

Tworzenie i uruchamianie prostych kontenerów w środowisku docker-compose

Grzegorz Koperwas

13 stycznia 2021

1 docker 101

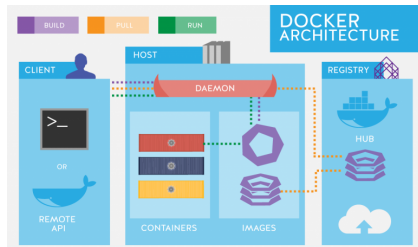
- Po co docker
- Po co obrazy aplikacji
- Pliki dockerfile

2 Uruchamianie w docker-compose

- Plik docker-compose.yml
- Uruchamianie kontenerów za pomocą docker-compose

Po co jest docker?

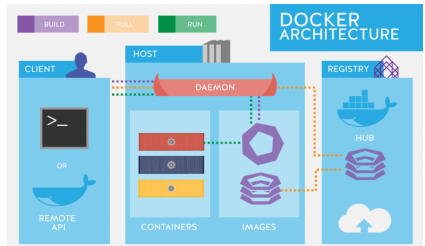
Docker znacząco ułatwia instalowanie i zarządzanie oprogramowaniem w systemach *Linux*.



Po co jest docker?

Docker znacząco ułatwia instalowanie i zarządzanie oprogramowaniem w systemach *Linux*.

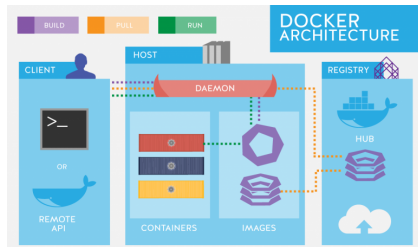
- Izoluje programy oraz ich zależności.



Po co jest docker?

Docker znacząco ułatwia instalowanie i zarządzanie oprogramowaniem w systemach *Linux*.

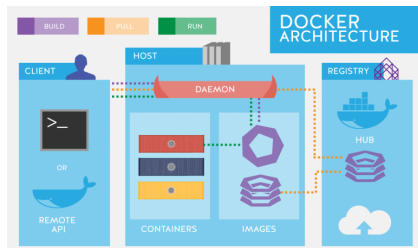
- Izoluje programy oraz ich zależności.
- Łatwa instalacja programów spoza repozytoriów naszej dystrybucji.



Po co jest docker?

Docker znacząco ułatwia instalowanie i zarządzanie oprogramowaniem w systemach *Linux*.

- Izoluje programy oraz ich zależności.
- Łatwa instalacja programów spoza repozytoriów naszej dystrybucji.
- Łatwe zarządzanie obrazami aplikacji.

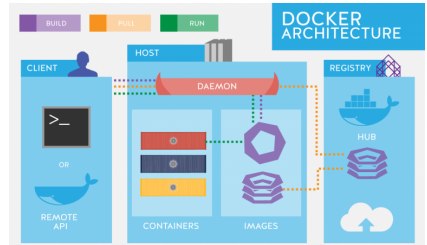


Po co jest docker?

Docker znacząco ułatwia instalowanie i zarządzanie oprogramowaniem w systemach *Linux*.

- Izoluje programy oraz ich zależności.
- Łatwa instalacja programów spoza repozytoriów naszej dystrybucji.
- Łatwe zarządzanie obrazami aplikacji.

- Skalowalność dla wielu serwerów za pomocą *Kubernetes*



Do czego nie jest docker

Docker ma oczywiście swoje wady:

Do czego nie jest docker

Docker ma oczywiście swoje wady:

- Nie jest tak szybki jak jego brak.

Do czego nie jest docker

Docker ma oczywiście swoje wady:

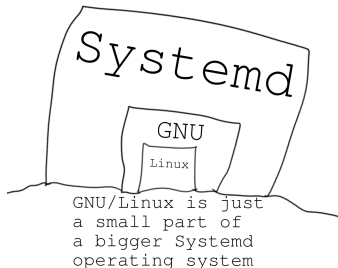
- Nie jest tak szybki jak jego brak.
- Trwałe przechowywanie danych jest *ciekawe*.

Do czego nie jest docker

Docker ma oczywiście swoje wady:

- Nie jest tak szybki jak jego brak.
- Trwałe przechowywanie danych jest *ciekawe*.

Jeśli coś może być łatwo osiągnięte czymś takim jak systemd, to warto to tak zrobić.



Czym jest obraz aplikacji i plik dockerfile

By uruchomić daną aplikację w środowisku docker'a musimy utworzyć jej obraz.

Czym jest obraz aplikacji i plik dockerfile

By uruchomić daną aplikację w środowisku docker'a musimy utworzyć jej obraz.

Docker tworzy swoje obrazy według instrukcji w pliku zwanym dockerfile.

Czym jest obraz aplikacji i plik dockerfile

By uruchomić daną aplikację w środowisku docker'a musimy utworzyć jej obraz.

Docker tworzy swoje obrazy według instrukcji w pliku zwanym `dockerfile`.

Możemy o nich myśleć jako instrukcjach które znajdują się na wielu repozytoriach na *Github*'ie mówiących jakie biblioteki są potrzebne do kompilacji oraz jak kompilować i instalować dany program.

Na potrzeby tej prezentacji będziemy chcieli stworzyć obraz z kompilatorem sieciowym *distcc*.

Na potrzeby tej prezentacji będziemy chcieli stworzyć obraz z kompilatorem sieciowym *distcc*.

Pierwszym krokiem do stworzenia jakiegoś obrazu aplikacji musimy wybrać obraz *bazowy*.

Na potrzeby tej prezentacji będziemy chcieli stworzyć obraz z kompilatorem sieciowym *distcc*.

Pierwszym krokiem do stworzenia jakiegoś obrazu aplikacji musimy wybrać obraz *bazowy*.

My będziemy wykorzystywali obraz `archlinux:latest`.

Na potrzeby tej prezentacji będziemy chcieli stworzyć obraz z kompilatorem sieciowym *distcc*.

Pierwszym krokiem do stworzenia jakiegoś obrazu aplikacji musimy wybrać obraz *bazowy*.

My będziemy wykorzystywali obraz `archlinux:latest`.

Nazwy obrazów

Nazwy obrazów są w formie:

`<autor>?/<nazwa>:tag`

Na przykład:

`jrottenberg/ffmpeg:4.1-nvidia`

Plik Dockerfile

```
1 FROM archlinux:latest
2 RUN pacman -Syu --noconfirm distcc make git gcc cmake
3 COPY ./dummy.txt /etc/dummy.txt
4 EXPOSE 3632/tcp
5 EXPOSE 3632/udp
6 ENTRYPOINT distccd --daemon --no-detach --verbose --allow-private
```

Plik dockerfile składa się z takich instrukcji:

Plik Dockerfile

```
1 FROM archlinux:latest
2 RUN pacman -Syu --noconfirm distcc make git gcc cmake
3 COPY ./dummy.txt /etc/dummy.txt
4 EXPOSE 3632/tcp
5 EXPOSE 3632/udp
6 ENTRYPOINT distccd --daemon --no-detach --verbose --allow-private
```

Plik dockerfile składa się z takich instrukcji:

FROM Definiuje bazowy obraz

Plik Dockerfile

```
1 FROM archlinux:latest
2 RUN pacman -Syu --noconfirm distcc make git gcc cmake
3 COPY ./dummy.txt /etc/dummy.txt
4 EXPOSE 3632/tcp
5 EXPOSE 3632/udp
6 ENTRYPOINT distccd --daemon --no-detach --verbose --allow-private
```

Plik dockerfile składa się z takich instrukcji:

FROM Definiuje bazowy obraz

RUN Uruchamia komendę w kontenerze

Plik Dockerfile

```
1 FROM archlinux:latest
2 RUN pacman -Syu --noconfirm distcc make git gcc cmake
3 COPY ./dummy.txt /etc/dummy.txt
4 EXPOSE 3632/tcp
5 EXPOSE 3632/udp
6 ENTRYPOINT distccd --daemon --no-detach --verbose --allow-private
```

Plik dockerfile składa się z takich instrukcji:

FROM Definiuje bazowy obraz

RUN Uruchamia komendę w kontenerze

EXPOSE Udostępnia porty w kontenerze

Plik Dockerfile

```
1 FROM archlinux:latest
2 RUN pacman -Syu --noconfirm distcc make git gcc cmake
3 COPY ./dummy.txt /etc/dummy.txt
4 EXPOSE 3632/tcp
5 EXPOSE 3632/udp
6 ENTRYPOINT distccd --daemon --no-detach --verbose --allow-private
```

Plik dockerfile składa się z takich instrukcji:

FROM Definiuje bazowy obraz

RUN Uruchamia komendę w kontenerze

EXPOSE Udostępnia porty w kontenerze

COPY Kopiuje plik/katalog *A* z naszego komputera do katalogu *B* w obrazie.

Plik Dockerfile

Obraz budujemy za pomocą polecenia:

```
$ docker build -t $nazwaObrazu .
```

Gdzie zamiast \$nazwaObrazu wpisujemy nazwę dla naszego obrazu.

Uruchamianie obrazów w środowisku docker-compose

Domyślnie Docker
uruchamia obrazy
poleceniami `docker run`.

```
# docker run \  
  --gpus all \  
  --network "host" \  
  --device /dev/ttyUSB0:/dev/tty2 \  
  --volume /bar/off/foo:/bar \  
  rtsp_over_serial:latest
```

Uruchamianie obrazów w środowisku docker-compose

Domyślnie Docker
uruchamia obrazy
poleceniami `docker run`.

`docker-compose` ułatwia
proces uruchamiania
poprzez zastosowanie pliku
YAML z opcjami które
normalnie byśmy wpisywali
jako argumenty polecenia
`docker run`

```
# docker run \  
  --gpus all \  
  --network "host" \  
  --device /dev/ttyUSB0:/dev/tty2 \  
  --volume /bar/off/foo:/bar \  
  rtsp_over_serial:latest
```

Format pliku docker-compose.yml

```
1 version: '3.3'
2
3 services:
4   distcc:
5     image: distcc:latest
6     restart: always
7     ports:
8       - "3632:3632"
```

Plik jest w formacie *YAML*, w
formie klucz → wartość

Format pliku docker-compose.yml

```
1 | version: '3.3'
2 |
3 | services:
4 |     distcc:
5 |         image: distcc:latest
6 |         restart: always
7 |         ports:
8 |             - "3632:3632"
```

Plik jest w formacie *YAML*, w formie klucz → wartość

Pod kluczem `services` umieszczamy kontenery jakie będziemy uruchamiać.

Format pliku docker-compose.yml

Kolejne klucze określają opcje dla kontenera (i z jakiego obrazu go stworzyć)

image Nazwa obrazu dla kontenera.

```
1 version: '3.3'
2
3 services:
4   distcc:
5     image: distcc:latest
6     restart: always
7     ports:
8       - "3632:3632"
```

Plik jest w formacie *YAML*, w formie klucz → wartość

Pod kluczem *services* umieszczamy kontenery jakie będziemy uruchamiać.

Format pliku docker-compose.yml

Kolejne klucze określają opcje dla kontenera (i z jakiego obrazu go stworzyć)

```
1 version: '3.3'
2
3 services:
4   distcc:
5     image: distcc:latest
6     restart: always
7     ports:
8       - "3632:3632"
```

image Nazwa obrazu dla kontenera.

restart Czy restartować kontener po jego zakończeniu.

Plik jest w formacie *YAML*, w formie klucz → wartość

Pod kluczem *services* umieszczamy kontenery jakie będziemy uruchamiać.

Format pliku docker-compose.yml

Kolejne klucze określają opcje dla kontenera (i z jakiego obrazu go stworzyć)

```
1 version: '3.3'
2
3 services:
4   distcc:
5     image: distcc:latest
6     restart: always
7     ports:
8       - "3632:3632"
```

Plik jest w formacie *YAML*, w formie klucz → wartość

Pod kluczem *services* umieszczamy kontenery jakie będziemy uruchamiać.

image Nazwa obrazu dla kontenera.

restart Czy restartować kontener po jego zakończeniu.

ports Konfiguracja forwardowania portów, w formie <port hosta>:<port kontenera>

Uruchamianie kontenerów komendą docker-compose

By uruchomić kontener wystarczy polecenie:

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

Gdzie opcja -d odłącza proces od terminala.

Uruchamianie kontenerów komendą docker-compose

By uruchomić kontener wystarczy polecenie:

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

Gdzie opcja `-d` odłącza proces od terminala.

Bonus

Docker-compose udostępnia nam też inne komendy takie jak `docker-compose logs` wypisującą logi na konsoli. Za pomocą takiego one-linera możemy sobie je wyświetlać na bieżąco:

```
$ watch --color "docker-compose logs | tail -n 20"
```