Docker

Przygotował Michał Tracewicz

Spis treści

Wstęp

Czym jest docker i do czego służy?

Docker jest narzędziem służącym do wirtualizacji na poziomie systemu operacyjnego. Jego głównym zastosowaniem jest uniezależnienie środowiska uruchomieniowego naszej aplikacji od systemu operacyjnego na którym pracujemy oraz bibliotek, framworków etc., które są na nim zainstalowane. Co za tym idzie możemy w stosunkowo łatwy sposób pozbyć się klasycznego problemu "A u mnie działa.". Dodatkowo pozwala to na uniknięcie dużej liczby problemów przy przenoszeniu naszej aplikacji ze środowiska deweloperskiego do środowiska produkcyjnego.

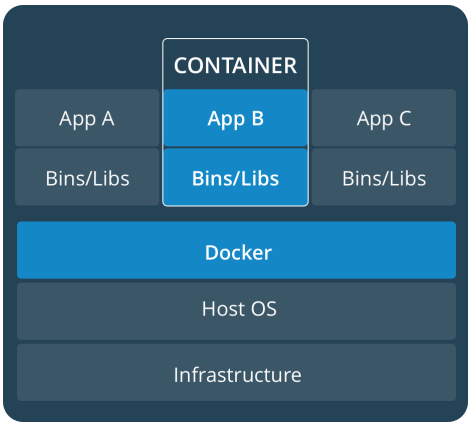
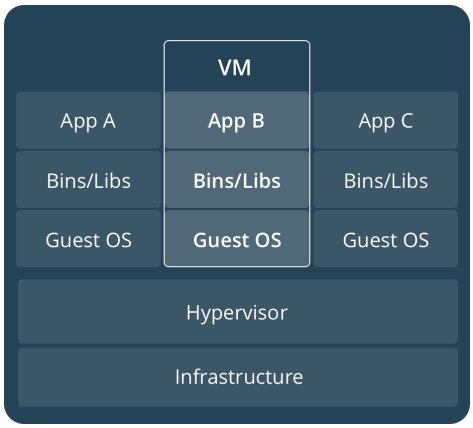
Czym docker nie jest.

Docker nie jest maszyną wirtualną. Jego celem nie jest symulowanie działania pełnego systemu operacyjnego np. w celu używania jako środowiska programistycznego, używania IDE etc. a jedynie bycie nakładką na obecnie używany system operacyjny.

Dlaczego docker a nie maszyna wirtualna

Konteneryzacja pozwala nam na znaczną oszczędność miejsca na dysku, przyśpieszyć wykonanie aplikacji (nie musimy poświęcać zasobów naszego komputera na obsługę kolejnego systemu operacyjnego) oraz ułatwienie procesu uruchomiania aplikacji i możliwość skalowania horyzontalnego.

Wizualna reprezentacja różnicy

Kontener	Maszyna wirtualne
	

Obrazki pochodzą ze strony: https://docs.docker.com/get-started/

Instalacja

Linux (Manjaro)

```
sudo pacman -S docker
sudo pacman -S docker-compose
sudo usermod -aG docker $USER
sudo systemctl enable docker
```

następnie restartujemy komputer.

Windows 10 (Wersje Pro, Education, Enterprise)

Zanim zaczniemy instalację musimy się upewni, że w ustawieniach BIOS mamy włączoną wirtualizację sprzętową.

Aby zainstalować dockera na systemie Windows 10 należy wejść na stronę dockera następnie zarejestrować się i pobrać aplikację docker desktop. Dodatkowo musimy włączyć następujące funkcje systemu Windows :

- Hyper-V
- Kontenery Windows

Kontenery

Obrazy

Aby zrozumieć czym są obrazy w dockerze najprościej będzie użyć porównania do programowania obiektowego. Gdy mówimy o obrazach są one jak klasy, jest to pewnego rodzaju projekt. Do zarządzania obrazami służy polecenie:

```
docker image
```

Aby wyświetlić pomoc używamy opcji '-h'. Niektóre ważniejsze zastosowania znajdują się poniżej.

Skąd je wziąć?

DockerHub

Jeżeli poszukujemy jakiegoś obrazu naszą najlepszą szansą jest strona DockerHub. Gdy znajdziemy interesujący nas obraz możemy go pobrać za pomocą polecenia:

```
docker pull postgres:latest
#tym poleceniem pobierzemy postgresql w najnowszej wersji.
#Możemy podmienić "latest" na inny tag np.:
docker pull postgres:9.6.16
#Jeżeli pominiemy tag przyjmie on domyślnie wartość latest
docker pull postgres
```

Budowanie

Inną opcją jest zbudowanie obrazu z pliku "dockerfile". Robimy to za pomocą polecenia:

```
docker build -t nazwa-obrazu:tag folder
```

Więcej informacji na temat budowania obrazów w sekcji Docker file

Listowanie obrazów

Możemy to zrobić za pomocą polecenia:

```
docker image ls
```

lub w skróconej wersji:

```
docker images
#Przykładowe wyjście
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
hello-world         latest              fce289e99eb9        10 months ago
1.84kB
```

Usuwanie obrazów

Aby usunąć konkretny obraz użyjemy polecenia polecenia:

```
docker image rm nazwa-obrazu
#lub
docker image rm image-id
```

Lub w skróconej wersji:

```
docker rmi nazwa-obrazu
#lub
docker rmi image-id
#możemy też podać listę obrazów do usunięcia
docker rmi nazwa-obrazu nazwa-obrazu2 ...
```

Istnieje też opcja usunięcia wszystkich nie używanych obrazów:

```
docker image prune
#otrzymamy takie ostrzeżenie, gdy wpiszemy 'y' usniemy wszystkie nieużywane
obrazy
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N]
```

Zarządzanie kontenerami

Kontynuując naszą metaforę kontenery są tym dla obrazów czym obiekty dla klas w paradygmacie obiektowym.

Jak utworzyć kontener

Najprostszym sposobem jest użycie plecenia "docker run". Poniżej przykład uruchomienia kontenera z obrazu hello world:

```
docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:4df8ca8a7e309c256d60d7971ea14c27672fc0d10c5f303856d7bc48f8cc17ff
Status: Downloaded newer image for hello-world:latest

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.
    (amd64)
 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://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Jeżeli użyjemy tego polecenia na obrazie, którego nie posiadamy aktualnie lokalnie obrazu, który próbujemy uruchomić docker przeszuka DockerHub w poszukiwaniu obrazu o podanej przez nas nazwie w najnowszej wersji.

Dla przykładu uruchomimy kontener z obrazu "debian" wersji z interaktywną powłoką bash.

```
docker run -it debian
#Aby wyjść i zatrzymać kontener użyjemy polecenia exit 13 gdzie 13 jest
przykładopwym kodem jaki możemy zwrócić zamykając kontener. Jeżeli checmy
odłączyć standardowe wejście i wyjście bez zamykania kontenera użyjemy
kombinacji CTRl-p CTRL-q
```

Używając powyższego polecenia związemy nasze standardowe wyjście i wejście z wyjściem i wejściem kontenera. Jeżeli chcemy tego uniknąć użyjemy opcji '-d'.

```
docker run -d postgres
#Gdy uruchamiamy kontener, który ma działać przez dłuższy czas warto nadać mu nazwę
docker run --name NAZWA -d postgre
#W wypadku gdy uruchomiliśmy kontener w ten sposób a chcielibyśmy się jednak podłączyć do standardowego wejścia i wyjścia możemy to zrobić poprzez
docker exec -it id-kontenera /bin/bash
#W powyższym przykładzie użyliśmy polecenia "docker exec", które pozwala nam na wykonanie jakiegoś polecenia w kontenerze.
```

Jeżeli uruchomimy naszą aplikację na jeden z wyżej wymienionych sposobów to nie będziemy mieli dostępu np. w naszej przeglądarce. Aby tego uniknąć użyjemy opcji '-p'.

```
docker run -d -p4443:4444 obraz
#w powyższym przykładzie ports 4443 jest portem lokalnym a port 4444 portem w kontenerze
```

Gdy chcemy zatrzymać uruchomiony kontener użyjemy polecenia:

```
docker stop id-kontenera
```

W sytuacji gdy mamy kontener, który już istnieje ale jest zatrzymamy możemy go uruchomić za pomocą polecenia:

```
docker start id-kontenera
```

Aby wkopiować jakiś plik do/z uruchomionego przez nas kontenera użyjemy polecenia:

```
docker cp id-kontenera:sciezka-pliku gdzie-wkleic
```

Zachowywanie zmian w kontenerze

Jeżeli zmodyfikowaliśmy nasz kontener np. poprzez utworzenie w nim pliku lub pobranie biblioteki to możemy tą zmodyfikowaną wersję zapisać za pomocą polecenia

```
docker commit id-kontenera nazwa-obrazu:tag
```

Woluminy

Jeżeli chcemy przechować dane wygenerowane/używane przez kontener możemy to zrobić za pomocą woluminów. Są one niezależnym od systemu hosta sposobem na przetrzymywanie danych przez kontenery.

Aby utworzyć wolumin użyjemy polecenia:

```
docker volume create nazwa-woluminu
```

Aby wyświetlić wszystkie woluminy na danej maszynie użyjemy polecenia:

```
docker volume ls
```

jeżeli chcemy usunąć, któryś z istniejących woluminów użyjemy do tego polecenia:

```
docker volume rm nazwa-woluminu
#Alternatywnie możemy usunąć wszystkie woluminy
docker volume prune
```

Aby uruchomić aplikację i podłączyć do niej wolumin użyjemy opcji '-v' w poleceniu docker run. Dla przykładu:

```
docker run -d --name devtest -v myvol2:/app nginx:latest
#W powyższej komędzie myvol2 jest nazwą woluminu a /app jest miejscem gdzie
zostanie on domontowany w kontenerze. Jeżeli katalog do którego zmountujemy nasz
wolumin nie jest pusty to jego zawartość zostanie zapisana w naszym woluminie.
```

Listowanie kontenerów

Aby pokazać aktualnie uruchomione kontenery użyjemy polecenia:

```
docker ps
```

Jeżeli chcemy jeszcze wyświetlić kontenery, również te, które nie są uruchomione:

```
docker ps -a
```

Usuwanie kontenera

Aby usunąć kontener, który jest zatrzymany użyjemy polecenia:

```
docker rm id-kontenera
#Jeżeli chcemy usunąć działający kontener użyjemy opcji '-f'
docker rm -f id-kontenera
#Jeżeli chcielibyśmy usunąć voluminy przypisane do kontenera który usuwamy
użyjemy opcji '-v'
docker rm -v id-kontenera
```

Możemy usunąć wszystkie zatrzymane kontenery za pomocą:

```
docker rm $(docker ps -a -q)
#Jeżeli chcemy usunąć wszystkie kontenery (włącznie z uruchomionymi)
docker rm -f $(docker ps -a -q)
```

Dockerfile

Dockerfile jest plikiem zawierającym instrukcje w jaki sposób docker ma zbudować nasz obraz.

Składnia

```
#Komentarze rozpoczynamy od '#'
FROM python:latest
```

```
#From zawsze musi być pierwszą linijką w pliku dockerfile, mówi nam jaki obraz mamy przyjąć za bazowy, tak jak przy docker pull możemy pominąć tag, wtedy przyjmie on wartość domyślną latest
ENV test /myapp
#ENV pozwala nam stworzyć zmienne środowiskowe/aliasy do używania w naszym pliku, tutaj gdy użyjemy $test podstawiona zostanie wartość /mydir
WORKDIR $test
#WORKDIR ustawiamy katalog roboczy dla naszego kontenera tzn. zod tego miejsca będą rozpatrywane wszystkie względne ścieżki podane w dalszej części pliku
COPY ./app .
#COPY ---chown=uzytkownik:grupa ./doc/* /app/doc/
#COPY kopiuje pliki ze wskazanego przez nas miejsca na dysku lokalnym (pierwszy argument) do miejsca w kontenerze (drugi argument). Jeżeli użyjemy opcji --chown uzytkownik:grupa (działa tylko na linuxie) to nasz plik w kontenerze będzie miał takiego właściciela i grupę
EXPOSE 8000
#EXPOSE pozwala nam wybrać na jakich poratach i z użyciem jakiego protokołu (domyślnie tcp) nasz kontener będzie nasłuchiwał na "świat" tj. pod jakim portem na naszej maszynie będzie dostępny.
#USER uzytkownik:grupa
#USER pozwala na ustawienie użytkownika i grupy, które będą używane gdy zrobimy docker run
VOLUME /data
#VOLUME tworzy nowy wolumin we wskazanym miejscu i zapełnia go danymi znajdującymi się w tym katalogu
RUN sudo pip install -r ./app/requirements.txt
#RUN pozwala nam wykonać polecenie w powłoce
CMD python3 ./app/manage.py runserver
#CMD może wystąpić tylko raz w dockerfile i jest domyślnie wykonane przy uruchamianiu kontenera
```

Jeżeli chcemy aby jakieś foldery/pliki były pomijane przy poleceniu COPY możemy stworzyć plik .dockerignore. Np.:

```
temp
.vscode/*
.idea/*
```

docker-compose.yml

Plik ten służy do łączenia kilku obrazów w jeden uruchamiany.

Przykładowy plik ze strony: https://docs.docker.com/compose/compose-file/

```
version: "3.7"
#Wersja
services:
#Składowe serwisy
  redis:
  #Nazwa serwisu
    image: redis:alpine
    #obraz
    ports:
    #porty wystawiane (jako, że podany jest jeden będzie on wystawiony w sieci zdwfiniownej w podpunkcie networks:)
```

```
      - "6379"
    networks:
    #możemy tworzyć sieci po których kontenery będą się komunikować, ten jest w
sieci frontend
      - frontend
    deploy:
      replicas: 2
      #ilość kontenerów tego serwisu uruchomionych w danym momęcie
      update_config:
      #Konfiguruje jak kontener powinien być aktualizowany
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
        #Ustala czy i jak kontener powinien zostać ponownie postawiony po
zakończeniu pracy

  db:
    image: postgres:9.4
    volumes:
    #Mountujemy volumin db-data do katlogu /var/lib/postgresql/data
      - db-data:/var/lib/postgresql/data
    networks:
      - backend
    deploy:
      placement:
        constraints: [node.role == manager]

  vote:
    image: dockersamples/examplevotingapp_vote:before
    ports:
    #W tym miejscu wystawiamy port 80 naszej aplikacji na port 5000 naszej
maszyny
      - "5000:80"
    networks:
      - frontend
    depends_on:
      - redis
    deploy:
      replicas: 2
      update_config:
        parallelism: 2
      restart_policy:
        condition: on-failure

  result:
    image: dockersamples/examplevotingapp_result:before
    ports:
      - "5001:80"
    networks:
      - backend
    depends_on:
    #Ta sekcja mówi nam, że service result polega na serwisie db więc w momęcie
użycia polecenia docker-compose up najpierw uruchomiony zostanie service db a
dopiero potem result
      - db
    deploy:
      replicas: 1
```

```
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure

  worker:
    image: dockersamples/examplevotingapp_worker
    networks:
      - frontend
      - backend
    deploy:
      mode: replicated
      replicas: 1
      labels: [APP=VOTING]
      restart_policy:
        condition: on-failure
        delay: 10s
        max_attempts: 3
        window: 120s
      placement:
        constraints: [node.role == manager]

  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    stop_grace_period: 1m30s
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]

networks:
#definiujemy sieci
  frontend:
  backend:
#definiujemy voluminy
volumes:
  db-data:
```

Gdy utworzymy taki plik możemy go użyć aby postawić naszą aplikację za pomocą polecenia:

```
docker-compose up
```

Następnie możemy ją zatrzymać poprzez:

```
docker-compose down
```

Bibliografia

https://docs.docker.com/

Wstęp

- https://docs.docker.com/engine/docker-overview/
- https://docs.docker.com/get-started/
- https://djangoforprofessionals.com/docker/

Instalacja

Manjaro

- https://gamblisfx.com/install-docker-on-manjaro-linux/
- https://manjaro.site/how-to-install-docker-on-manjaro-18-0/

Windows

- https://docs.docker.com/docker-for-windows/install/

Kontenery

- https://hub.docker.com
- https://docs.docker.com/engine/reference/commandline/cp/
- https://docs.docker.com/engine/reference/commandline/run/
- https://docs.docker.com/engine/reference/commandline/commit/

- https://howtoprogram.xyz/2017/03/18/attach-detach-docker-container/
- man docker image
- https://docs.docker.com/storage/volumes/

Docker file

- https://docs.docker.com/engine/reference/builder/

docker-compose.yml

- https://docs.docker.com/compose/compose-file/