Dydaktyka Informatyki 17(2022)
ISSN 2083-3156; e-ISSN 2543-9847
DOI: 10.15584/di.2022.17.9
http://www.di.ur.edu.pl

Uniwersytet Rzeszowski

Jacek WOŁOSZYN[1], **Michał WOŁOSZYN**[2]

[1] *ORCID: 0000-0003-4340-9853. Dr inż., Uniwersytet Technologiczno-Humanistyczny w Radomiu, Wydział Informatyki i Matematyki, Katedra Informatyki, ul. Malczewskiego 20A; 26-600 Radom; e-mail: jacek.woloszyn@uthrad.pl*
[2] *Student, Goldsmiths, University of London,* 8 *Lewisham* Way, London SE14 6NW; *e-mail: mwolo001@gold.ac.uk;*
data złożenia tekstu do Redakcji DI: 4.02.2022; data wstępnej oceny artykułu: 11.02.2022

KONTENERYZACJA USŁUG DJANGO I POSTGRES Z WYKORZYSTANIEM DOCKER-COMPOSE

CONTAINERIZATION OF DJANGO AND POSTGRES SERVICES USING DOCKER-COMPOSE

Słowa kluczowe: konteneryzacja, virtualizacja, Docker, docker-compose.
Keywords: contener, virtualization, Docker, docker-compose.

Streszczenie

W artykule przedstawiono proces umieszczania aplikacji dostarczającej usługi sieciowe w kontenerach Dockera. Docker pozwala na skrócenie w znaczący sposób czasu utworzenia i utrzymania środowisk pracy. Umożliwia skalowanie infrastruktury aplikacji w rzeczywistym czasie i poprawia wykorzystanie zasobów poprzez obsługę wielu kontenerów. Zapewnia także budowanie aplikacji sterowanych zdarzeniami zorientowanych na usługi. Całość pozwala na skupienie się na celach biznesowych realizowanego celu. W części 1. przedstawiono zarys technologii opartej na Dockerze. Część 2. opisuje kolejne kroki przygotowania środowiska budowy pliku konfiguracyjnego i uruchomienia kontenera od utworzenia pliku Dockerfile, docker-compose.yml, poprzez konfigurację bazy danych i uruchomienia samej usługi. Część 3. skupia się na procesie instalacji frameworka Django. Część 4. zawiera szczegółowe informacje dotyczące szczegółów konfiguracji bazy danych.

Abstract

The article shows the process of placing applications that provide web services in Docker containers, which allows you to significantly reduce the time to set up and maintain a work environment. Docker enables real-time scaling of the application infrastructure and improves resource

utilization by supporting multiple containers. It also provides building of event-driven service-oriented applications. The whole thing allows you to focus on your business goals. The following chapters examine the Docker-based technology. They describe the steps of preparing the environment for building the configuration file and starting the container from creating the Dockerfile file, docker-compose.yml, through database configuration and starting the service itself.

Wstęp

Kontenery uruchamiają obrazy, które są tworzone poprzez połączenie obrazu podstawowego oraz kolejnych warstw na nim. Zawierają wszystko, co jest niezbędne do uruchomienia aplikacji. Znakomicie upraszczają proces zarządzania oprogramowaniem na serwerach, są skalowalne i w dużym stopniu automatyzują proces przenoszenia aplikacji działających klasycznie do kontenera. Uruchomienie ich wewnątrz pozwala na izolację parametrów od zewnętrznego środowiska, a tym samym dają możliwości uruchomienia kolejnych wersji tej samej aplikacji na fizycznej maszynie. Wykorzystanie proponowanego rozwiązania znakomicie umożliwia wykorzystanie zasobów fizycznych maszyny i ekonomiczne zarządzanie nimi.

Konteneryzacja

Jeszcze nie tak dawno większość aplikacji była wdrażana bezpośrednio na sprzęcie fizycznym bezpośrednio w systemie operacyjnym hosta. Ze względu na przestrzeń użytkownika środowisko wykonawcze było współdzielone między aplikacjami. Takie rozwiązanie działało doskonale do momentu, kiedy należało uruchomić równolegle dwie lub więcej aplikacji opartych na bibliotekach w różnych wersjach pozostających ze sobą w konflikcie.

Rozwiązania oparte na Dockerze pozwalają uniknąć tego rodzaju problemów. Niekompatybilność stosowanych bibliotek dla różnych wersji tego samego oprogramowania. Różnorodność w wersji bibliotek przy wdrożeniu na hoście produkcyjnym zdecydowanie utrudniała prawidłowe wdrożenie. Nawet przy zgodnych wersjach często trzeba sobie radzić z różnicami w konfiguracji w ogólnym środowisku, chociażby takimi jak różnice w sposobie nadawania uprawnień do plików, czy różnymi wersjami systemu operacyjnego. Wszystkie te problemy pojawiają się w momencie, kiedy wdrażany jest kod na hoście produkcyjnym. Powstaje zatem pytanie, czy programiści powinni wykonywać swoją pracę tylko w tych środowiskach, które odpowiadają serwerom produkcyjnym, czy też środowisko produkcyjne powinno być skonfigurowane do ustawień programisty. W świecie idealnym wszystko powinno być spójne od maszyny

programisty po serwery produkcyjne. Jednak w świecie rzeczywistym jest to dość trudne do osiągnięcia. Wymuszanie spójności na wielu platformach jest trudne, szczególnie gdy nad systemami pracuje niewielki zespół ludzi.

Z pomocą przychodzą usługi konteneryzacji, gdzie liderem jest Docker[1]. Programista może szybko umieścić swój kod w kontenerze, który sam zdefiniował został stworzony jako plik Dockera. Zespół inżynierów wdrożeniowców może szybko go wdrożyć zgodnie z instrukcjami programisty.

Rozwiązanie oparte na Dockerze korzysta z podstawowych funkcji jądra systemu operacyjnego, które umożliwiają ich realizację. Każdy kontener posiada swoją unikalną przestrzeń nazw, tak zwane bloki konstrukcyjne do kontenera, każdy z nich izoluje aplikacje od innych rozwiązań, to powoduje, że każdy kontener może posiadać własną numerację procesów, a nadrzędna przestrzeń nazw może widzieć podrzędne przestrzenie nazw i wpływać na nie. Aby umożliwić pracę w sieci w kontenerach tworzone są specjalne pary potencjalnych interfejsów w dwu różnych sieciowych przestrzeniach nazw, które umożliwiają komunikację pomiędzy sobą.

Budowa kontenerów

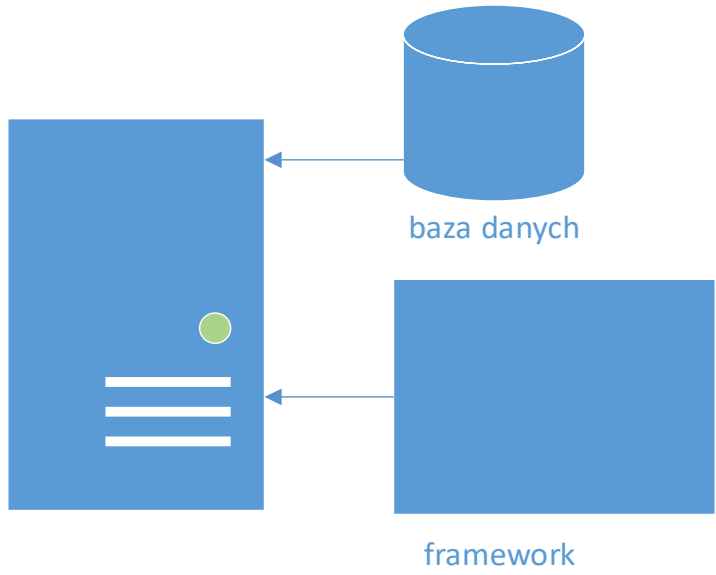
W klasycznym rozwiązaniu wdrożenie aplikacji na rzeczywistym hoście wyglądałoby w sposób następujący. Pierwszym z elementów, który należy zainstalować jest framework Django, który wymaga do funkcjonowania środowiska Python wraz z niezbędnymi bibliotekami. Do prawidłowej pracy frameworka niezbędna jest baza danych, w której będą zapisywane dane. Standardowo do mniejszych projektów wykorzystywana jest baza SQLite3. Jednak, aby zapewnić odpowiednią szybkość i bezpieczeństwo w tym przykładnie użyto bazy Postgres. Wymienione składniki będą funkcjonować w środowisku systemu operacyjnego Linux. Przykład tradycyjnego rozwiązania wdrożenia pokazuje rys. 1.

Rozwiązanie takie posiada jednak wszystkie wady, o których wcześniej wspomniano.

Aby wyeliminować wszystkie niedoskonałości takiej konfiguracji należy zastosować konteneryzację.

Dzięki zastosowaniu przestrzeni nazwy PID można wielokrotnie uruchamiać tę samą aplikację w różnych odizolowanych środowiskach w odmiennych konfiguracjach. Uruchamiając różne instancje tego samego programu na wielu kontenerach.

[1] J.S. Chelladhurai, V. Singh, P. Raj, *Docker dla praktyków*, Helion, Gliwice 2018; K. Cochrane, *Docker Coookbook Second Ediction*, Packt 2021.



Rys. 1. Klasyczne rozwiązanie wdrożenia aplikacji

Źródło: opracowanie własne.

Do utworzenia kontenera, w którym będą jednocześnie funkcjonować framework Django[2] i baza danych Postgres wykorzystano polecenie docker-compose. W pierwszej kolejności należy utworzyć plik Dockerfile[3], który definiuje zawartość obrazu aplikacji oraz go konfiguruje. Zawartość pliku Dockerfile pokazuje listing

W kolejnych wierszach listingu zlecono pobieranie Pythona[4] wersji 3, który jest nadrzędnym obrazem. Z kolei ustawiono zmienne środowiskowe definiujące katalog roboczy. W tym przypadku nosi nazwę code. W pliku requirements.txt przedstawionym na listingu wymieniono wszystkie niezbędne biblioteki, które są niezbędne do prawidłowego funkcjonowania projektu.

Polecenie

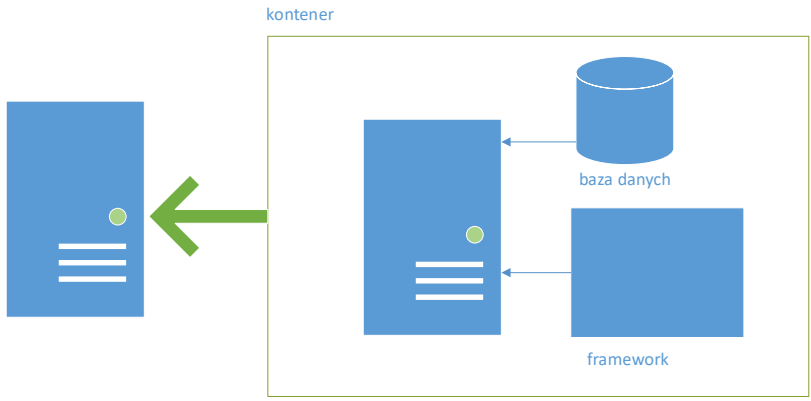
RUN pip install -r requirements.txt

instaluje biblioteki w środowisku.

[2] J.S. Chelladhurai, V. Singh, P. Raj, *Docker dla praktyków – Docker…,* dz. cyt.; Y. Hilpisch, *Derivatives, Analytics with Python*, Wiley 2015.

[3] R. Mckendrick, *Mastering Docker – Foutrh Ediction*, Packt 2021.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, É. Duchesnay, *12 Scikit-learn: Machine Learning in Python*, (Oct):2825−2830, 2011; M. Goodrich, R. Tamassia, M. Goldwasser, *Data Structures and Algorithms in Python*, Wiley 2013.



Rys. 2. Rozwiązanie wdrożenia z wykorzystaniem kontenera

Źródło: opracowanie własne.

```
Django>=3.0,<4.0
psycopg2>=2.8
django-bootstrap4
django-tastypie
djangorestframework
Markdown
pandas
PyPDF2
python-social-auth
Pillow
```

Listing 1. Zawartość pliku requirements.txt

Źródło: opracowanie własne.

```
# syntax=docker/dockerfile:1
FROM python:3
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1
WORKDIR /code
COPY requirements.txt /code/
RUN pip install -r requirements.txt
COPY . /code/
```

Listing 2. Zawartość pliku plik Dockerfile

Źródło: opracowanie własne.

Do utworzenia środowiska niezbędny jest jeszcze plik docker-compose.yml pokazany na listingu 3., który opisuje usługi tworzące aplikację. W tym przypadku będzie to katalog usług WWW, jaki jest wymagany do jego działania oraz baza danych w tym przypadku Postgres. W pliku zawarte są również informacje, w jaki sposób łączą się usługi ze sobą. Woluminy mogą wymagać zamontowania wewnątrz kontenerów. Zdefiniowane są również informacje o portach, które te usługi udostępniają. Volumes jest to zdefiniowany punkt dostępu dla hosta i kontenera. Jest jedynym wspólnym punktem dla obu systemów pozwalającym z zewnątrz dostać się do kontenera data/db:/var/lib/postgresql/data (rys. 3.).

Dla użytkownika zewnętrznego dostęp do usługi będzie możliwy poprzez port 8003, który jest zmapowany z portu 8000 kontenera (rys. 4.). W części db environment są zdefiniowane zmienne środowiskowe zawierające informacje dla baz danych o nazwie użytkownika o nazwie bazy danych jak hasło. Te same zmienne są wykorzystane w części web do połączenia z bazą. Ta część zawiera również komendę pozwalającą na uruchomienie serwera wirtualnego na porcie 8000. Należy jednak pamiętać że jest on mapowany na port 8003 dla użytkownika końcowego i z niego należy właśnie korzystać.

Host path	Container path
/root/d23docker/data/db	/var/lib/postgresql/data

Rys. 3. Połączone woluminy db

Źródło: opracowanie własne.

Container port	Protocol	Host IP	Host port
8000	tcp	0.0.0.0	8003
8000	tcp	::	8003

Rys 4. Mapowanie postów dla struktury web

Źródło: opracowanie własne.

Host path	Container path
/root/d23docker	/code

Rys 5. Mapowanie woluminów dla web

Źródło: opracowanie własne.

```
version: "3.9"

services:
db:
    image: postgres
    volumes:
```

```
      - ./data/db:/var/lib/postgresql/data
  environment:
      - POSTGRES_NAME=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    web:
      build: .
      command: python manage.py runserver 0.0.0.0:8000
      volumes:
        - .:/code
      ports:
        - "8003:8000"
      environment:
        - POSTGRES_NAME=postgres
        - POSTGRES_USER=postgres
        - POSTGRES_PASSWORD=postgres
  depends_on:
  - db
```

Listing 3. Plikdocker-compose.yml

Źródło: opracowanie własne.

Instalacja Django w kontenerze

W przygotowanym środowisku deweloperskim należy utworzyć wymagany projekt lub też skopiować struktury gotowego projektu już wcześniej utworzonego. Aby utworzyć nowy pusty projekt należy wydać polecenie:

```
sudodocker-compose run web django-admin startprojectnazwa_projektu
```

.

Zarówno tworzony nowy projekt lub też sklonowany gotowy posiada zapewne inne ustawienia do połączenia z bazą danych aniżeli utworzona z obrazu pliku baza. Dlatego kolejnym krokiem jest konfiguracja uruchomionej działającej bazy do oczekiwań projektu. Jak widać na listingu 4., baza danych jest gotowa do akceptowania przychodzących połączeń na porcie 5432. Nie wyświetla żadnych błędów ani problemów.

```
2022-01-28 14:29:55.319 UTC [1] LOG:  startingPostgreSQL 14.1 (Debian 14.1-1.pgdg110+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
```

```
    2022-01-28T14:29:55.320181707Z 2022-01-28 14:29:55.320 UTC [1]
LOG:  listening on IPv4 address "0.0.0.0", port 5432
    2022-01-28T14:29:55.320192262Z 2022-01-28 14:29:55.320 UTC [1]
LOG:  listening on IPv6 address "::", port 5432
    2022-01-28T14:29:55.320845591Z 2022-01-28 14:29:55.320 UTC [1]
LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
    2022-01-28T14:29:55.323399379Z  2022-01-28  14:29:55.323  UTC
[26] LOG:  database system was shut down at 2022-01-28 08:18:38
UTC
    2022-01-28T14:29:55.337609396Z 2022-01-28 14:29:55.337 UTC [1]
LOG:  database system is ready to accept connections
```

Listing 4. Uruchomiona baza danych

Źródło: opracowanie własne.

Konfigurowanie bazy Postgres

Dla przypadku sklonowanej gotowej aplikacji należy dodać zgodnie z ustawieniami nowego usera /użytkownika/ nową bazę, jak i nadać im odpowiednie uprawnienia. W analizowanym przykładzie należy zaimportować również zdumpowaną /archiwizowaną/ poprawną bazę danych do nowo uruchomionej struktury.

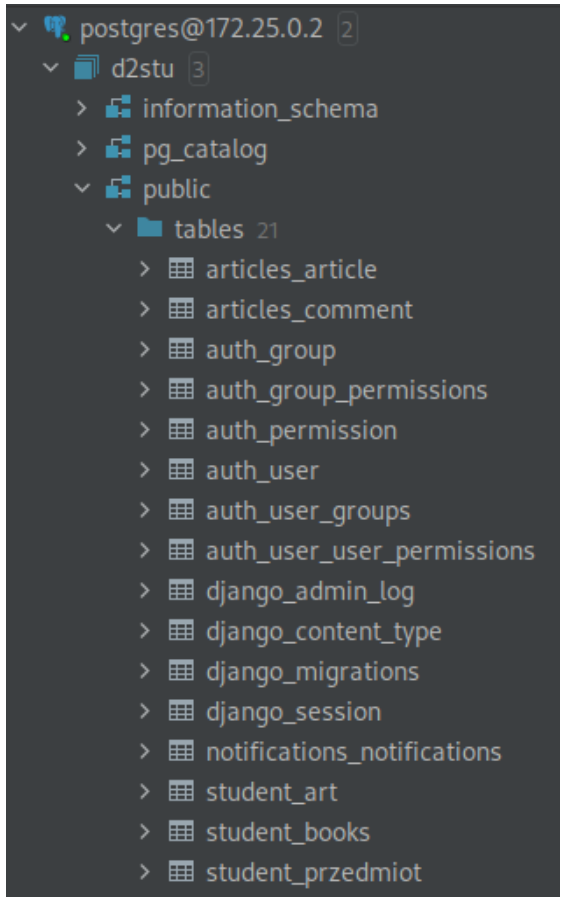
W kolejnych krokach zmieniamy użytkownika na Postgres. Za pomocą psql należy dodać nowego użytkownika i utworzyć bazę danych. Zmienić uprawnienia oraz zaimportować bazę danych. Opisany proces, jak i wynik tych działań dla importowanej bazy danych przedstawiony jest na listingu 5. i rys. 6. i 7.

```
    root@81d1dd0f58b8:/# supostgres
    postgres@81d1dd0f58b8:/$ psql
    psql (14.1 (Debian 14.1-1.pgdg110+1))
    Type "help" for help.

    postgres=# create user user with encrypted password 'pass';
    postgres=# create database d2stu;
    postgres=# grant all privileges on database d2stu to d2stu;
    postgres@f544530d539b:/$   psql   -U   d2stu   -d   d2stu   -f
/var/lib/postgresql/data/d2stu.sql
```

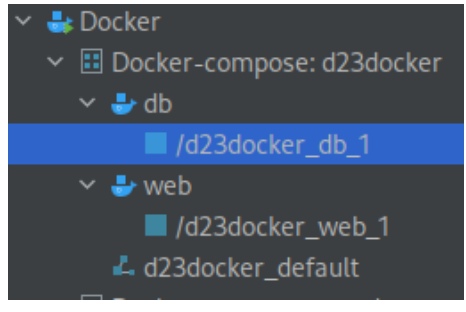
Listing 5. Proces importu bazy danych

Źródło: opracowanie własne.



Rys. 6. Struktura zaimportowanej bazy danych

Źródło: opracowanie własne.



Rys. 7. Wynik działania docker-composeup

Źródło: opracowanie własne.

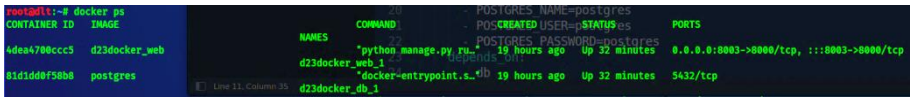
Aby umożliwić połączenie Django z nową bazą danych, w ustawieniach w pliku settings.py w sekcji database należy wprowadzić odpowiednie zmiany umożliwiające połączenie z zainstalowaną bazą. Przykład takich ustawień znajduje się w listingu 6.

```
DATABASES = {
    'default': {
'ENGINE': 'django.db.backends.postgresql',
'NAME': os.environ.get('POSTGRES_NAME'),
        'USER': os.environ.get('POSTGRES_USER'),
        'PASSWORD': os.environ.get('POSTGRES_PASSWORD'),
'HOST': 'db',
        'PORT': 5432,
    }
}
```

Listing 6. Konfiguracja bazy danych

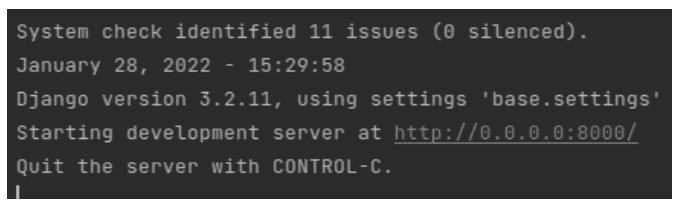
Źródło: opracowanie własne.

Za pomocą polecenia docker-compose uruchamiamy i łączymy obydwa kontenery we wspólnym środowisku. Wynik działania tego polecenia jest przedstawiony na rys. 8. Przedstawia on działające obydwa kontenery. Zgodnie z oczekiwaniami kontener zawierający bazę danych oczekuje połączeń na porcie 5432. Kontener zawierający aplikacje jest uruchomiony i oczekuje połączeń na porcie 8000, które z kolei jest przemapowane dla użytkownika końcowego na port 8003.



Rys 8. Uruchomione kontenery

Źródło: opracowanie własne.



Rys. 9. Uruchomiony serwis w kontenerze web

Źródło: opracowanie własne.

Poniżej znajduje się listing całej struktury plików i katalogów.

```
root@dlt:~/d23docker# ls -li
razem 5876
21900486 drwxr-xr-x 5 root root    4096 2020-05-04  articles
21900070 drwxr-xr-x 2 root root    4096 01-27 20:31 base
21638266 -rw-r--r-- 1 root root 5956445 01-25 15:59 d2stu.sql
21637887 drwxr-xr-x 3 root root    4096 01-27 20:27 data
21638183 -rw-r--r-- 1 root root      498 01-27 20:17 docker-
compose.yml
21638182 -rw-r--r-- 1 root root     188 01-27 20:16 Dockerfile
21900633 drwxr-xr-x 2 root root    4096 2018-07-31  images
21638204 -rwxr-xr-x 1 root root     660 01-27 20:18 manage.py
21900640 drwxr-xr-x 3 root root    4096 2017-05-21  media
22159524 drwxr-xr-x 5 root root    4096 2020-05-04  notifica-
tions
21638184 -rw-r--r-- 1 root root      133 01-27 20:59 require-
ments.txt
22159592 drwxr-xr-x 8 root root    4096 2018-07-31  static
31076519 drwxr-xr-x 6 root root    4096 01-27 19:19 student
31201986 drwxr-xr-x 2 root root    4096 2020-05-04  templates
31202010 drwxr-xr-x 5 root root    4096 2020-05-04  userpro-
file
```

Listing 7. Struktura plików i katalogów

Źródło: opracowanie własne.

Podsumowanie

W artykule przedstawiono proces uruchomienia aplikacji w środowisku kontenerowym. Jednym z zadań wykonywanych przez Dockera podczas kreowania zamkniętych środowisk jest konfigurowanie przestrzeni nazw i grup kontrolnych. Utrzymuje on procesy w każdym z kontenerów odizolowane nie tylko od innych kontenerów, ale także od samego systemu hosta. Grupy kontrolne zapewniają, że każdy kontener otrzymuje własny udział zasobów takich jak procesor, pamięć czy dysk. Zapewnia, że każdy kontener nie wyczerpuje całkowicie wszystkich zasobów na danym hoście. Proces nadzorujący zarządzanie kontenerami uruchamiania poszczególne środowiska w sieci. Oznacza to, że można izolować kontenery na poziomie aplikacji. Wszystkie kontenery aplikacji ‘A’ nie będą miały żadnego dostępu w warstwie sieciowej dla kontenerów w aplikacji

‘B’. Ponadto każda taka izolacja sieciowa może działać na jednym hoście platformy Docker i przy użyciu domyślnego sterownika sieciowego może obejmować wiele hostów platformy. Rozwiązanie takie doskonale się sprawdza w konfiguracjach typu klient – serwer. Znakomicie upraszcza sposób zarządzania oprogramowaniem i nadaje administrowaniu nowy wymiar. Zamknięcie usług sieciowych w kontenerach w znaczący sposób upraszcza administrowanie systemem i zwiększa bezpieczeństwo pracy w rozproszonej sieci.

Bibliografia

Boschetti A., Massaron L., *Python. Podstawy nauki o danych*, Helion, Gliwice 2017.

Chelladhurai J. S., Singh V., Raj P., *Docker dla praktyków*, Helion, Gliwice 2018.

Cochrane K., *Docker Cookbook Second Ediction*, Packt 2021.

Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M., Duchesnay É., *12 Scikit-learn: Machine Learning in Python* (Oct):2825−2830, 2011.

Goodrich M., Tamassia R., Goldwasser M., *Data Structures and Algorithms in Python*, Wiley 2013.

Hilpisch Y., *Derivatives Analytics with Python*, Wiley 2015.

Mckendrick R., *Mastering Docker – Foutrh Ediction*, Packt 2021.