Docker virtualizáció

A következőkben megnézzük hogyan lehet használni a gyakorlatban a népszerű Docker virtualizációt.

Kérem, hogy mindenki lépjen be a http://docker.iit.uni-miskolc.hu/ oldalon. Majd lépjen be ugyanitt a megjelenő start gombra kattintva.

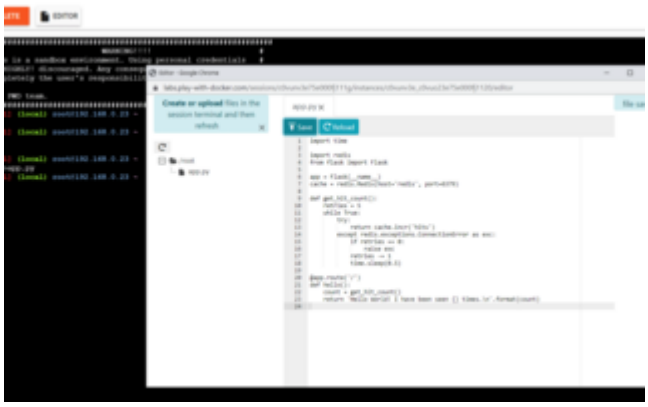
A megjelenő képen bal oldalon a “+ Add new instance” gomb megnyomása után a következő képet fogjuk látni:



Hozzunk létre egy app.py nevő állományt, a következtő parancs futtatásával, és nyomjuk meg a Ctrl+d a befejezéshez:

```
cat>app.py
```

A delete gomb melletti Editor feliratú gomb megnyomásával egy új ablakban megjelenik a fájl editor. Ebben nyissuk meg a most létrehozott **app.py**-t.



Másoljuk be az editorba az alábbi kódot:

```
import time

import redis
from flask import Flask

app = Flask(__name__)
cache = redis.Redis(host='redis', port=6379)
```

```
def get_hit_count():
    retries = 5
    while True:
        try:
            return cache.incr('hits')
        except redis.exceptions.ConnectionError as exc:
            if retries == 0:
                raise exc
            retries -= 1
            time.sleep(0.5)

@app.route('/')
def hello():
    count = get_hit_count()
    return 'Hello World! I have been seen {} times.\n'.format(count)
```

Ez a Python program az egyik legegyszerűbb Flask elnevezésű webkiszolgálót/keretrendszert mutatja. A fenti kódrészlet egy számlálót valósít meg, ami a **redis** nevű cache rendszer segítségével számolja a látogatókat.

Hozzunk létre egy **requirements.txt** nevű állományt a korábbi módon:

```
cat>requirements.txt
```

Majd az editor segítségével másoljuk bele az alábbi sorokat:

```
flask
redis
```

Ezzel definiáljuk, hogy az alkalmazásunknak mik a függőségei. Ebben az esetben ezek a **flask framework** és a **redis cache**. Ez azért kell, mert a konténer üres konfigurációval indul, és a requirements.txt használatával fogjuk telepíteni a függőségeket. Azaz kézzel nem telepítünk semmit, csak szabványos módon.

Dockerfile létrehozása

Hozzunk létre egy **Dockerfile** elnevezésű állományt a következő tartalommal a szokásos módon:

```
FROM python:3.7-alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
EXPOSE 5000
COPY . .
CMD ["flask", "run"]
```

Itt meg kell állnunk egy kis magyarázatra. A Dockerfile feladata az, hogy lépésről lépésre definiálja a virtuális gép létrehozását.

Soronként a következőket definiáljuk:

- Hozzon létre egy kiinduló virtuális gépet (image) a python 3.7-es támogatással és a **alpine** nevű linux kernellel.
- a munkakönyvtárunk a /code lesz.
- Állítsunk be két környezeti változót, ami a flask-nak szükséges a kiszolgáláshoz.
- Telepítsük a gcc-t és más függőségeket. (ez azért kell, mert a Python sok csomagot c/c++ forrás állományokból fordít)
- másoljuk be a requirements.txt-t a munkakönyvtárba (ez azért kell, mert a konténernek saját fájlrendszere van, a Dockerfile mellett lévő állományokat nem tudja közvetlenül olvasni.)
- EXPOSE parancs tcp portot nyit meg kifelé (jelen esetben az 5000-est)
- mindent másoljuk be a munkakönyvtárba
- az utolsó sor a telepítés utáni indító parancsot definiálja, jelen esetben: “flask run”

Compose állomány létrehozása

Más leírásokban ennél a pontnál elindítják a virtuális gépet. Mi nem tesszük meg, hanem továbblépünk a **docker-compose** lehetőségeire, amivel rugalmasan tudunk több virtuális gépet egyszerre kezelni. Nem feltétlenül kell Dockerfile-t sem létrehozni, ha Interneten is elérhető szabványos konfigurációkat használunk.

Hozzuk létre a docker-compose.yml állományt a szokásos módon:

```
version: "3.3"
services:
  web:
    build: .
    ports:
      - "80:5000"
  redis:
    image: "redis:alpine"
```

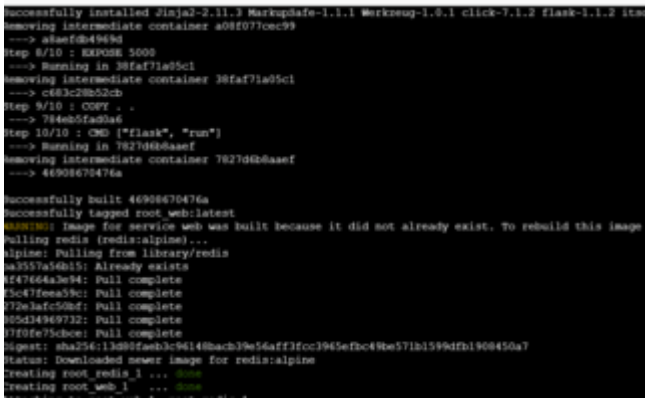
Ez az állomány szolgáltatásokban gondolkodik. Minden **service** egy különálló docker image, viszont a nevükre hivatkozva belsőleg elérik egymást. A fenti konfiguráció **web** elnevezésű szolgáltatását a **build: .** miatt a Dockerfile alapján hozzuk létre, viszont a másik **redis** szolgáltatást a szabványos “redis:alpine” konfiguráció alapján használjuk.

A **web** esetén a belsőleg kinyitott 5000-es portot láthatóvá tesszük a 80-as port-on.

Indítsuk el a következő parancsot és várjuk meg ameddig lefut:

```
docker-compose up
```

A következő képernyőhöz hasonlót kell látnunk (de 5000 helyett a 80-as portot adjuk meg), ha mindent jól állítottunk be előzőleg. Nyomjuk meg a nyíllal jelölt gombot.



Fejlesztési lehetőségek

Frissítsük többször a böngészőt, láthatjuk, hogy a látogatásszám dinamikusan frissül.

Ctrl + c segítségével megállíthatjuk a futtatást.

cseréljük le a docker-compose.yml tartalmát a következőre:

```
version: "3.3"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
    environment:
      FLASK_ENV: development
  redis:
    image: "redis:alpine"
```

A **volumes** beállítja, hogy a virtuális gépben belül levő könyvtár, jelen esetben a working directory a gazda rendszerhez legyen kötve, illetve kimásolva. Ha nem a gyökérbe szeretnénk mappelni, akkor pl: ./mycode:/code is megadható, de a mycode könyvtárnak az indítás előtt léteznie kell.

Indítsuk el újra a rendszert:

```
docker-compose up
```

Majd látható, hogy a konzolban developer módra kapcsoltunk. Módosítsuk az editor segítségével a app.py-t, mondjuk az utolsó függvényben a kiírás szövegét és frissítsük a böngészőt.

Docker compose parancsok

Futó virtuális gépek listázása:

```
docker-compose ps
```

Szolgáltatások log-jainak megtekintése:

```
docker-compose logs -t -f web
```

-t az időbélyeget, a -f folyamatos nyomonkövetést biztosít.

Egy adott instance milyen környezeti változókat használ?

```
docker-compose run web env
```

Hogyan állíthatjuk le a szolgáltatásokat?

```
docker-compose stop
```

Hogyan tudunk teljesen letörölni mindent leállítás után?

```
docker-compose down --volumes
```

Hogyan tudunk shellbe belépni egy konténeren belül?

```
docker-compose exec <containername> sh
```

Hogyan tudjuk újrafordítani a tárolót?

Általában a fejlesztés során a változások elmentődnek és a módosítások nem hajtódnak végre. Ilyenkor hasznos az alábbi parancs:

```
docker-compose build --no-cache
```

Ha WSL2-ben szeretnénk elindítani a docker-t akkor az alábbi sort másoljuk be egy indító szkript-be (p. run_docker.sh):

```
sudo /usr/bin/dockerd -H unix:// -H tcp://0.0.0.0:2375
```

Ilyenkor egy külön terminálba először a `./run_docker.sh-t` kell indítani.

Honnan tudok előre elkészített minta container-eket letölteni?

https://github.com/docker/awesome-compose

Feladat: a fenti linkről telepítsük az egyik megoldást.

From:
https://edu.iit.uni-miskolc.hu/ - **Institute of Information Science - University of Miskolc**

Permanent link:
https://edu.iit.uni-miskolc.hu/tanszek:oktatas:informatikai_rendszerek_epitese:docker_vitualizacio

Last update: **2026/04/24 13:12**