

Лабораторна робота № 9.2

Створення проекту, що складається з контейнерів Django+PostgreSQL, 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): **Django+PostgreSQL** для розроблення додатку.

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 Docker 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 Docker файлах 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 User!»

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 % ім'я_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 Докерфайл 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:

Открыть ▾



*Dockerfile
~/Загрузки

Сохранить

☰

⏮

🖼

✖

```
# 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
```

Текст ▾ Ширина табуляции: 8 ▾ Стр 29, Стлб 27 ▾ ВСТ

```
#####
# 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.

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

А точка в кінці означає що Докерфайл знаходиться в тій же категорії, з якої виконується команда.

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 прапора «*-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:

```
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 – 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 Hub*, 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*** (це аналог команди ***docker run***) 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

| <i>REPOSITORY</i> | <i>TAG</i> | <i>IMAGE ID</i> | <i>CREATED</i> | <i>SIZE</i> |
|-------------------|------------|-----------------|----------------|-------------|
|-------------------|------------|-----------------|----------------|-------------|

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 два контейнери для розроблення web-додатків:

- повнофункціонального серверного веб-фреймворка ***Django***, написаного на Python,
- вільної об'єктно-реляційної системи управління базами даних ***PostgreSQL***.

Визначення компонентів проекту

Перед початком повинен бути встановлений *Docker Compose*. Для цього проекту потрібно створити *Dockerfile*, файл залежностей Python та файл *docker-compose.yml*. (Для цього файлу можна використовувати розширення *.yml* або *.yaml*.)

1. Створіть порожній каталог для проекту.

Доцільно назвати каталог таким, що легко запам'ятовується. Цей каталог є контекстом для вашого образу додатка. Каталог повинен містити ресурси лише для створення цього образу.

2. Створіть новий файл під назвою *Dockerfile* у своєму проєкті.

Dockerfile визначає вміст образу програми за допомогою однієї або декількох команд побудови, які налаштовують цей образ. Після побудови ви можете запустити образ в контейнер. Для отримання додаткової інформації про *Dockerfile* див. Посібник користувача Docker (<https://docs.docker.com/get-started/>) та посилання на *Dockerfile* (<https://docs.docker.com/engine/reference/builder/>).

3. Додайте наступний вміст у файл *Dockerfile*.

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

Цей *Dockerfile* починається із батьківського образу *Python 3*. Батьківський образ модифікується шляхом додавання нового каталогу коду. Батьківський образ додатково модифікується шляхом встановлення вимоги *Python*, які визначені у файлі вимог *requirements.txt*.

4. Збережіть та закрийте *Dockerfile*.

5. Створіть файл вимог *requirements.txt* у вашому каталозі проєкту.

Цей файл використовується командою ***RUN pip install -r requirements.txt*** у вашому *Dockerfile*.

6. Додайте необхідне програмне забезпечення у файл.

```
Django>=2.0,<3.0
psycopg2>=2.7,<3.0
```

7. Збережіть та закрийте файл вимог *requirements.txt*.

8. Створіть файл з назвою *docker-compose.yml* у вашому каталозі проєкту.

Файл *Docker-compose.yml* описує сервіси, які створюють ваш додаток.

У цьому прикладі ці сервіси - це **веб-сервер та база даних**.

Файл *compose* також описує які образи Докера ці сервіси використовують, як вони зв'язуються між собою, і які томи, які вони можуть знадобитися, встановлені всередині контейнерів. Нарешті, файл *docker-compose.yml* описує, які порти необхідні цим службам.

9. Додайте у файл наступну конфігурацію.

```
version: '3'

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

```

Цей файл визначає два сервіси: db-сервіс та веб-сервіс.

10. Збережіть і закрийте файл *docker-compose.yml*.

Створити проект Django

На цьому етапі створюємо проект для запуску *Django*, будуючи образ з контексту збірки, визначеної в попередній процедурі.

1. *Перейдіть до кореневого каталогу вашого проекту.*
2. *Створіть проект Django, виконуючи команду run для docker-compose наступним чином.*

sudo docker-compose run web django-admin startproject composeexample

Команда вказує *Compose* запустити *django-admin startproject composeexample* в контейнері, використовуючи образ та конфігурацію веб-сервіса. Оскільки веб-образу поки ще не існує, *Compose* створює його з поточного каталогу, як вказано у збірці: рядок в *docker-compose.yml*.

Після того, як побудовано образ веб-сервіса, *Compose* запускає його та виконує команду *django-admin startproject* у контейнері. Ця команда вказує *Django* створити набір файлів і каталогів, що представляють проект *Django*.

3. *Після завершення команди Docker-Compose, складіть список вмісту вашого проекту.*

```

$ ls -l
drwxr-xr-x 2 root    root    composeexample
-rw-rw-r-- 1 user    user    docker-compose.yml
-rw-rw-r-- 1 user    user    Dockerfile
-rwxr-xr-x 1 root    root    manage.py
-rw-rw-r-- 1 user    user    requirements.txt

```

Оскільки *Linux* при роботі з *Docker* створені файли *django-admin* належать користувачу *root* (це відбувається тому, що контейнер працює від імені користувача *root*), тому треба змінити власника на нові файли:

sudo chown -R \$USER:\$USER

Якщо ви працюєте з *Docker* на *MacOS* або *Windows*, ви вже повинні мати право власності на всі файли, включаючи файли, створені *django-admin*. Перерахуйте файли лише для того, щоб це підтвердити.

```
$ ls -l
total 32
-rw-r--r-- 1 user staff 145 Feb 13 23:00 Dockerfile
drwxr-xr-x 6 user staff 204 Feb 13 23:07 composeexample
-rw-r--r-- 1 user staff 159 Feb 13 23:02 docker-compose.yml
-rwxr-xr-x 1 user staff 257 Feb 13 23:07 manage.py
-rw-r--r-- 1 user staff 16 Feb 13 23:01 requirements.txt
```

Підключіть базу даних

У цьому розділі налаштуємо підключення до бази даних для *Django*.

1. Відредагуйте файл *composeexample/settings.py* у своєму каталозі проектів.
2. Замініть *DATABASES = ...* на наступне:

```
# setting.py

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

Ці налаштування визначаються образом *Postgres Docker*, який вказаний у *docker-compose.yml*.

3. Збережіть і закрийте файл.
4. Запустіть команду ***docker-compose up*** з каталогу верхнього рівня для вашого проекту.

```
$ docker-compose up
djangosample_db_1 is up-to-date
Creating djangosample_web_1 ...
Creating djangosample_web_1 ... done
Attaching to djangosample_db_1, djangosample_web_1
```

db_1 | The files belonging to this database system will be owned by user "postgres".
db_1 | This user must also own the server process.
db_1 |
db_1 | The database cluster will be initialized with locale "en_US.utf8".
db_1 | The default database encoding has accordingly been set to "UTF8".
db_1 | The default text search configuration will be set to "english".

...

web_1 | May 30, 2017 - 21:44:49
web_1 | Django version 1.11.1, using settings 'composeexample.settings'
web_1 | Starting development server at http://0.0.0.0:8000/
web_1 | Quit the server with CONTROL-C.

На даний момент ваш додаток *Django* повинен працювати в порту 8000 на хості Docker. На робочому столі Docker для Mac та Docker Desktop для Windows перейдіть на веб-браузер **http://localhost:8000**, щоб побачити сторінку вітання *Django*.

Якщо ви використовуєте *Docker Machine*, тоді IP MACHINE_VM *docker-machine* повертає IP-адресу хоста *Docker*, до якої ви можете додати порт (<Docker-Host-IP>: 8000).

На цьому установка закінчена.

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