Wirtualne środowiska programistyczne

Wirtualizacja

Proces symulowania przez oprogramowanie pewnych zasobów np. wirtualna maszyna stosuje wirtualizację w celu symulowania pracy maszyny z danym systemem operacyjnym bez wpływania na realnie posiadany system operacyjny.

HashiCorp

Vagrant

Co to Vagrant?

Vagrant jest narzędziem umożliwiającym tworzenie maszyn wirtualnych na podstawie pliku konfiguracyjnego. Taki VM może zostać uruchomiony na środowisku VirtualBox, VMWare, czy w chmurze. Można się w ten sposób uniezależnić od środowiska, w którym się pracuje.

Jak korzystać z Vagranta?

Aby korzystać z Vagranta należy po prostu zainstalować właściwą wersję dla danego systemu operacyjnego. Następnie potrzebna jest konfiguracja maszyny. Opisuje ją plik Vagrantfile, którego składnia jest oparta o język Ruby.

Plik konfiguracyjny zawiera w sobie informację o obrazie systemu operacyjnego z jakiego ma skorzystać, jakie pakiety doinstalować, jak skonfigurować sieć i usługi oraz inne parametry.

```
Vagrant.configure(“2”) do |config|

config.vm.box = “bento/ubuntu-16.10”

config.vm.network: private_network, ip: ‘192.168.56.2’

config.vm.provider “virtualbox” do |vb|

        vb.name = “name”

        vb.cpus = 4

        vb.memory = 8192

        vb.gui = true

end

config.vm.provision “file”,

        source: “eclipse.desktop”,

        destination: “home/vagrant/Desktop/eclipse.desktop”

end
```


docker

Docker jest to narzędzie, które pozwala umieścić aplikacje, jak i jej zależności (biblioteki, pliki konfiguracyjne, lokalne bazy danych itp.) w lekkim, przenośnym, wirtualnym kontenerze, który można uruchomić na prawie każdym serwerze.

Docker uruchamia w kontenerze tylko i wyłącznie procesy aplikacji bez potrzeby emulowania warstwy sprzętowej i systemu operacyjnego, dzięki czemu zwiększa się efektywność wykorzystania zasobów sprzętowych.

Docker umożliwia przenoszenie kontenerów pomiędzy różnymi systemami lub architekturami, bez konieczności wprowadzania zmian w aplikacji.

Docker wykorzystuje już istniejące rozwiązania stosowane w systemach z rodziny Linux:

Grupy kontrolne i przestrzenie nazw.

Kontenery Linuxa.

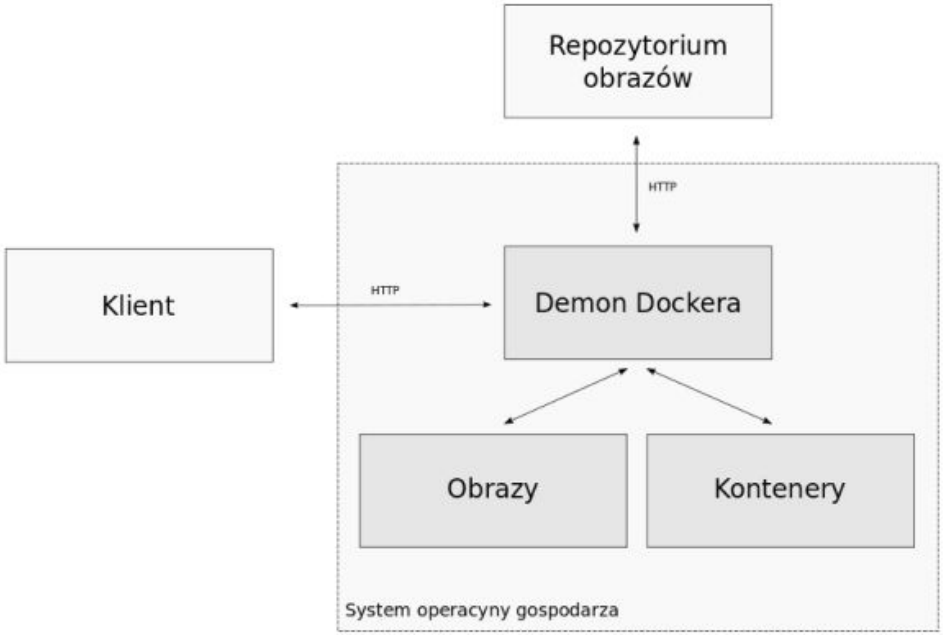
Ujednolicony system plików.

Przestrzenie nazw (ang. namespaces) i grupy kontrolne (ang. cgroups) pozwalają na zarządzenie dostępem do zasobów systemowych (procesor, pamięć, sieć itd.) przez poszczególne procesy.

Kontenery linuxa (LXC, ang. Linux Containers) pozwalają na uruchomienie wielu systemów gości w obrębie jednego systemu gospodarza, zarządzanie ich zasobami (cgroups) i odizolowanie ich od systemu operacyjnego (namespaces).

Ujednolicony system plików (UFS, ang. Union File System) umożliwia zarządzanie plikami i katalogami pochodzącymi z różnych systemów plików tak jakby stanowiły jeden system plików.

Architektura Dockera



Demon Dockera jest odpowiedzialny za zarządzanie obrazami i kontenerami, np.: pobiera obrazy z repozytorium, buduje obrazy, uruchamia kontenery.

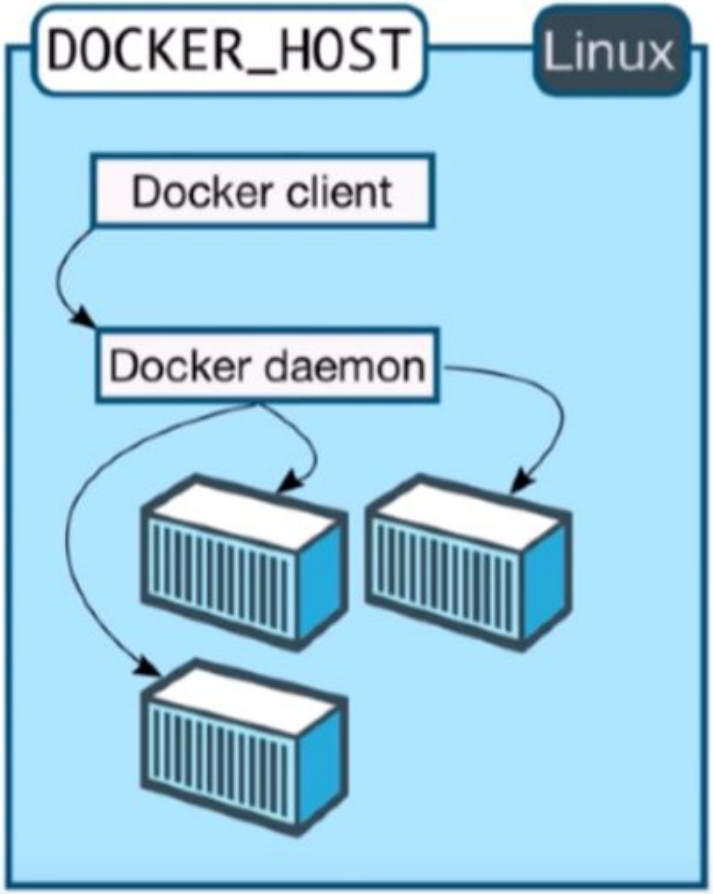
Klient Dockera umożliwia wydawanie poleceń demonowi. Komunikacja pomiędzy klientem a demonem odbywa się za pomocą protokołu HTTP. W przypadku komunikacji lokalnej wykorzystywane są do tego gniazda UNIX, a w przypadku komunikacji zdalnej gniazda TCP.

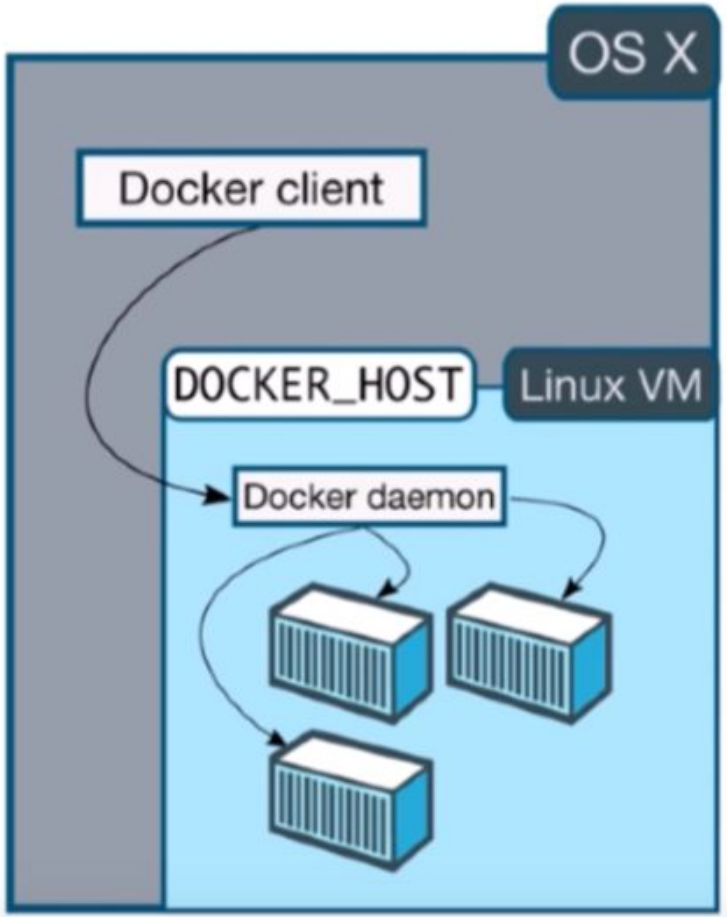
Repozytorium obrazów przechowuje obrazy kontenerów Dockera. W ekosystemie Dockera znajduje się jedno oficjalne repozytorium (docker hub). Zawiera ono oficjalne obrazy oraz obrazy udostępnione przez innych użytkowników. Istnieje również możliwość uruchomienia prywatnego repozytorium.

Wymagania

Warunkiem korzystania z Dockera jest posiadanie 64-bitowego systemu z rodziny Linux z jądrem w wersji co najmniej 3.10.

Docker dostarcza też narzędzia Docker Toolbox pozwalające na uruchamianie Docker Engine na różnych systemach operacyjnych, poprzez zastosowanie wbudowanej wirtualizacji (VirtualBox).


DOCKER_HOST
Linux
Docker client
Docker daemon


OS X
Docker client
DOCKER_HOST
Linux VM
Docker daemon

Obraz

Obraz stanowi podstawę do stworzenia kontenera. Obraz to inaczej taki szkielet, gotowe środowisko dla naszej aplikacji.

Obraz jest ujednoliconą kolekcją warstw UFS (Union File System). Każda warstwa jest wynikiem wykonania pewnej czynności podczas tworzenia obrazu, np. instalacji pakietu, skopiowania danych, ustawienia zmiennej środowiskowej. Warstwy obrazu są dostępne tylko do odczytu.

Podstawowym źródłem obrazów jest repozytorium obrazów Dockera: dockerhub. W repozytorium znajduje się wiele oficjalnych obrazów, np. ubuntu, postgres, redis, node oraz nieoficjalne obrazy stworzone przez użytkowników Dockera.

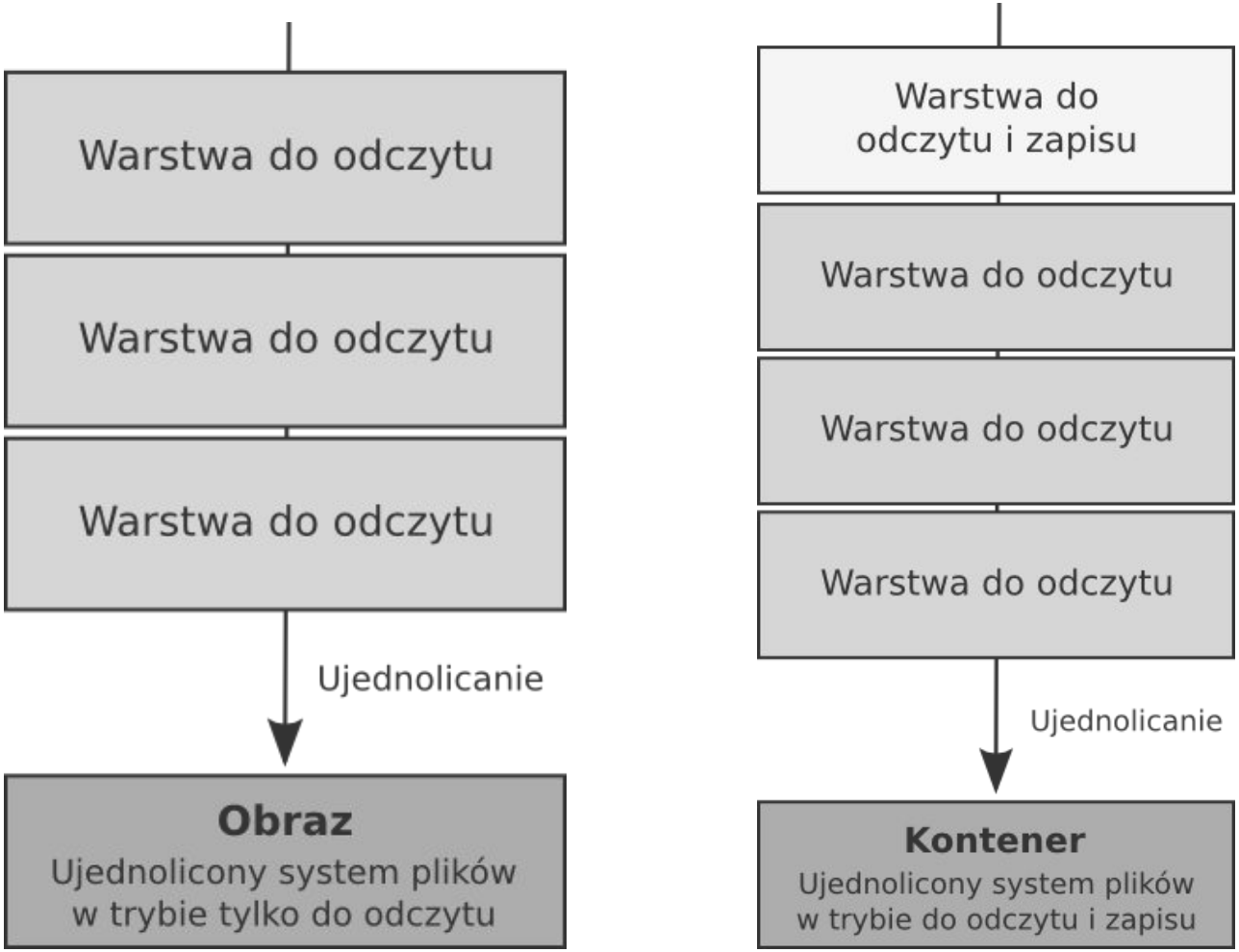
Konteneryzacja

Konteneryzacja polega na tym, że umożliwia uruchomienie wskazanych procesów aplikacji w wydzielonych kontenerach, które z punktu widzenia aplikacji są odrębnymi instancjami środowiska uruchomieniowego. Każdy kontener posiada wydzielony obszar pamięci, odrębny interface sieciowy z własnym prywatnym adresem IP oraz wydzielony obszar na dysku, na którym znajduje się zainstalowany obraz systemu operacyjnego i wszystkich zależności / bibliotek potrzebnych do działania aplikacji.

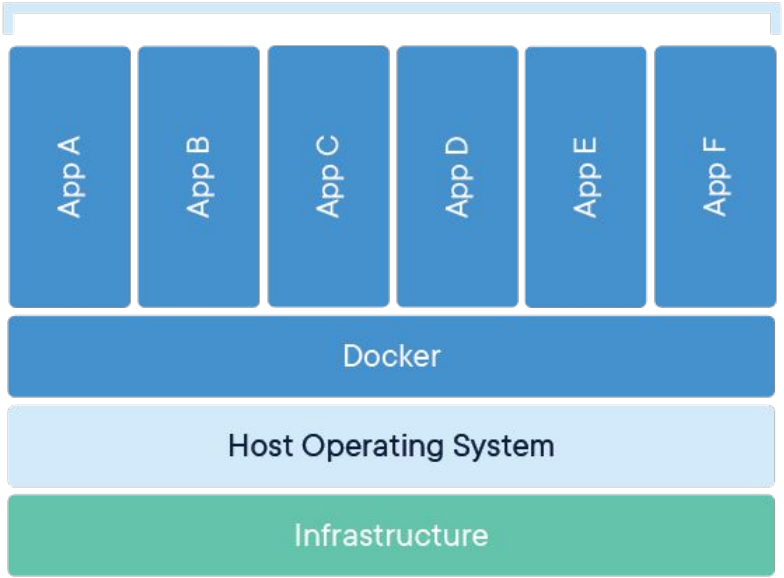
Kontener można pojmować jako komponent systemu informatycznego, który pełni pojedynczą funkcję w jego architekturze.

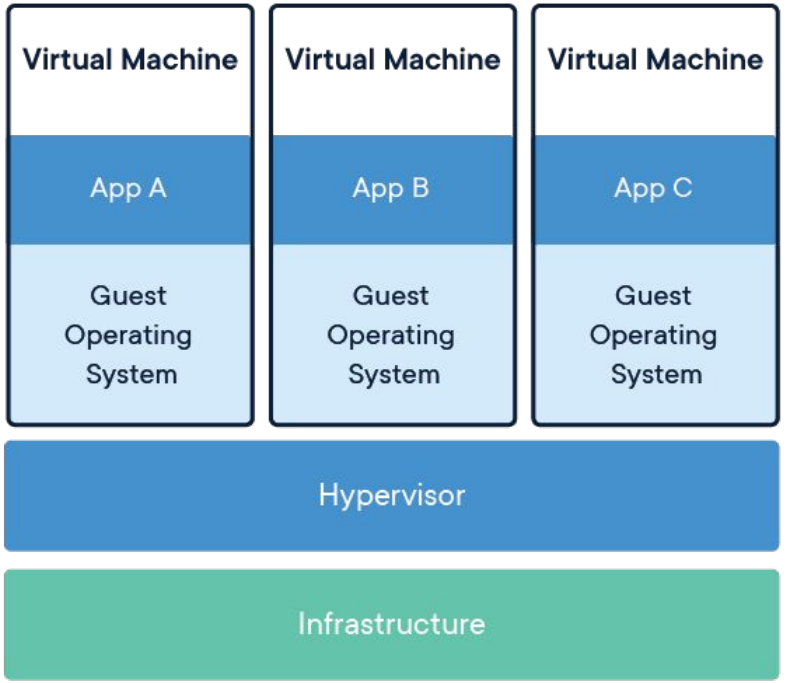
Przeznaczeniem kontenera jest wykonywanie pojedynczego zadania (procesu). Kontener może znajdować się w jednym ze stanów:

- “created” – Kontener, który został stworzony, ale jeszcze nie uruchomiony.
- “restarting” – Kontener w trakcie ponownego uruchamiania swojego procesu.
- “up” – Kontener wykonujący swój proces.
- “paused” – Kontener, którego proces został wstrzymany.
- “exited” – Kontener, który zakończył wykonywanie swojego procesu.


Warstwa do odczytu
Warstwa do odczytu
Warstwa do odczytu
Ujednolicanie
Obraz
Ujednolicony system plików
w trybie tylko do odczytu
Warstwa do
odczytu i zapisu
Warstwa do odczytu
Warstwa do odczytu
Warstwa do odczytu
Ujednolicanie
Kontener
Ujednolicony system plików
w trybie do odczytu i zapisu

Containerized Applications



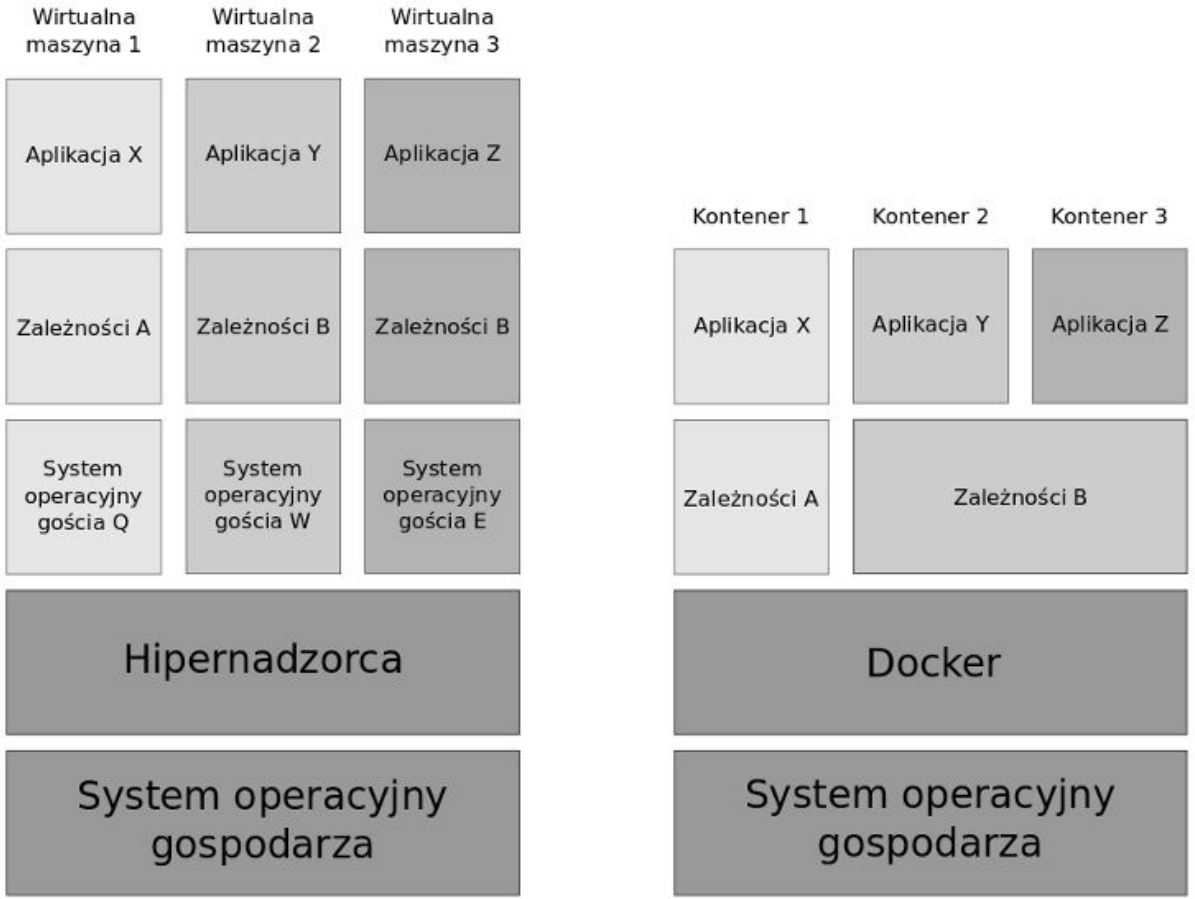


Kontener a maszyna wirtualna

Kontenery Dockera, w porównaniu z wirtualnymi maszynami, pozwalają osiągnąć istotnie niższe zużycie pamięci dyskowej i niższe czasy uruchomienia.

Kontenery są w stanie współdzielić warstwy UFS obrazów (warstwy tylko do odczytu). Poprzez to następuje niższe zużycie pamięci dyskowej. Załóżmy, że mamy obraz kontenera zajmujący 1 GB oraz obraz wirtualnej maszyny o takim samym rozmiarze. Jeżeli chcielibyśmy za ich pomocą stworzyć *n* wirtualnych maszyn i *n* kontenerów to w przypadku wirtualnych maszyn potrzebowalibyśmy *n* GB, a w przypadku kontenerów 1 GB.

Niższe czasy uruchomienia są możliwe poprzez współdzielenie jądra systemu pomiędzy kontenerami, a systemem gospodarza. Dzięki temu rozwiązaniu kontenery nie są obarczone narzutem czasowym związanym z uruchomieniem wirtualnej maszyny i działaniem hipernadzorcy.


Wirtualna maszyna 1
Wirtualna maszyna 2
Wirtualna maszyna 3
Aplikacja X
Aplikacja Y
Aplikacja Z
Zależności A
Zależności B
Zależności B
System operacyjny gościa Q
System operacyjny gościa W
System operacyjny gościa E
Hipernadzorca
System operacyjny gospodarza
Kontener 1
Kontener 2
Kontener 3
Aplikacja X
Aplikacja Y
Aplikacja Z
Zależności A
Zależności B
Docker
System operacyjny gospodarza

Podstawowe komendy dot. obrazów i kontenerów

```
# pobranie najnowszej wersji obrazu
sudo docker pull nazwa_obrazu
# pobranie konkretnej wersji obrazu
sudo docker pull nazwa_obrazu:wersja
# wyszukanie obrazu w dockerhubie
sudo docker search nazwa_obrazu
# pokazanie wszystkich pobranych obrazów
sudo docker images
# usunięcie obrazu
sudo docker rmi id_obrazu
```

```
# pokazanie wszystkich aktywnych kontenerów
sudo docker ps
# pokazanie wszystkich stworzonych kontenerów
sudo docker ps -a
# stworzenie kontenera
sudo docker create nazwa_obrazu
# uruchamianie kontenera
sudo docker run <nazwa_kontenera>
# start kontenera
sudo docker start
# zatrzymanie kontenera
sudo docker stop
# kopiowanie do/z kontera (działa podobnie jak cp lub scp)
sudo docker cp <źródło> <cel>
# wejście do kontenera w celu uruchomienia w nim jakiś rzeczy
sudo docker exec -it <nazwa_kontenera> bash
```

Zalety Dockera dla środowisk korporacyjnych

Bezpieczeństwo: kontenery są odizolowane od siebie, pomimo tego że działają w ramach tego samego systemu operacyjnego. Dzięki temu stanowią idealne rozwiązanie dla zespołów, które chcą szybko i elastycznie rozwijać swoją infrastrukturę jednocześnie przestrzegając norm bezpieczeństwa.

Zalety Dockera dla środowisk korporacyjnych

Przenośność infrastruktury i środowiska aplikacji: uruchamianie kontenera na dowolnym środowisku, umożliwia skalowanie aplikacji bez konieczności instalowania dodatkowego oprogramowania i dostosowywania kodu źródłowego aplikacji.

Zalety Dockera dla środowisk korporacyjnych

Szybkość: kontenery usprawniają cykl życia aplikacji, co pomaga programistom szybciej tworzyć aplikacje, a zespół operacyjny IT może sprawniej reagować na zmieniające się potrzeby biznesowe wdrażając nowe ulepszone wersje aplikacji.

The End