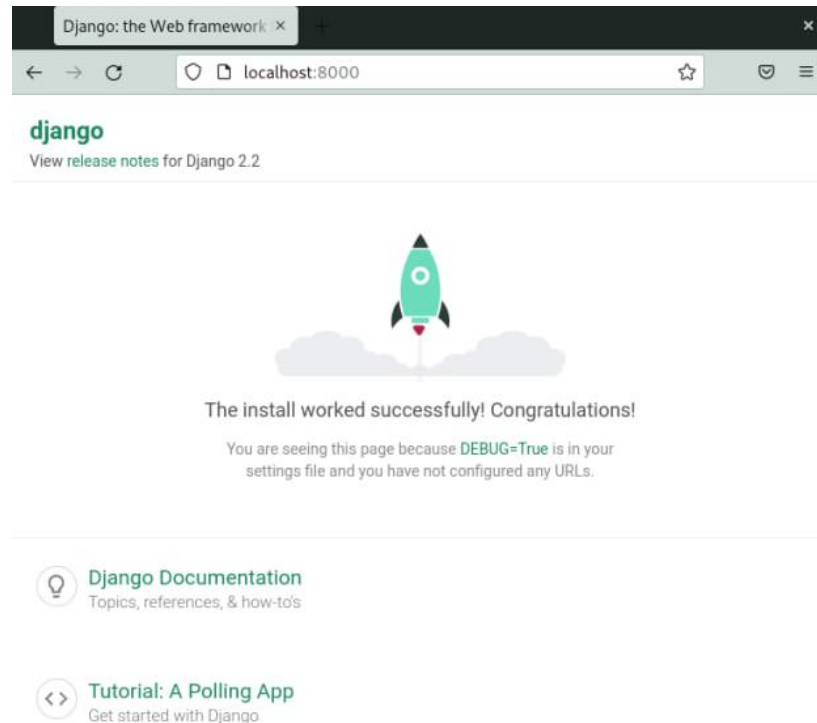
```
# Database
# https://docs.djangoproject.com/en/2.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'postgres',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': 'db',
        'PORT': 5432
    }
}
```

Переходимо в корінь та прописуємо docker-compose up – щоб підняти всі сервіси. Бачимо як вони запускаються, дату запуску. Також, можемо побачити статус запитів GET – 200 OK. Тобто, все добре.

```
alex@debian: ~/Documents/Lab9-2/Website x alex@debian: ~/Documents/Lab9-2/Website x
alex@debian:~/Documents/Lab9-2/Website$ docker-compose up
[*] Running 2/2
 3 Container website-db-1 Running 3-04
 3 Container website-web-1 Created 1-29
Attaching to website-db-1, website-web-1
website-web-1 | Watching for file changes with StatReloader
website-web-1 | Performing system checks...
website-web-1 |
website-web-1 | System check identified no issues (0 silenced).
website-web-1 |
website-web-1 | You have 17 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
website-web-1 | Run 'python manage.py migrate' to apply them.
website-web-1 | December 30, 2022 - 22:00:42
website-web-1 | Django version 2.2.28, using settings 'composeexample.settings'
website-web-1 | Starting development server at http://0.0.0.0:8000/
website-web-1 | Quit the server with CONTROL-C.
website-web-1 | [30/Dec/2022 22:03:11] "GET / HTTP/1.1" 200 16348
website-web-1 | [30/Dec/2022 22:03:12] "GET /static/admin/css/fonts.css HTTP/1.1" 200 423
website-web-1 | Not Found: /favicon.ico
website-web-1 | [30/Dec/2022 22:03:12] "GET /favicon.ico HTTP/1.1" 404 1980
website-web-1 | [30/Dec/2022 22:03:12] "GET /static/admin/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 200 85876
website-web-1 | [30/Dec/2022 22:03:12] "GET /static/admin/fonts/Roboto-Bold-webfont.woff HTTP/1.1" 200 86184
website-web-1 | [30/Dec/2022 22:03:13] "GET /static/admin/fonts/Roboto-Light-webfont.woff HTTP/1.1" 200 85692
```

В браузері переходимо на localhost, обов'язково вказуємо порт 8000, який був відкритий заздалегідь та відповідає сервісу web.



Бачимо привітальну сторінку, а це означає, що все працює. Зупиняємо контейнери за допомогою команди `docker-compose stop`:

```
alex@debian:~/Documents/Lab9-2/Website$ docker-compose stop
[+] Running 2/2
  # Container website-web-1 Stopped      4.9s
  # Container website-db-1 Stopped      3.7s
alex@debian:~/Documents/Lab9-2/Website$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
alex@debian:~/Documents/Lab9-2/Website$
```

Контейнери зупинені. Завдання виконане.

Робота була виконана на останній версії дистрибутиву Debian – 11 (Bullseye). Характеристики системи, виведені за допомогою утиліти `neofetch`:

```
alex@debian
-----
OS: Debian GNU/Linux 11 (bullseye) x86_64
Host: VirtualBox 1.2
Kernel: 5.10.0-20-amd64
Uptime: 7 hours, 1 min
Packages: 2065 (dpkg)
Shell: bash 5.1.4
Resolution: preferred
DE: GNOME 3.38.6
WM: Mutter
WM Theme: Adwaita
Theme: Green-Submarine [GTK2/3]
Icons: Adwaita [GTK2/3]
Terminal: gnome-terminal
CPU: Intel i3-8130U (2) @ 2.207GHz
GPU: 00:02.0 VMware SVGA II Adapter
Memory: 640MiB / 1982MiB
```


Контрольні запитання:

1) Що таке Docker Compose?

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.

2) Що таке Dockerfile?

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

3) Які вам відомі команди для роботи з Dockerfile?

FROM – основа образу, RUN – запустити команду, тощо

4) У чому полягає алгоритм створення проекту для розроблення web застосування?

Алгоритм полягає у компонуванні всіх сервісів (мікросервісів) між собою. Якщо це робити без Docker – то все це з великою ймовірністю не буде працювати на іншій машині, тому що, можливо, не будуть встановлені залежності, або ще з якихось причин. Docker ізолює всі сервіси по контейнерам, і тому вони будуть працювати будь-де. Але, ці контейнери потрібно пов'язати між собою, з цією задачею справляється програма Docker Compose.

Висновок: за результатами виконання цієї лабораторної роботи було ознайомлено з процесом написання скриптів Dockerfile, створенням контейнерів, компонуванням за допомогою Docker Compose. Також, був створений веб-додаток за допомогою веб-фреймворка для Python під назвою Django та бази даних PostgreSQL.

Додаткові джерела:

- 1) [MongoDB.com – MongoDB documentation](https://www.mongodb.com/docs/)
- 2) [Linux.die.net – wget utility documentation](https://linux.die.net/~wget/)
- 3) [Jira.MongoDB.com – Is repo down permanently?](https://jira.mongodb.com/browse/IS-1000)
- 4) [StackOverflow – apt-utils problems](https://stackoverflow.com/questions/40710000/apt-utils-problems)