Docker

1. Co to jest Docker i do czego służy?

Zapoznaj się z artykułem kliknij TU

2. Instalacja Dockera i pierwsze uruchomienie

Cel: Sprawdzenie działania Dockera i uruchomienie pierwszego kontenera.

1. Sprawdź instalację Dockera:
    - W terminalu wpisz:

```
docker --version
```

    - Jeśli Docker jest poprawnie zainstalowany, zobaczysz wersję Dockera.
    - Jeśli nie jest zainstalowany, przeprowadź ten proces zgodnie z instrukcją na stronie kliknij TU
2. Uruchom pierwszy kontener:
    - W terminalu wpisz:

```
docker run hello-world
```

Opis: To polecenie uruchamia kontener z oficjalnym obrazem `hello-world`, który wyświetli prosty komunikat potwierdzający, że Docker działa poprawnie.

3. Wyświetl listę kontenerów:
    - Aby zobaczyć uruchomione kontenery, wpisz:

```
docker ps
```

    - Aby zobaczyć również zatrzymane kontenery, użyj:

```
docker ps -a
```

2. Zarządzanie obrazami Dockera

Cel: Nauka zarządzania obrazami Dockera, ich pobierania i usuwania.

1. Pobierz oficjalny obraz Python:
    - Pobierz obraz Pythona z Docker Hub:

```
docker pull python:3.9
```

2. Wyświetl listę obrazów:
    - Sprawdź, czy obraz został pobrany:

```
docker images
```

3. Tworzenie i uruchamianie kontenerów

Cel: Nauka uruchamiania kontenerów i interakcji z nimi.
Opis komendy: https://docs.docker.com/reference/cli/docker/container/run/

1. Uruchom interaktywny kontener z Pythonem:
    - Uruchom kontener w trybie interaktywnym:

```
docker run -it python:3.9
```

    - Po uruchomieniu zobaczysz interaktywną konsolę Pythona. Wpisz prosty kod:

```
print("Witaj w Dockerze!")
```

    - Aby opuścić konsolę i zatrzymać kontener, wpisz:

```
exit()
```

2. Uruchom kontener w tle:
    - Uruchom kontener z Nginx w trybie odłączonym (w tle):

```
docker run -d -p 8080:80 nginx
```

Opis: Serwer Nginx będzie dostępny lokalnie pod adresem `http://localhost:8080`.

3. Zatrzymaj kontener:
    - Wyświetl listę uruchomionych kontenerów i zatrzymaj Nginx:

```
docker ps
docker stop ID_kontenera
```

4. Tworzenie własnego obrazu Dockera z użyciem Dockerfile

Cel: Nauka tworzenia własnych obrazów Dockera na podstawie pliku `Dockerfile`.

1. Stwórz nowy folder projektu:
    - W terminalu utwórz folder w swoim katalogu na dysku D katalog o nazwie `docker_python_app` i przejdź do niego:
      Żeby zmienić dysk w CMD/PS wpisz “D:”

```
mkdir docker_python_app
cd docker_python_app
```

2. Stwórz plik aplikacji Python:
    - Utwórz plik `app.py` i dodaj do niego poniższy kod:

```
print("Witaj! To aplikacja działająca w kontenerze Docker.")
```

3. Stwórz plik `Dockerfile`:
    - W folderze projektu utwórz plik `Dockerfile` (bez żadnego rozszerzenia!) z następującą zawartością:
      Tip: rozszerzenie można usunąć poleceniem mv jak na linuxie

```
FROM python:3.9
COPY app.py /app/app.py
WORKDIR /app
CMD ["python", "app.py"]
```

Opis:

- `FROM` – bazowy obraz, tutaj Python 3.9.
- `COPY` – kopiuje plik `app.py` do katalogu `/app` w obrazie.
- `WORKDIR` – ustawia domyślny katalog roboczy na `/app`.
- `CMD` – określa polecenie, które ma zostać uruchomione po uruchomieniu kontenera

4. Zbuduj obraz:
    - Zbuduj obraz na podstawie `Dockerfile`:

```
docker build -t python_app .
```

5. Uruchom obraz jako kontener:
    - Uruchom kontener z utworzonego obrazu:

```
docker run python_app
```

Wynik: Powinieneś zobaczyć komunikat `Witaj! To aplikacja działająca w kontenerze Docker.`

5. Sieci w Dockerze

Cel: Nauka tworzenia i zarządzania sieciami Dockera oraz komunikacji między kontenerami.

1. Utwórz sieć Docker:
    a. Utwórz nową sieć o nazwie `moja_siec`:

```
docker network create moja_siec
```

2. Uruchom kontener z serwerem bazy danych (MySQL):
    - Uruchom kontener z MySQL i podłącz go do sieci:

```
docker run --name moj_mysql --network moja_siec -e MYSQL_ROOT_PASSWORD=haslo123
```

```
-e MYSQL_DATABASE=test_db -d mysql:5.7
```

Opis:

- `--network moja_siec` – podłącza kontener do sieci `moja_siec`.
- `-e` – ustawia zmienne środowiskowe, np. hasło do bazy danych i nazwę bazy.

3. Uruchom kontener z klientem MySQL:
    - Uruchom drugi kontener i połącz się z bazą danych:

```
docker run -it --network moja_siec mysql:5.7 mysql -h moj_mysql -uroot
-phaslo123 test_db
```

    - Po połączeniu, wykonaj w kliencie SQL prostą operację:

```
CREATE TABLE test_table (id INT AUTO_INCREMENT PRIMARY KEY, name
VARCHAR(50));
INSERT INTO test_table (name) VALUES ('Docker');
SELECT * FROM test_table;
```

6. docker-compose

Cel: Zapisanie konfiguracji kontenerów w pliku zamiast wielokrotnego klepania komend

1. W katalogu projektu stwórz plik `docker-compose.yml`

```
services:
  nazwa-uslugi:
    build:
      context: .
      dockerfile: Dockerfile
```

Opis:

- `services:` w tym kluczu wpisujemy kolejne klucze naszych usług
- `nazwa-uslugi:` identyfikator usługi wykorzystywany w pozostałych komendach docker-compose
- `build:` konfiguracja budowania obrazu
- `context:` ścieżka do katalogu, z którego pliki używane są przy budowaniu obrazu
- `dockerfile:` ścieżka do pliku Dockerfile

2. Zmień `Dockerfile`

```
FROM python:3.9
COPY app.py /app/app.py
WORKDIR /app
CMD ["tail", "-f", "/dev/null"]
```

Wyjaśnienie: wprowadzenie takiego polecenia do CMD pozwala nam utrzymać kontener przy życiu do momentu jego ręcznego wyłączenia

3. Uruchomienie kontenera

```
docker-compose up -d
```

Wyjaśnienie: Użycie flagi -d (“detached”) pozwala nam wrócić do terminala po wystartowaniu usług

4. Upewnij się, że kontener działa

```
docker-compose ps
```

Wyjaśnienie: Wyświetla **stan** wszystkich usług zdefiniowanych w `docker-compose.yml`

5. Zatrzymaj kontener

```
docker-compose stop
```

Wyjaśnienie: Zatrzymuje wszystkie usługi zdefiniowane w `docker-compose.yml`. Aby zatrzymać tylko wybrane, dodajemy ich nazwy po “stop”.

7. Praca z danymi w Dockerze

Cel: Nauka korzystania z wolumenów Dockera w celu współdzielenia danych między kontenerem a systemem hosta.

1. Zmień plik `docker-compose.yml`

```
services:
  nazwa-uslugi:
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - .:/app
```

Wyjaśnienie: Klucz volumes pozwala na mapowanie lokalizacji na komputerze hosta do ścieżek na kontenerze - tutaj katalog, w którym znajduje się `docker-compose.yml` do `/app` w kontenerze

2. Uruchom kontener

```
docker-compose up -d
```

3. Wejdź do kontenera i odpal w nim `ls`

```
docker-compose exec nazwa-uslugi bash
```

Wyjaśnienie: docker-compose exec uruchamia na wskazanym kontenerze podane polecenie, w tym przypadku otwiera nam terminal `bash`

4. Stwórz nowy plik z systemu hosta:
    - W folderze lokalnym utwórz plik `hello.py` z następującą treścią:

```
print("To jest plik współdzielony!")
```

5. Uruchom plik z kontenera:

```
python hello.py
```

Wynik: Powinieneś zobaczyć komunikat `To jest plik współdzielony!`.

8. Sieci w Dockerze - wersja z docker-compose

Cel: Demonstracja zalet docker-compose na przykładzie komunikacji między usługami/kontenerami

1. Zmień plik `docker-compose.yml`

```
services:
  serwer-mysql:
    image: "mysql:5.7"
    environment:
      - MYSQL_DATABASE=test_db
      - MYSQL_ROOT_PASSWORD=haslo123
    ports:
      - "3037:3036"

  klient-python:
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - .:/app
```

Opis:

- `image:` analogicznie do `FROM` z `Dockerfile` przyjmuje obraz z dockerhuba jako bazowy (Nazwa:tag)
- `environment:` tu wprowadzamy zmienne środowiskowe
- `ports:` mapowanie portów - z lewej strony port “dla hosta”, z prawej strony port używany przez usługę

Uwaga: wszystkie usługi zdefiniowane w jednym pliku `docker-compose.yml` automatycznie przynależą do tej samej sieci Dockera.

2. Zmień plik `Dockerfile`

```
FROM python:3.9
COPY app.py /app/app.py
RUN pip install mysql-connector-python
```

```
WORKDIR /app
CMD ["tail", "-f", "/dev/null"]
```

Wyjaśnienie: Polecenie `RUN` pozwala nam wywoływać komendy w trakcie budowania obrazu

3. Zmień plik `app.py`

```
import mysql.connector

db = mysql.connector.connect(user='root', password='haslo123',
host='serwer-mysql', database='test_db')

cursor = db.cursor()
cursor.execute("CREATE TABLE test_table (id INT AUTO_INCREMENT PRIMARY
KEY, name VARCHAR(50));")
cursor.execute("INSERT INTO test_table (name) VALUES ('Docker');")
db.commit()
cursor.close()
db.close()
```

4. Uruchom kontenery

```
docker-compose up -d
```

5. Wejdź do kontenera `klient-python` i uruchom w nim plik `app.py`

```
docker-compose exec klient-python bash

# po wejściu do kontenera
python app.py
```

6. Wejdź do kontenera `serwer-mysql` i sprawdź czy tabela się utworzyła

```
docker-compose exec serwer-mysql bash

# po wejściu do kontenera
mysql -uroot -phaslo123

# uruchamiamy zapytania SQL sprawdzajace czy tabela istnieje i zawiera
dane
```