Wirtualizacja systemów i sieci komputerowych – laboratorium

Ćwiczenie 4: Wprowadzenie do konteneryzacji. Podstawowe komendy i konfiguracja środowisk

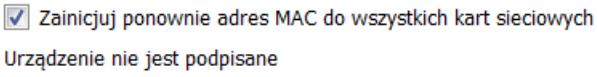
Celem ćwiczenia jest przybliżenie wybranych zagadnień z zakresu administracji kontenerami w systemach Linux na przykładzie narzędzia Docker CE

Wymagania wstępne:

Przygotowanie środowiska pracy należy rozpocząć od importu obrazu dockerlab.ova z dysku lokalnego.

W tym celu należy uruchomić narzędzie VirtualBox, następnie z menu File wybrać opcje importu pliku: „Importuj urządzenie wirtualne”

Należy zaznaczyć opcje:



Karta sieciowa powinna działać w trybie „bridge”., tryb nasłuchiwania „pozwalaj wszystkim”.

Stworzyć drugą bliźniaczą maszynę (opcjonalnie jak będzie potrzeba).

Do obu maszyn logowanie odbywa się za pomocą loginu: **vuko** i hasla:**1qaz2wsx**. Po zalogowaniu do systemu należy odczytać właściwy dla naszej karty sieciowej adres ip. Zaleca się by dalsze polecenie wykonywać za pomocą aplikacji Putty po ustanowieniu połączenia ssh na wskazany adres.

Czynności dodatkowe:
Wyłączyć defaultowy firewall dla Ubuntu komendą: *sudo ufw disable*
Oraz sprawdzić połączenie ssh: *service ssh status* ewentualnie wystartować.

Podstawowe zasady pracy z kontenerami:

Zweryfikować poprawność instalacji Dockera. Wydając polecenie ***sudo docker run hello-world*** powinien wyświetlić się ekran powitalny aplikacji. Polecenie **„run”** w aplikacji Docker uruchamia obraz.

```
lab@ubuntu:~$ sudo docker run hello-world
[sudo] password for lab:

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 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.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

Więcej opcji dla polecenie run wpisz: ***docker run –help*** w celu otworzenia manuala.

Poleceniem jest docker **„pull”,** służące do zaciągnięcia obrazów z zewnętrznego repozytorium:
Ma ono jeden parametr, którym jest nazwa obrazu jaki potrzeba pobrać. Wszystkie obrazy dostępne są w ogólnodostępnym repozytorium na stronie hub.docker.com

```
lab@ubuntu:~$ sudo docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
88286f41530e: Pull complete
Digest: sha256:1072e499f3f655a032e88542330cf75b02e7bdf673278f701d7ba61629ee3ebe
Status: Downloaded newer image for alpine:latest
```

Obrazy znajdujące się w lokalnym repozytorium można podejrzeć poleceniem ***docker images***

```
lab@ubuntu:~$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              7328f6f8b418        2 weeks ago         3.97MB
hello-world         latest              1815c82652c0        4 weeks ago         1.84kB
```

Aby uzmysłowić sobie jak łatwo na podstawie obrazu uruchomić w pełni działająca aplikacje pobierzmy jeszcze jeden obraz o nazwie **sonarqube** – jest to aplikacja do analizy jakości kodu aplikacji.

```
lab@ubuntu:~$ sudo docker pull sonarqube
Using default tag: latest
latest: Pulling from library/sonarqube
c75480ad9aaf: Pull complete
18d67befbc4e: Pull complete
1f5d2d0853c7: Pull complete
5de358416a75: Pull complete
4049b231edea: Pull complete
6617c62c7c10: Pull complete
aa26fbcddb08: Pull complete
```

Na podstawie pobranego obrazu uruchomiony zostanie pierwszy dzialajacy kontener:
polecenie:***sudo docker run -d --name mysonar -p 9000:9000 sonarqube***

gdzie: parametr **d** oznacza, iż kontener będzie działał w tle, „**mysonar**” nazwa tworzonego kontenera, parametr „p” pokazuje jak mają zostać przekierowane porty miedzy hostem a kontenerem. Aby zweryfikować, iż kontener z aplikacją rzeczywiście działa należy na hoście w przeglądarce gdzie zainstalowany jest Oracle Virtualbox wpisać adres: http://DOCKER_HOST_IP:9000, gdzie za DOCKER_HOST_IP podajemy ip maszyny wirtualnej. W przeglądarce powinna ukazać się działająca aplikacja sonarqube.

Statusy kontenerów można podejrzeć w aplikacji Docker polecenie: ***sudo docker ps –a***

```
lab@ubuntu:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORT
S                   NAMES
bab336f516d7        hello-world         "/hello"            35 minutes ago      Exited (0) 35 minutes ago
                    admiring_bell
```

Za pomocą podanej instrukcji i prostej pętli for istnieje możliwość uruchomienia oraz stopowanie/usuwania wielu instancji aplikacji w zaledwie kilka sekund. Sytuacja taka jest niewykonalna za pośrednictwem standardowych maszyn wirtualnych.

Możesz interaktywnie wykonywać polecenia na kontenerze z sonarqube. Dla przykładu wykonajmy polecenie grep oraz sprawdzmy używaną w nim wersję javy:

docker exec -ti mysonar grep java.command /opt/sonarqube/conf/wrapper.conf
docker exec -ti mysonar java -version

Uruchom komendę (ls –l) wewnątrz kontenera:

docker run --name=task4 alpine ls –l

Wystartuj powłokę w kontenerze:
docker run --name=task5_0 alpine /bin/sh

Powtórz polecenie z flagą –ti:
docker run -ti --name=task6 alpine /bin/sh

Wykonaj kilka przykładowych poleceń wewnątrz kontenera np.:
ls –l
uname –a
mkdir /testdir

Aby wyjść naciśnij **crtl+p+q**

Wykonaj commit wprowadzonych zmian: **docker commit task6 image_from_task6**, sprawdź jego poprawność za pomocą: ***docker images*** oraz ***docker diff task5***

Ściągnij teraz obraz przykładowej aplikacji busybox wykonując polecenie: ***docker pull busybox***
Otaguj busybox : ***docker tag busybox my_busybox1*** oraz ***docker tag busybox my_busybox1***

Ściągnij dowolny inny obraz z repozytorium dockera a następnie postaraj się go usunąć za pomocą polecenia: ***docker rmi nazwa_obrazu***
Zastopuj jeden z kontenerów: ***docker stop nazwa_kontenera*** oraz usuń go: ***docker rm nazwa_kontenera***

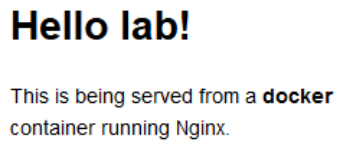
Możesz teraz sprawdzić statystki pozostałych kontenerów : ***docker stats***
Dla każdego z nich możliwe jest prześledzenie jakie procesy działają wewnątrz kontenera: ***docker top nazwa_kontenera***, więcej szczegółów o statnie kontenera dostarcza komenda: ***docker inspect nazwa_kontenera***. Jeśli potrzeba sprawdzić logi można posłużyć się komendą: ***docker logs nazwa_kontenera*** lub ***docker logs –ft nazwa_kontenera.***

Ściągnij obraz **seqvence/static-site** z repozytorium, spróbuj wykonać polecenie **run** w którym zawrzesz:
- nazwę konterena „static-site”
- definicję autora w zmiennej środowiskowej AUTHOR=imie
- uruchomisz go w tle i przekierujesz porty wewnętrzne 23456 na port 80
- narzucisz limit wykorzystania pamięci na 300M

W tym celu należy wydać polecenie:
docker run --name static-site2 -e AUTHOR="You" -d -p "23456:80" -m 300M seqvence/static-site

```
lab@ubuntu:~$ sudo docker run --name static-site2 -e AUTHOR="lab" -d -p "23456:80" -m 300M seqvence/
static-site
```

Można teraz zobaczyć efekt w przeglądarce wpisując ip hosta wraz z portem 23456



Skasuj teraz wszystkie kontenery, które istnieją. Dla każdego z nich wydaj polecenie: ***docker rm –f nazwa_kontenera***

Sieć w środowisku Docker

Sprawdź jaka sieć działa dla Twojego środowiska:
docker network ls

Stwórz kontener testowy:
docker run -d -P --name webapp training/webapp python app.py

Pobierz obraz ubuntu:
docker pull ubuntu

Wykonaj komendę: ***docker run -itd --name=networktest ubuntu***

Sprawdź ustawienia sieci w trybie bridge:
docker network inspect bridge

Odłącz kontener z sieci:
docker network disconnect bridge networktest

Stwórz połączenie: my-bridge-network
docker network create -d bridge my-bridge-network

Ponownie sprawdź jak prezentuje się ekran sieci: ***docker network ls***

Przyjrzyj się typowi stworzonej wcześniej sieci:
docker network inspect my-bridge-network

Podłącz inny kontener pod stworzoną sieć:
docker run -d --name db --network=my-bridge-network --name db training/postgres

Uruchom kontener web1:
docker run -d --name web1 training/webapp python app.py

Sprawdź czy masz połączenie między web1 a db
docker exec -ti web1 ping db

Jeśli brak połączenia wydaj polecenie podłączające kontener pod sieć:
docker network connect my-bridge-network web1

Sprawdź ponownie połączenie za pomocą ping z web1

Skasuj teraz wszystkie kontenery.
Skasuj wszystkie istniejące obrazy.

Wolumeny w środowisku Docker

Wszystkie wcześniej stworzone bądź istniejące kontenery i obrazy, jeżeli nie zostały usunięte to należy usunąć!

Uruchom kontener testowy z opcją –v pozwalającą na podłączenie wolumenu na dane:

docker run -d -P --name web -v /webapp training/webapp python app.py

Sprawdź poprawność utworzonego kontenera:***docker inspect web***

Wypróbuj alternatywną metodę tworzenia kontenera z wolumenem:
docker run -d -P --name web2 -v /src/webapp:/webapp training/webapp python app.py

Sprawdź co się zmieniło w porównaniu do wcześniejszego wywołania: ***docker inspect web2***

Przetestuj również kolejne alternatywne podejście:
docker run -d -P --name web3 -v /src/webapp:/webapp:ro training/webapp python app.py

Stwórz kontener na dane:***docker create -v /dbdata --name dbstore training/postgres /bin/true***

Wykorzystaj go dla swojej aplikacji:***docker run -d --volumes-from dbstore --name db1 training/postgres***

Może to być zrobione wielokrotnie:***docker run -d --volumes-from dbstore --name db2 training/postgres***

Ćwiczenie – Docker Build

Stwórz katalog zad w katalogu domowym użytkownika, a następnie przejdź do niego i stwórz tam pliki (nadaj im prawa do wykonywania):
„Dockerfile" bez rozszerzenia, z treścią:

```
FROM ubuntu:latest
RUN apt-get update -y
RUN apt-get install -y python python-dev python-distribute python-pip
RUN pip install flask
COPY . /app
WORKDIR /app
ENTRYPOINT ["python"]
CMD ["app.py"]
```

Komenda **From** określa nam obraz wyjściowy dla naszego testowego, następnie wykonujemy komendę **update** za pomocą apt-get i instalujemy pyton z wszystkimi potrzebnymi zależnościami. Dalej następuje przekopiowanie do folderu app, ustawienie katalogu roboczego i wykonanie komendy **cmd** mającej na celu uruchomienie aplikacji app.py
app.py z treścią (plik python formatowany tabulatorami):

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
        return 'Flask Dockerized'

if __name__ == '__main__':
        app.run(debug=True,host='0.0.0.0')
```

Zbuduj całość poleceniem:
docker build -t flaskapp:latest .
Nie zapomnij o kropce na końcu linijki!

Stwórz kontener na bazie wykreaowanego obrazu:
docker run –p 5000:5000 --name myfirstapp flaskapp

Sprawdź poprawność wykonania w przeglądarce na hoście gospodarza:

http://YOUR_HOST_IP:5000, powinien ukazać się napis:

Flask Dockerized

***

Zmodyfikuj dowolnie plik app.py. Np: Rozszerz funkcję hello_world o możliwość renderowania dokumentu html (index.html) na którym może wyświetlona zostać dowolna grafika. W tym celu należy skorzystać dodatkowo z importu render_template. Zwróć render_template('index.html', url=nazwa.gif) zamiast tekstu w funkcji hello_world. W pliku index.html zawrzyj podstawową strukturę html dla strony oraz tag do obrazu „img „albo jako link obrazkowy do wgranej na serwer grafiki.
Zweryfikuj czy nowa wersja obrazu zbudowana za pomocą :
docker build -t flaskapp:2.0 .
zawiera wprowadzone zmiany tworząc dla niej nowy kontener:
docker run -p 5000:5000 --name myfirstapp2 flaskapp:2.0

Ćwiczenie (Docker Swarm):

Za pomocą utworzonych maszyn wirtualnych oraz narzędzia docker swarm zbuduj farmę kontenerów, której zadaniem będzie wykonywanie zadanych czynności na systemie bazowym alpine. Na farmę muszą się składać minimum dwa węzły. Serwisy działające w farmie powinny dać się replikować.

Za pomocą programu putty i protokołu ssh zaloguj się do maszyny oznaczonej „Dockerlab" następnie wydaj polecenie: ***sudo docker swarm init --advertise-addr*** <YOURHOSTIP>

W odpowiedzi powinien zostać wygenerowany token, dzięki któremu można dołączyć dodatkowe węzły do farmy:

```
lab@ubuntu:~$ sudo docker swarm init --advertise-addr 192.168.0.158
Swarm initialized: current node (kxiujdka7x6aq1j6rdxagassh) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3a51efr0rlf0zniluso8qq7kvimjninq4fowfbedezwypj5d6e-4ejnreqbmq5xov1lz5lpidz8a 192.168.0.158:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Jeśli informacja ta uległa by zgubieniu można uzyskać ją ponownie za pomocą: ***sudo docker swarm join-token worker***
Wygenerowany token można teraz skopiować i wkleić na maszynie oznaczonej jako „Dockerlab_1"

```
lab@ubuntu:~$ sudo docker swarm join --token SWMTKN-1-3a51efr0rlf0zniluso8qq7kvi
mjninq4fowfbedezwypj5d6e-4ejnreqbmq5xov1lz5lpidz8a 192.168.0.158:2377
[sudo] password for lab:
This node joined a swarm as a worker.
```

Po powrocie na maszynę „Dockerlab" i wydaniu komendy: ***sudo docker node ls*** można zaobserwować iż farma posiada teraz 2 hosty:

```
lab@ubuntu:~$ sudo docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
kxiujdka7x6aq1j6rdxagassh *   ubuntu              Ready               Active              Leader
segyptq9ju0ov6rlt9to6pgvk     ubuntu              Ready               Active
```

Wydając na każdej z maszyn polecenie: ***sudo docker info*** możemy sprawdzić jaką rolę w farmie odgrywa każdy ze stworzonych węzłów.

Spróbujmy teraz utworzyć serwis Dockera o nazwie lab, którego zadaniem będzie wywoływanie co określoną jednostkę czasu w pętli drukowania komunikatu:
sudo docker service create --replicas 1 --name lab alpine /bin/sh -c "while (true); do echo Hello from Docker; sleep 2; done"

```
lab@ubuntu:~$ sudo docker service create --replicas 1 --name lab alpine /bin/sh -c "while (true); do echo Hello from Docker; sleep 2; done"
xttxxpt53mq051thnulnwbw95
```

Podejrzyjmy czy rzeczywiście udało się osiągnąć zamierzony efekt i status serwisu jest poprawny. Aby to osiągnąć należy wydać komendy: ***sudo docker service ls:***

```
lab@ubuntu:~$ sudo docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
xttxxpt53mq0        lab                 replicated          1/1                 alpine:latest
```

oraz ***sudo docker service ps lab***:

```
lab@ubuntu:~$ sudo docker service ps lab
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE           ERROR
ij3ip0y6z1db        lab.1               alpine:latest       ubuntu              Running             Running 4 minutes ago
```

lub ***sudo docker service inspect –pretty lab***:

```
lab@ubuntu:~$ sudo docker service inspect --pretty lab

ID:             xttxxpt53mq05lthnulnwbw95
Name:           lab
Service Mode:   Replicated
 Replicas:      1
Placement:
UpdateConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Update order:      stop-first
RollbackConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Rollback order:    stop-first
ContainerSpec:
 Image:         alpine:latest@sha256:1072e499f3f655a032e88542330cf75b02e7bdf673278f701d7ba61629ee3ebe
 Args:          /bin/sh -c while (true); do echo Hello from Docker; sleep 2; done
Resources:
Endpoint Mode:  vip
```

Zweryfikujmy czy kontener działa tylko w jednej instancji na jednym nodzie. Można do tego celu wykorzystać polecenie ***sudo docker node ls*** by przypomnieć sobie ID kontenerów, a następnie wykonać weryfikacje poleceniem ***sudo docker node ps id_noda***

```
lab@ubuntu:~$ sudo docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
kxiujdka7x6aq1j6rdxagassh *   ubuntu              Ready               Active              Leader
segyptq9ju0ov6rlt9to6pgvk     ubuntu              Ready               Active
lab@ubuntu:~$ sudo docker node ps segyptq9ju0ov6rlt9to6pgvk
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE
lab@ubuntu:~$ sudo docker node ps kxiujdka7x6aq1j6rdxagassh
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE
ij3ip0y6z1db        lab.1               alpine:latest       ubuntu              Running             Running 11 minutes
```

Zmien skalowanie na 5 za pomoca: ***sudo docker service scale lab=5***

```
lab@ubuntu:~$ sudo docker service scale lab=5
lab scaled to 5
```

Sprawdź czy serwis jest przeskalowany na 5 instancji:

```
lab@ubuntu:~$ sudo docker service ps lab
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE            ERROR               PORTS
ij3ip0y6z1db        lab.1               alpine:latest       ubuntu              Running             Running 20 minutes ago
sb098ucxiwsk        lab.2               alpine:latest       ubuntu              Running             Running 2 minutes ago
pmndkwnjw2k2        lab.3               alpine:latest       ubuntu              Running             Running 2 minutes ago
exnkuww6a54f        lab.4               alpine:latest       ubuntu              Running             Running 2 minutes ago
kkf0roic1pyc        lab.5               alpine:latest       ubuntu              Running             Running 2 minutes ago
```

Zweryfikuj poleceniem sudo docker node ps ID_node dla obu maszyn jak Docker rozłożył obciążenie:

```
lab@ubuntu:~$ sudo docker node ps kxiujdka7x6aq1j6rdxagassh
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE            ERROR               PORTS
ij3ip0y6z1db        lab.1               alpine:latest       ubuntu              Running             Running 22 minutes ago
exnkuww6a54f        lab.4               alpine:latest       ubuntu              Running             Running 3 minutes ago
lab@ubuntu:~$ sudo docker node ps segyptq9ju0ov6rlt9to6pgvk
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE            ERROR               PORTS
sb098ucxiwsk        lab.2               alpine:latest       ubuntu              Running             Running 3 minutes ago
pmndkwnjw2k2        lab.3               alpine:latest       ubuntu              Running             Running 3 minutes ago
kkf0roic1pyc        lab.5               alpine:latest       ubuntu              Running             Running 3 minutes ago
```

Usuń zreplikowany serwis za pomocą sudo docker service rm lab, zweryfikuj, czy wszystkie instancje na każdym z węzłów zostały usunięte poprawnie:

```
lab@ubuntu:~$ sudo docker service rm lab
lab
lab@ubuntu:~$ sudo docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
kxiujdka7x6aq1j6rdxagassh *   ubuntu              Ready               Active              Leader
segyptq9ju0ov6rlt9to6pgvk     ubuntu              Ready               Active
lab@ubuntu:~$ sudo docker node ps segyptq9ju0ov6rlt9to6pgvk
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE       ERROR               PORTS
lab@ubuntu:~$ sudo docker node ps kxiujdka7x6aq1j6rdxagassh
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRENT STATE       ERROR               PORTS
```

Przejdź na maszynę wirtualną „Dockerlab_1” i wydaj polecenie ***sudo docker swarm leave*** w celu wyjścia z farmy:

```
lab@ubuntu:~$ sudo docker swarm leave
[sudo] password for lab:
Node left the swarm.
```

Docker Compose

Celem tego zadania jest stworzenie infrastruktury składającej się z:

- serwera MySQL
- aplikacji WordPress

Na początek należy pobrać brakujące obrazy czyli w naszym przypadku obraz WordPress.

docker pull wordpress:latest

Aby móc zarządzać całą naszą infrastrukturą naraz skorzystamy z dodatkowego pakietu docker-compose . Instalacja poleceniem:

sudo apt-get install pip
sudo pip install docker-compose

Po wykonaniu powyższych poleceń może być potrzebne ponowne uruchomienie terminala. Następnie utwórzmy nowy katalog:

mkdir wordpress

W utworzonym katalogu tworzymy plik o nazwie **docker-compose.yml**. O następującej zawartości:

```
version: '3'
services:
  db:
    image: mysql:latest
    volumes:
    - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: haslo
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
```

```
    depends_on:
     - db
    image: wordpress:latest
    ports:
     - "80:80"
    restart: always
    environment:
     WORDPRESS_DB_HOST: db:3306
     WORDPRESS_DB_USER: wordpress
     WORDPRESS_DB_PASSWORD: wordpress
 volumes:
   db_data:
```

Powyższy plik w formacie YAML opisuje naszą infrastrukturę. Definiujemy w nim dwa kontenery: db oraz wordpress. Dla każdego z nich określamy obraz z którego ma być stworzony - pole *image* .

Dla kontenera MySQL dodatkowo ustawiamy szereg zmiennych środowiskowych. Dzięki temu utworzona zostanie baza o nazwie *wordpress* oraz użytkownik *wordpress* z hasłem *wordpress* . Pole *restart* określa kiedy kontener może zostać uruchomiony ponownie wartość. Dzięki dodaniu wolumenu *db_data* i zamontowaniu go w katalogu roboczym nie utracimy bazy danych po wyłączeniu kontenera.

Kontener WordPress jest zależny od kontenera zawierającego bazę danych. Mówi o tym linia *depends_on* . Dzięki takie konfiguracji kontener *wordpres* zostanie uruchomiony dopiero po poprawnym uruchomieniu kontenera *db*. Za pomocą pola *ports* konfigurujemy mapowanie portów kontenera. W poprzednim zadaniu to samo uzyskaliśmy za pomocą flagi -p 3066:3066. Podobnie jak w przypadku kontenera *db* należy ustawić kilka zmiennych środowiskowych.

Dokładny opis poszczególnych zmiennych można znaleźć tutaj:
https://hub.docker.com/_/wordpress/

Aby uruchomić tak skonfigurowane środowisko możemy uruchomić wydając polecenie:

docker-compose up

Polecenie należy uruchomić w katalogu zawierającym plik *docker-compose.yml*.

Teraz możemy za pomocą przeglądarki internetowej wejść pod adres *localhost* i skonfigurować WordPressa.