Laboratorium 6

Docker compose.

1. Podstawowy obraz
    a. Stwórz katalog first, a w nim plik docker-compose.yaml
        i. Nazwa serwisu – app
        ii. Obraz -debian:12
        iii. Nazwa kontenera – first_deb
    b. Wykonaj polecenie docker compose config. Sprawdź output.
    c. Wykonaj polecenie docker compose up -d.

2. Dyrektywa **build.**
    a. Stwórz strukturę katalogów second/app a w nim plik Dockerfile.
        i. Arg to ubu_ver=22.04
        ii. Obraz bazowy to ubuntu z tagiem podanym w ARG
        iii. Workdir to /app
        iv. Zmienna środowiskowa city ustawiona na kraków
        v. Skopiuj do obrazu, do katalogu /app skrypt.sh (wyświetlający tekst hello $city)
        vi. Jako entrypoint ustaw start skryptu.
    b. Stwórz plik docker-compose.yaml
        i. Nazwa serwisu app
        ii. Obraz zbudowany na podstawie stworzonego Dockerfile
            1. Dyrektywa

```
build:
  context: ./app
   args:
     ubu_ver:  ’20.04’
```

        iii. Nazwa obrazu (dyrektywa image na wysokości build) app:1
    c. Uruchom config.
    d. Uruchom serwis i sprawdź działanie.

3. Zmienne środowiskowe – **environment:**
    a. Zmodyfikuj poprzedni projekt dodając do pliku zmienne środowiskowe (wysokość dyrektywy build):

```
environment:
    - version: 1
    - project_type: test
```

    b. Uruchom config
    c. Uruchom serwis – za pomocą polecenia docker exec NAZWA env| grep version sprawdź poprawność wprowadzonych zmiennych środowiskowych.
    d. Zmodyfikuj poprzedni projekt, tak by wykorzystać plik o nazwie .env. Stwórz plik i zdefiniuj w nim zmienne – a w pliku .yaml odnieś się do zmiennych poprzez ${nazwa_zmiennej}
    e. Przetestuj przy użyciu polecenia docker compose config

f. Zmodyfikuj poprzedni projekt, tak by w pliku .yaml zamiast zmiennych środowiskowych wprowadzić dyrektywę **env_file:** (wysokość image) jako wartość klucza podaj nazwę pliku, w którym umieściłeś definicję zmiennych.
g. Przetestuj przy uzyciu polecenia docker compose config, czy system wczytał wartości zmiennych z pliku.

4. **Command**: (pozwala na definiowanie własnych poleceń w ramach serwisu)
   a. Zmodyfikuj poprzedni plik docker compose dodając w nim kolejny serwis:
      i. Nazwa deb:
      ii. Obraz: debian:12
      iii. Command: sleep inf

```
command:
  - sleep
  - inf
```

   b. Uruchom i sprawdź działanie.

5. **Limits:** (wprowadzanie limitów do działanie kontenera)
   a. Zmodyfikuj plik z poprzedniego zadania tak, aby wprowadzić limity użycia zasobów dla serwisu deb:
      i. Na wysokości image i command wprowadź:

```
deploy:
  resources:
     limits:
        memory: 64M
```

   b. Uruchom config
   c. Uruchom serwis.
   d. Za pomocą polecenia docker stats sprawdź czy zostały wprowadzone wymagane limity.

6. Montowanie zasobów: **volumes:**
   a. Zmodyfikuj parametry serwisu deb, dodając do nich montowanie zasobów:
      i. Dyrektywa volumes: (wysokość command i image)
         1. Zamontuj katalog ~/test:/data1
         2. Zamontuj volumen voltest:/voldata

```
volumes:
    - „/home/name/test:/data1”
    - „voltest:/voldata”
```

         3. Dokonaj rejestracji wolumenów na poziomie klucza service: tak by były dostępne dla innych serwisów.

```
volumes:
   voltest:
```

4. Uruchom serwis w trybie interaktywnym i sprawdź podmontowanie zasobów.
5. Zmodyfikuj plik docker compose, tak by użyć innej składni montowania zasobów (wysokość image, command)

```
volumes:
   - type:bind
     source:/home/name/test
     target:/data1
   - type: volume
     source: voltest
     target:/voldata
```

7. Sieci: **networks:**
   a. Zmodyfikuj plik .yaml z poprzedniego zadania wzbogacając go o możliwość komunikacji sieciowej.
      i. Do serwisu deb dodaj dyrektywę **networks:** (wysokość volumes, commands)

```
networks:
  appnets:
    ipv4_address: 192.168.20.20
```

      ii. Stwórz nowy serwis o nazwie web:
         1. Obraz: - nginx:latest
         2. Przekierowanie portów **ports:** (wysoskość image)9999:80

```
ports:
 - 9999:80
```

         3. Wystawienie portu 80 – dyrektywa **expose:**

```
expose:
  - 80
```

      iii. Podepnij sieć analogicznie jak w przypadku serwisu deb, ale z adresem 192.168.20.30

```
networks:
  appnets:
    ipv4_address: 192.168.20.30
```

      iv. Zarejestruj sieć, tak by była ona dostępna dla wszystkich serwisów (wysokość service:)

```
networks:
  appnets:
```

```
ipam:
  driver: default
  config:
   - subnet: '192.168.20.0/24'
```

b. Dokonaj konfiguracji serwisów.
c. Uruchom kontenery
d. Przetestuj komunikację między nimi (możliwa jest komunikacja jedynie pomiędzy serwisami w ramach jednej sieci)

8. **Depends-on:** (jeśli serwis jest zależny od pozostałych)
   a. Zmodyfikuj plik .yaml w taki sposób, by w serwisie deb pojawił się klucz **depends_on** (wysokość image)

```
depends_on:
  - web
```

9. Uruchamianie wielu instancji na podstawie jednego pliku .yaml. Żeby uruchomić wiele instancji wystarczy w poleceniu docker compose up dodac flagę -p i podać oryginalną nazwę. Pamiętaj o możliwych konfliktach, np. przy przekierowaniu portów.

10. **Ciekawostka 😊 (uruchamianie kontenerów z interfejsem graficznym)**
    a. Pobierz obraz kali linux

```
docker pull kalilinux/kali-rolling
```

    b. Uruchamiamy z przekierowaniem portów.

```
docker run -itd --name kali-gui -p 5800:5900 kalilinux/kali-rolling
```

    c. Wchodzimy do kontenera

```
docker exec -it kali-gui /bin/bash
```

    d. Instalujemy server vnc.

```
apt update

apt install kali-desktop-xfce tightvncserver
```

    e. Ustawiamy użytkownika za pomocą zmiennej systemowej.

```
export USER=root
```

f. Tworzymy hasło

```
vncpasswd
```

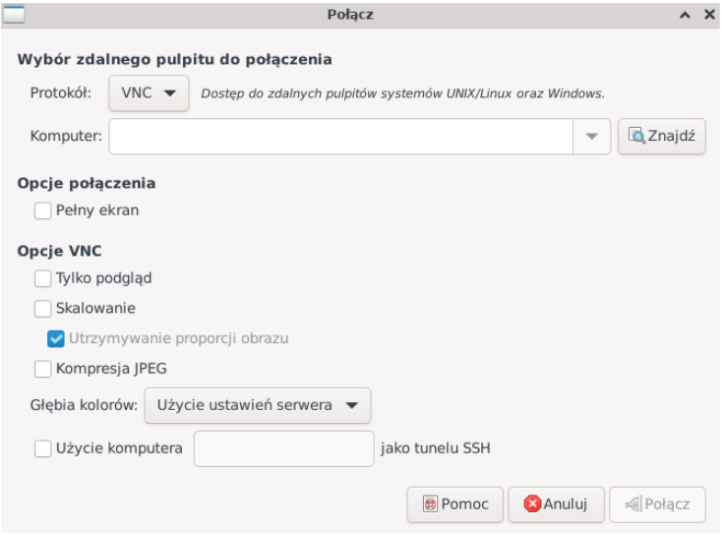
g. Udostępniamy kontener

```
tightvncserver :0 -geometry 1366x768 -depth 24
```

h. Na maszynie wirtualnej instalujemy program kliencki – vinagre

```
sudo apt update && sudo apt -y install vinagre
```

i. Łączymy się.



W polu komputer localhost:5800