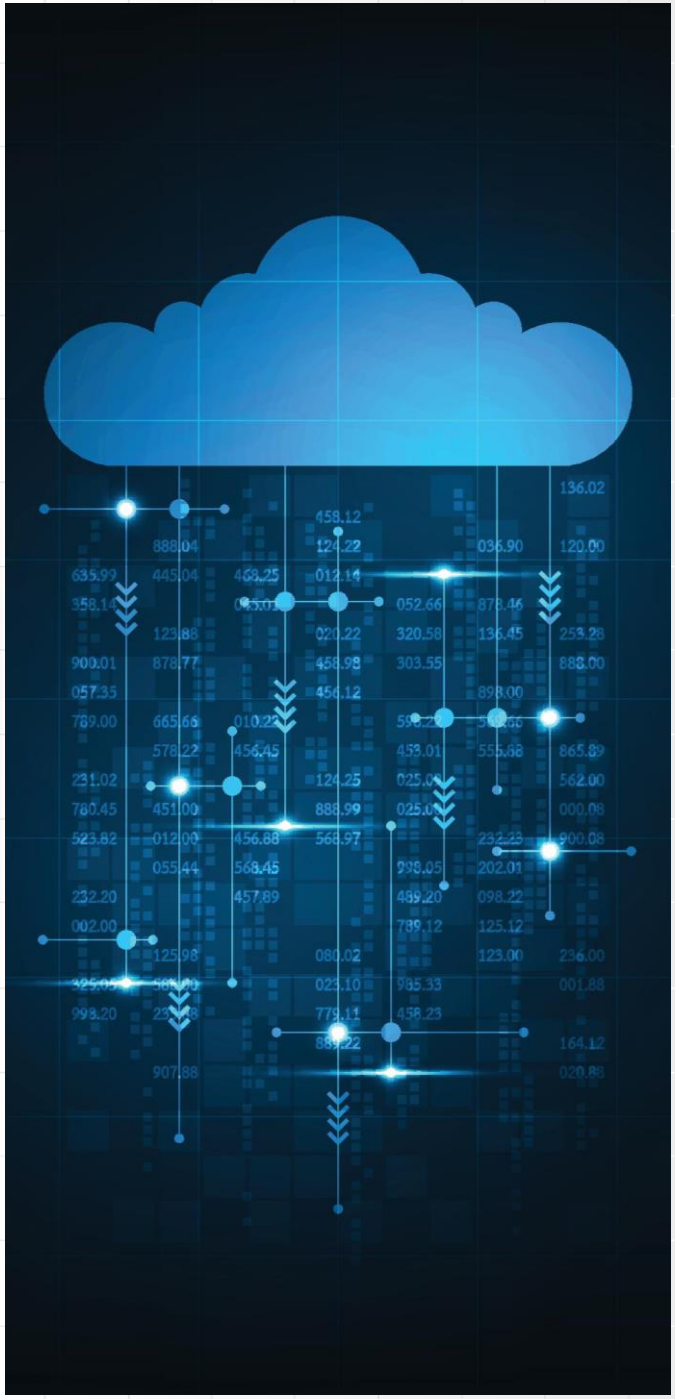


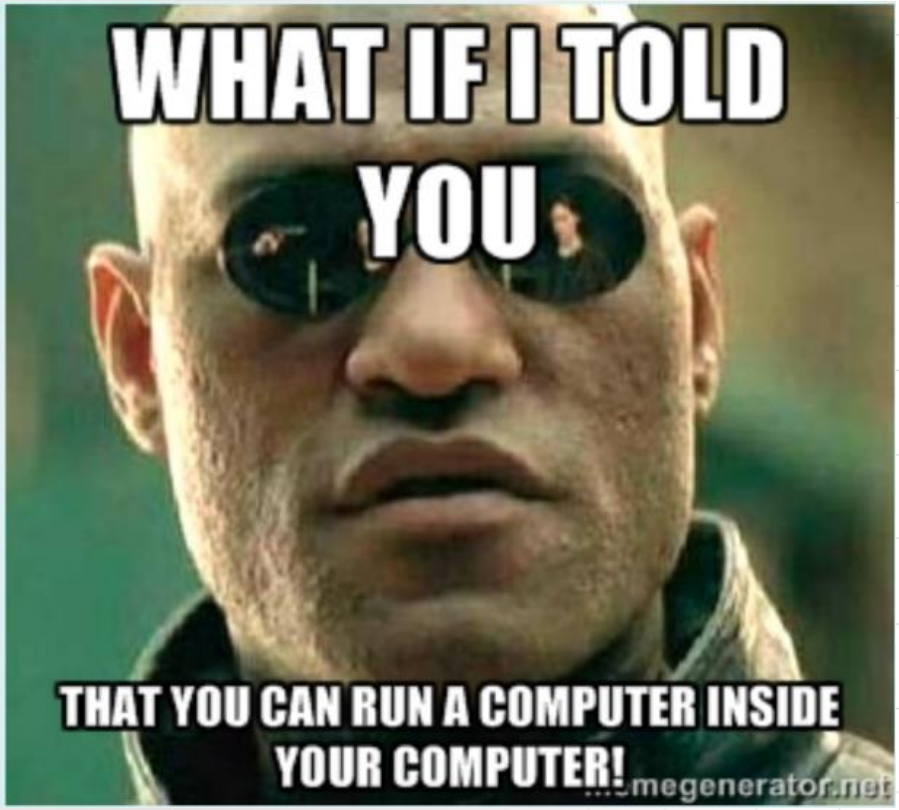
Programowanie w chmurze

Rafał Palak

Politechnika Wrocławska

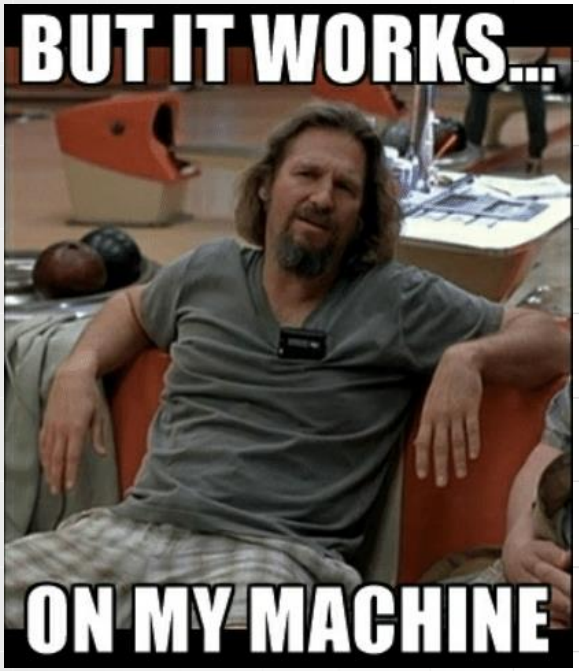
Docker

- **Platforma oprogramowania** służąca do **automatyzacji procesu wdrażania, skalowania oraz zarządzania aplikacjami przez konteneryzację**. Pozwala deweloperom **„pakować" aplikacje wraz z ich zależnościami** w standardowe jednostki oprogramowania, zwane **kontenerami**, które są **izolowane od środowiska i jednolicie działają na różnych systemach** i **infrastrukturach**.



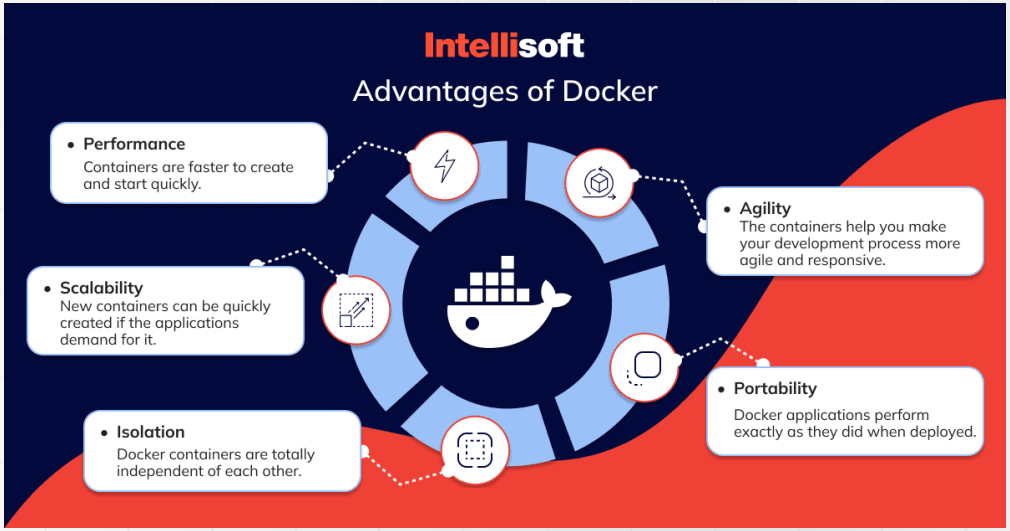
Docker - inowacyjność

- **Przenośność** - Aplikacje zapakowane w kontenery Docker mogą działać **niezmiennie na różnych środowiskach**, od deweloperskich laptopów po serwery produkcyjne, niezależnie od platformy.
- **Szybkość i Elastyczność** - Kontenery uruchamiają się szybko i efektywnie, co ułatwia skalowanie aplikacji i eksperymentowanie z nowymi wersjami i konfiguracjami.
- **Izolacja Aplikacji** - Konteneryzacja zapewnia izolację aplikacji, co zmniejsza konflikty zależności i ułatwia zarządzanie wersjami.
- **Optymalizacja Zasobów** - Dzięki współdzieleniu jądra systemu operacyjnego i lekkiej architekturze, Docker umożliwia lepsze wykorzystanie zasobów w porównaniu do maszyn wirtualnych.



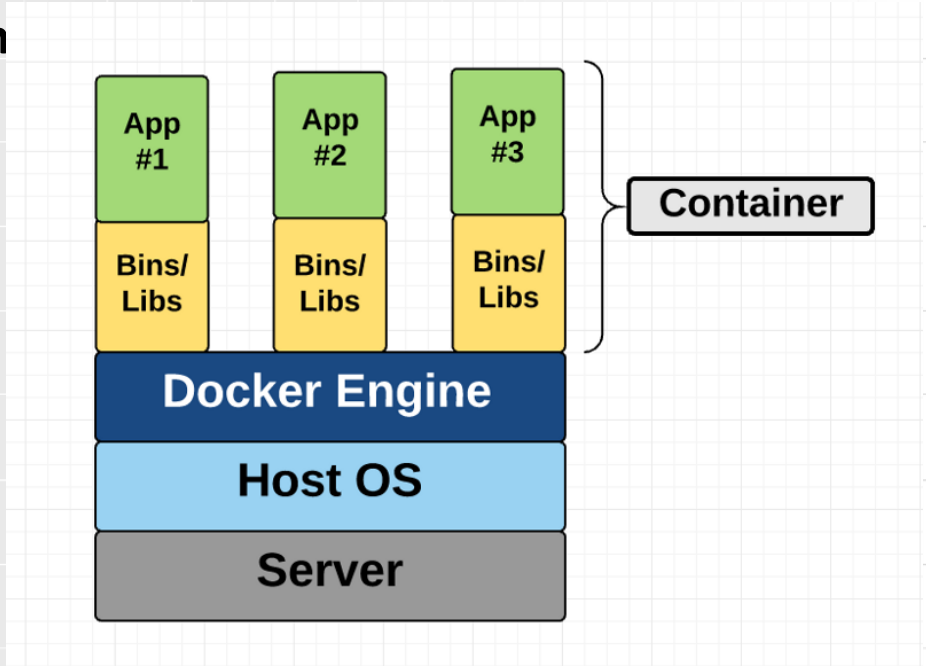
Docker – kluczowe korzyści

- **Zwiększona efektywność dewelopmentu i wdrożeń** - Docker upraszcza i przyspiesza workflow deweloperski od tworzenia przez testowanie po wdrożenie aplikacji.
- **Łatwość zarządzania** - **Zarządzanie** aplikacjami w kontenerach jest **prostsze** dzięki użyciu narzędzi Docker oraz ekosystemowi rozwiązań wspierających konteneryzację.
- **Spójność środowisk** - Docker gwarantuje, że aplikacja będzie **działać tak samo niezależnie od miejsca uruchomienia**.



Kontener [1]

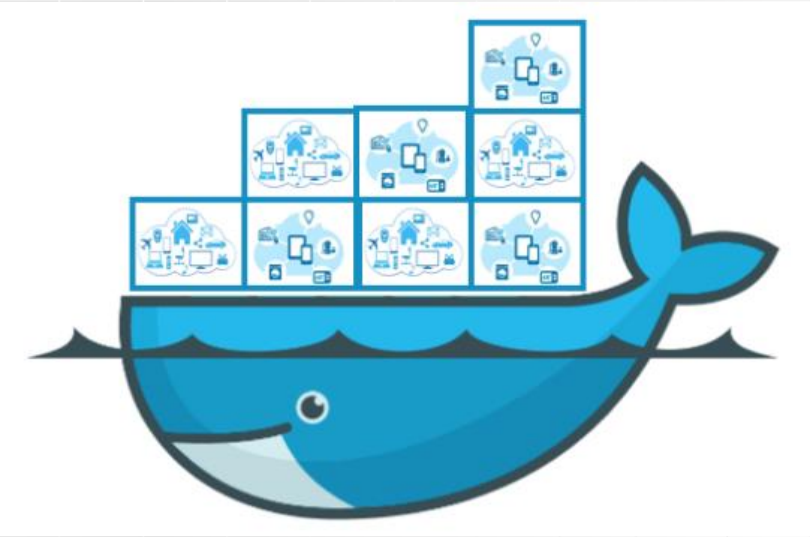
- Kontener to **proces uruchomiony w izolowanym środowisku na maszynie gospodarza**, który **jest odizolowany od wszystkich innych procesów działających na tej maszynie gospodarza**.



https://docs.docker.com/get-started/

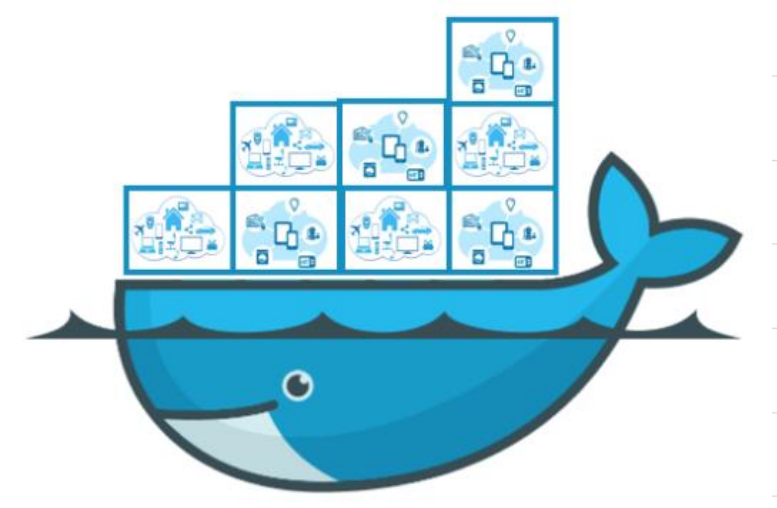
Kontener [2]

- Kontenery **izolują procesy aplikacji** na poziomie systemu operacyjnego hosta,
- Wykorzystują **funkcje jądra systemu Linux, takie jak cgroups i namespaces**,
- Aplikacja w kontenerze ma **własny widok systemu plików, sieci, użytkowników** itd., ale wciąż **korzysta z tego samego jądra systemu operacyjnego**, na którym działa kontener,
- **Nie zawiera własnego jądra systemu operacyjnego ani nie emuluje sprzętu**
- Jest wykonalną instancją obrazu,



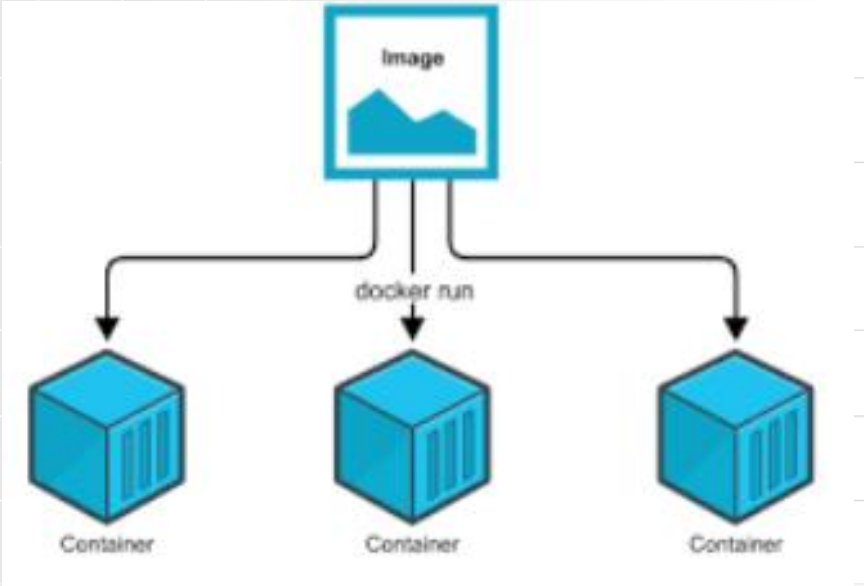
Kontener [3]

- Ponieważ kontenery **współdzielą jądro** systemu operacyjnego z hostem i **nie muszą emulować sprzętu**, są **znacznie lżejsze i szybsze w uruchamianiu** niż maszyny wirtualne
- Kontenery **startują niemal natychmiastowo**, co jest dużą zaletą w środowiskach, gdzie szybkie skalowanie i elastyczność są kluczowe.
- Kontenery można opisać jako **formę wirtualizacji na poziomie systemu operacyjnego**, w przeciwieństwie do tradycyjnej **wirtualizacji opartej na maszynach wirtualnych, które emulują całe środowisko sprzętowe**. Dzięki temu kontenery mogą być **bardziej efektywne pod względem zużycia zasobów i szybkości** działania.



Obraz

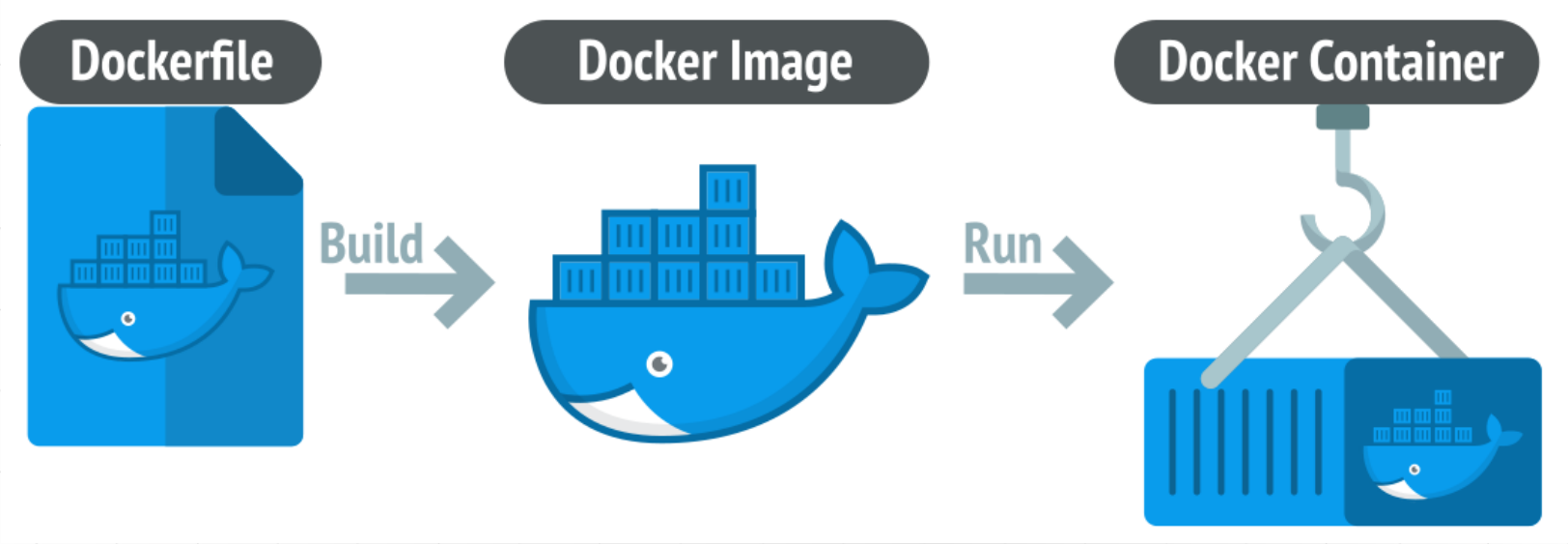
- Działający kontener korzysta z izolowanego systemu plików. **Ten izolowany system plików jest dostarczany przez obraz**, który musi zawierać wszystko, co potrzebne do uruchomienia aplikacji - wszystkie zależności, konfiguracje, skrypty, pliki binarne itp. Obraz zawiera także inne konfiguracje kontenera, takie jak zmienne środowiskowe, polecenie domyślne do uruchomienia i inne metadane.



https://docs.docker.com/get-started/

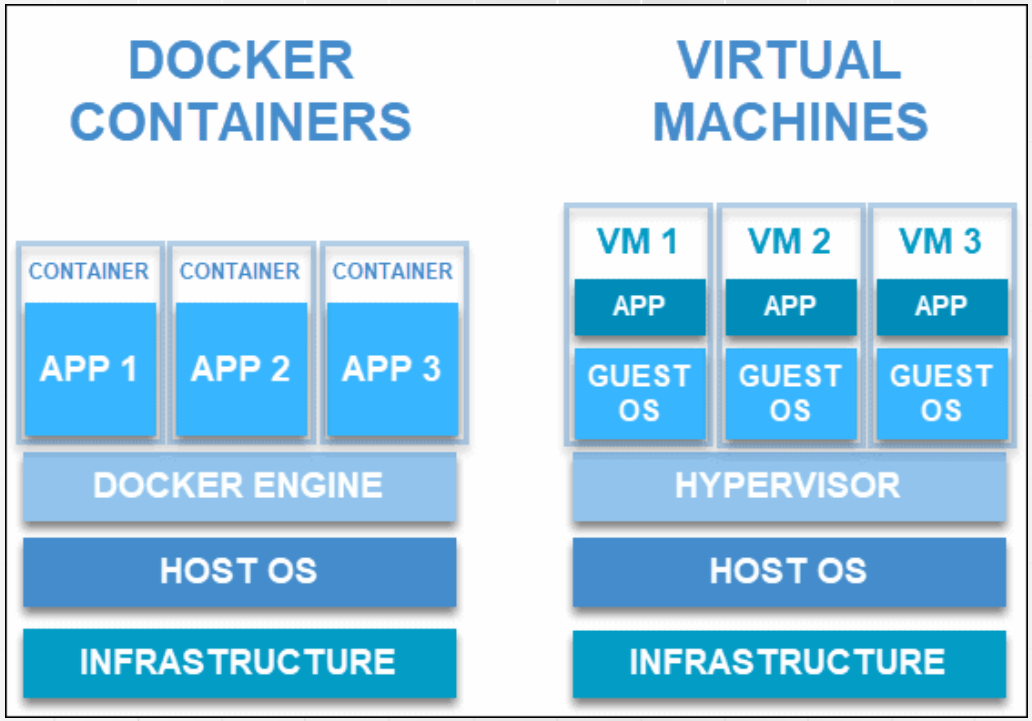
Obraz

- Definiuje mini system, na którym będziemy uruchamiać daną aplikację.
- Tworzenie takiego obrazu jest podobne do stawiania serwera - instalujemy odpowiednie paczki, konfigurujemy je, kopiujemy pliki itd.
- Składa się z kilku warstw **system operacyjny** np.: Ubuntu lub Alpine, **warstwy związane z uruchamianą aplikacją** np.: zainstalowana Java/Python, **nasza aplikacja**



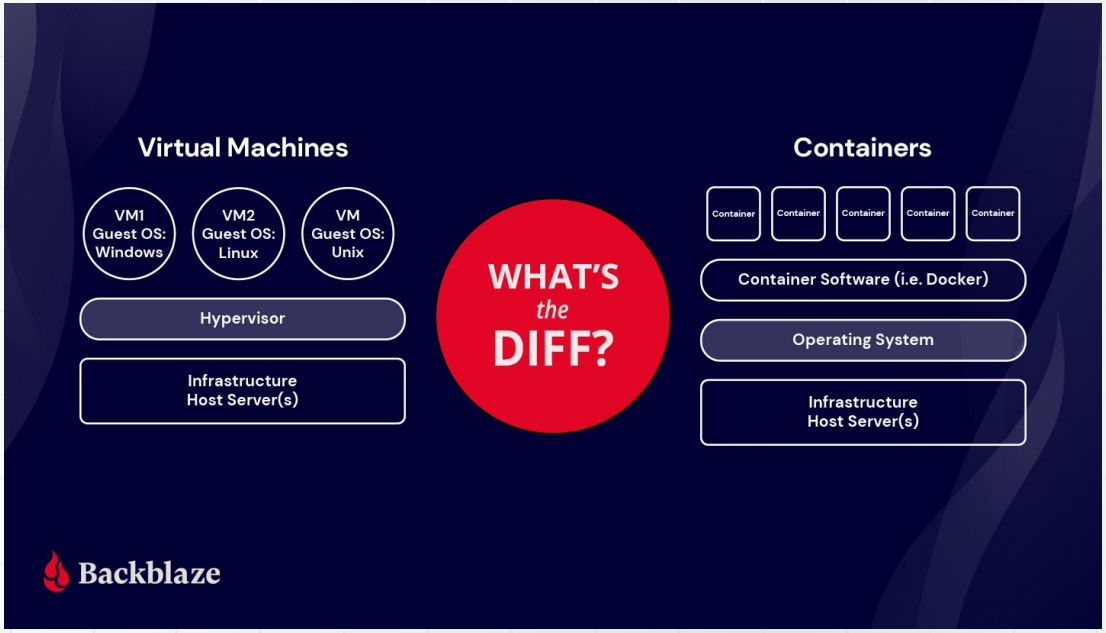
Maszyna wirtualna

- Emulowany komputer z **własnym systemem operacyjnym**
- Uruchomiony na oprogramowaniu zwanym **hypervisorem**, działającym na **fizycznym sprzęcie komputerowym**
- P**ozwala na uruchomienie wielu systemów operacyjnych na jednym fizycznym serwerze**, każdy w **izolowanym środowisku**.



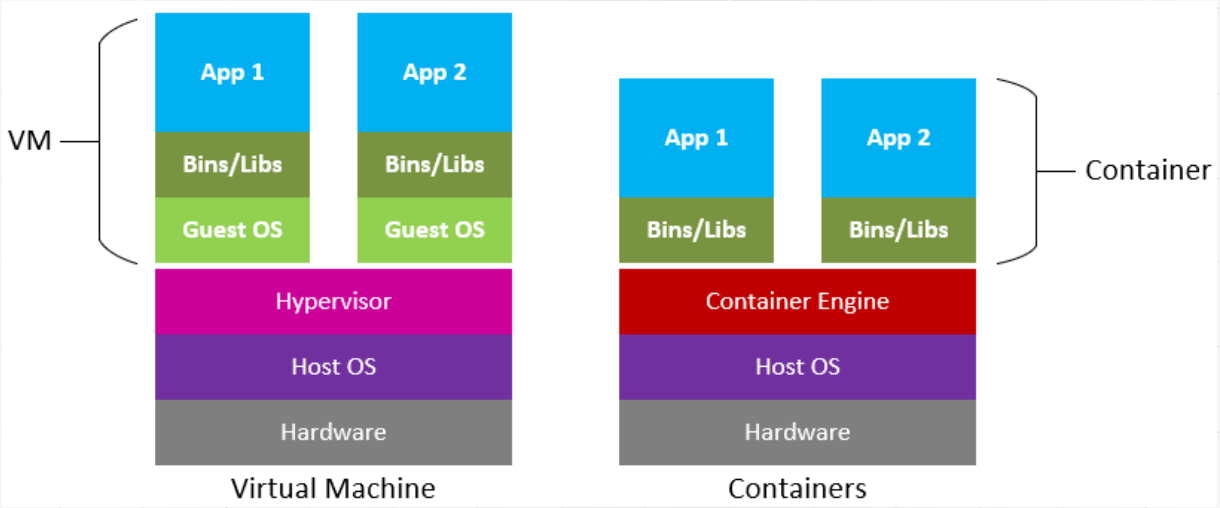
Maszyna wirtualna - Hypervisor

- Program lub warstwa oprogramowania **umożliwiająca hostowanie jednej lub wielu VM**, zarządzająca dystrybucją zasobów sprzętowych do każdej maszyny wirtualnej. Wyróżnia się dwa typy:
  - **Typ 1 (bare-metal)**: Oprogramowanie działa bezpośrednio na sprzęcie fizycznym (np. VMware ESXi, Microsoft Hyper-V).
  - **Typ 2 (hosted)**: Oprogramowanie działa na systemie operacyjnym hosta (np. Oracle VirtualBox, VMware Workstation).



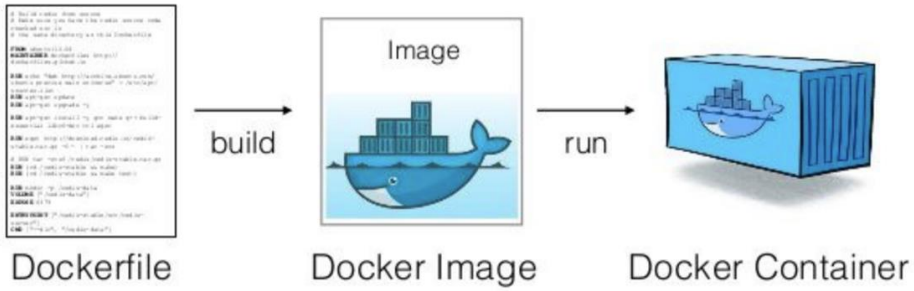
Maszyna wirtualna – wady i zalety

- **Izolacja** - Maszyny wirtualne **oferują pełną izolację od systemu hosta i od siebie nawzajem**, co zapewnia bezpieczeństwo i elastyczność.
- **Zasoby** - **Współdzielenie zasobów fizycznego hosta**; możliwość przydzielania zasobów (CPU, pamięci RAM, przestrzeni dyskowej) dla każdej VM.
- **Przenośność - Łatwość migracji maszyn** wirtualnych między hostami fizycznymi.
- **Nadmiarowość i Odporność na Awarię** - Ułatwia tworzenie środowisk wysokiej dostępności i planowanie ciągłości działania.
- **Zużycie Zasobów** - Większe zużycie zasobów w porównaniu do kontenerów, **ze względu na potrzebę uruchamiania pełnego systemu operacyjnego** dla każdej VM.
- **Startup Time** - Dłuższy czas uruchamiania niż w przypadku kontenerów.



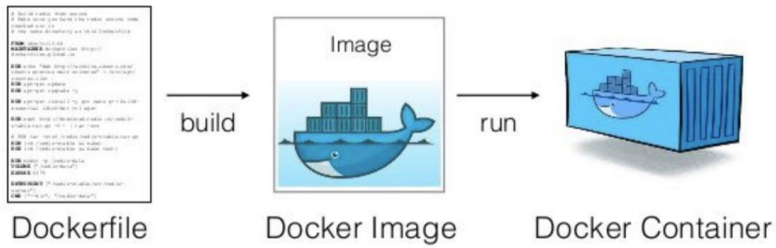
Dockerfile

- Zwykły plik tekstowy o nazwie Dockerfile (brak rozszerzeń, sama nazwa)
- Zawiera szereg instrukcji, dzięki którym pozwalna stworzyć w pełni funkcjonalny obraz który będzie zawierał wszystko, co jest potrzebne do uruchomienia aplikacji.
- Dockerfile składa się z par (instrukcja, argumenty) dla instrukcji
- Opisuje on dokładnie, z jakich elementów powinno składać się środowisko wykonawcze dla umieszczonej w obrazie aplikacji.



Dockerfile – wybrane instrukcje

- **FROM**: określenie obrazu bazowego.
- **RUN**: wykonanie poleceń w nowej warstwie na wierzchu bieżącego obrazu i zatwierdzenie wyników.
- **COPY/ADD**: kopiowanie plików i katalogów do obrazu.
- **WORKDIR**: ustawienie katalogu roboczego dla instrukcji **RUN, CMD, ENTRYPOINT, COPY, i ADD**.
- **CMD**: ustawienie domyślnego polecenia lub opcji, które zostaną wykonane podczas uruchamiania kontenera.
- **EXPOSE**: informowanie Dockera, że kontener nasłuchuje na określonych portach w trakcie działania.
- **ENV**: ustawienie zmiennych środowiskowych.

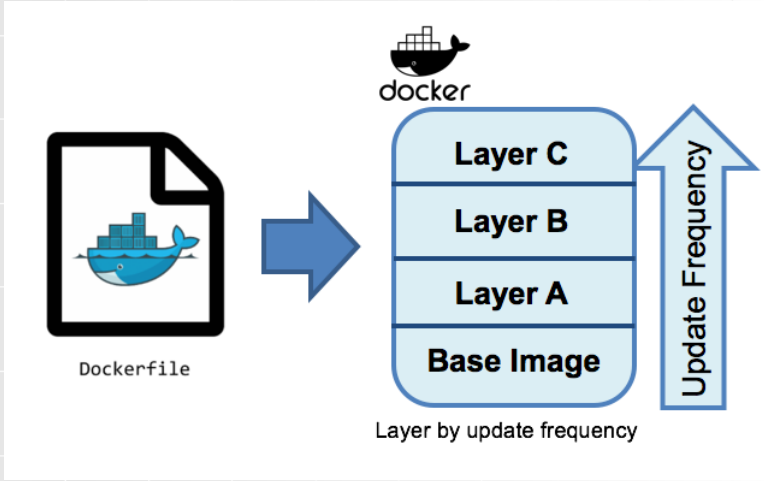


Dockerfile - przykład

```
# Obraz bazowy
FROM node:alpine

# Instalacja paczek
RUN npm install

# Komenda startowa
CMD ["npm", "start"]`
```

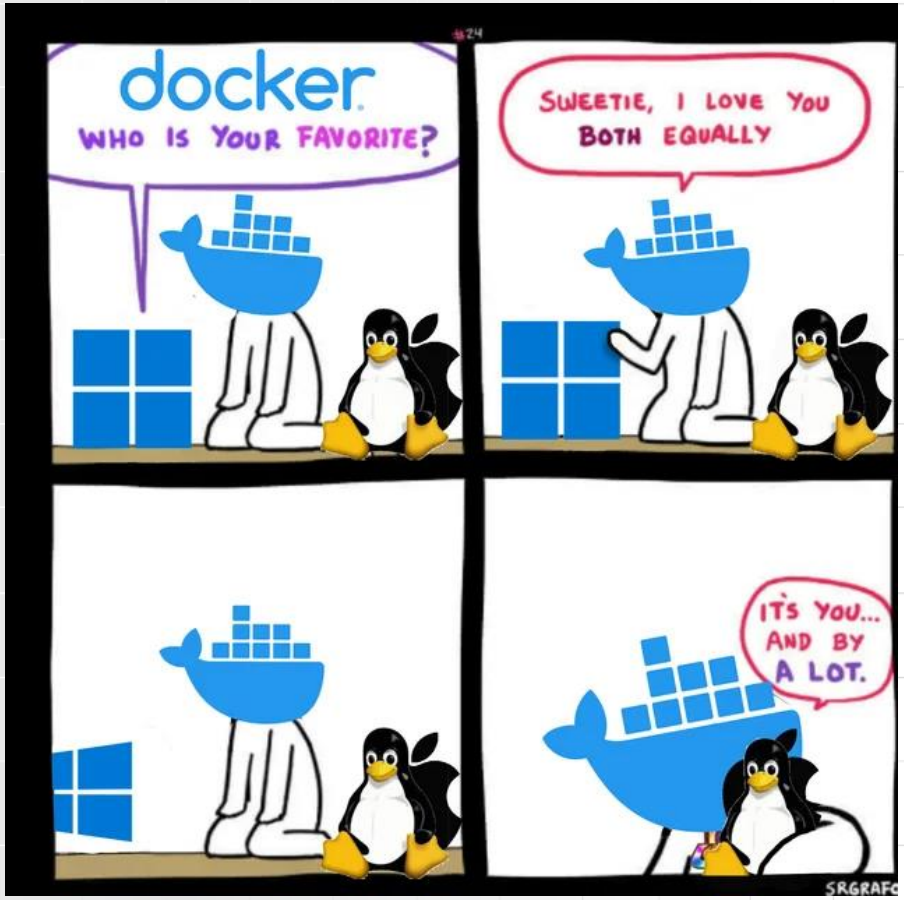
Dockerfile - Obraz bazowy

- Jest swego rodzaju system operacyjnym dla tworzonego przez obrazu.
- Jest to baza, do której będziemy dodawać kolejne aplikacje (np. biblioteki) oraz nasz własny kod.
- Można użyć **„gołego" systemu** operacyjnego, np. Ubuntu lub Alpine. **W takim przypadku musimy samodzielnie zainstalować wszystkie potrzebne nam narzędzia** (np. Node.js).
- Można użyć **gotowego obrazu**, który to posiada już większość tych **narzędzie preinstalowanych**. Dzięki temu nasz **plik Dockerfile będzie dużo mniejszy i łatwiejszy do zarządzania**.



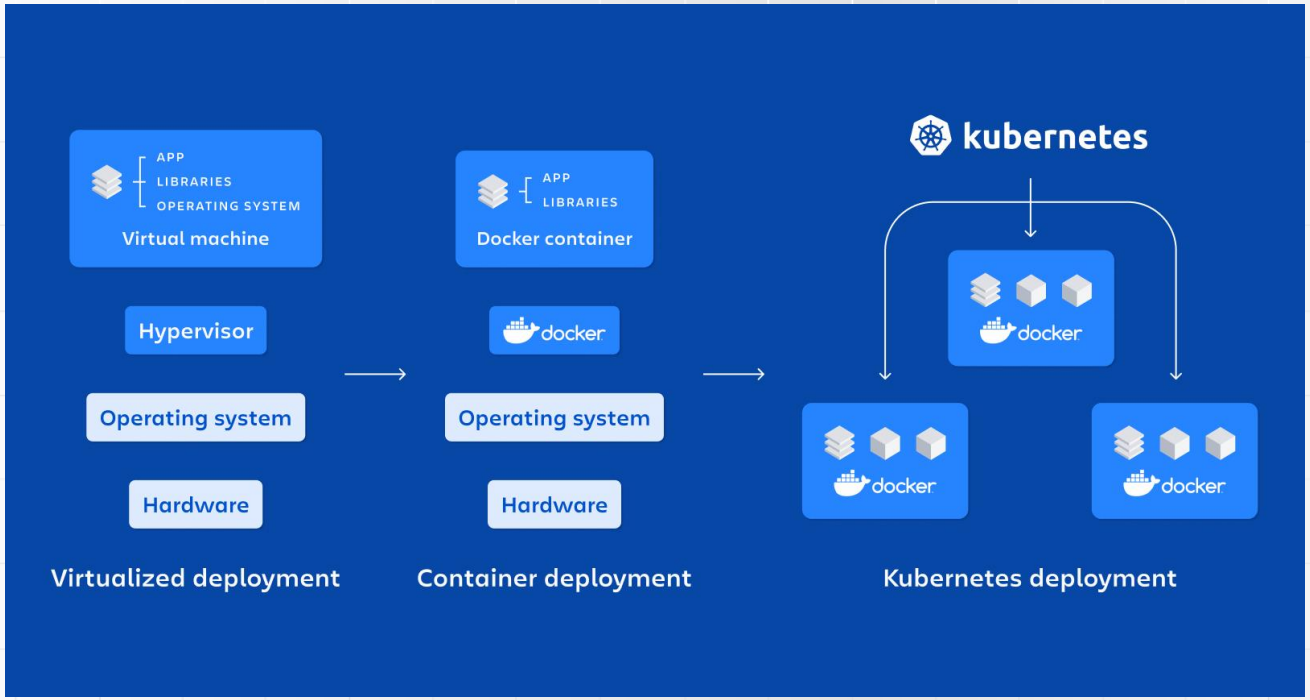
.dockerignore

- Określia które pliki nie powinny być kopiowane do obrazu pliku
- Pozwala na zmniejszenie rozmiaru obrazu
- Działa podobnie jak .gitignore



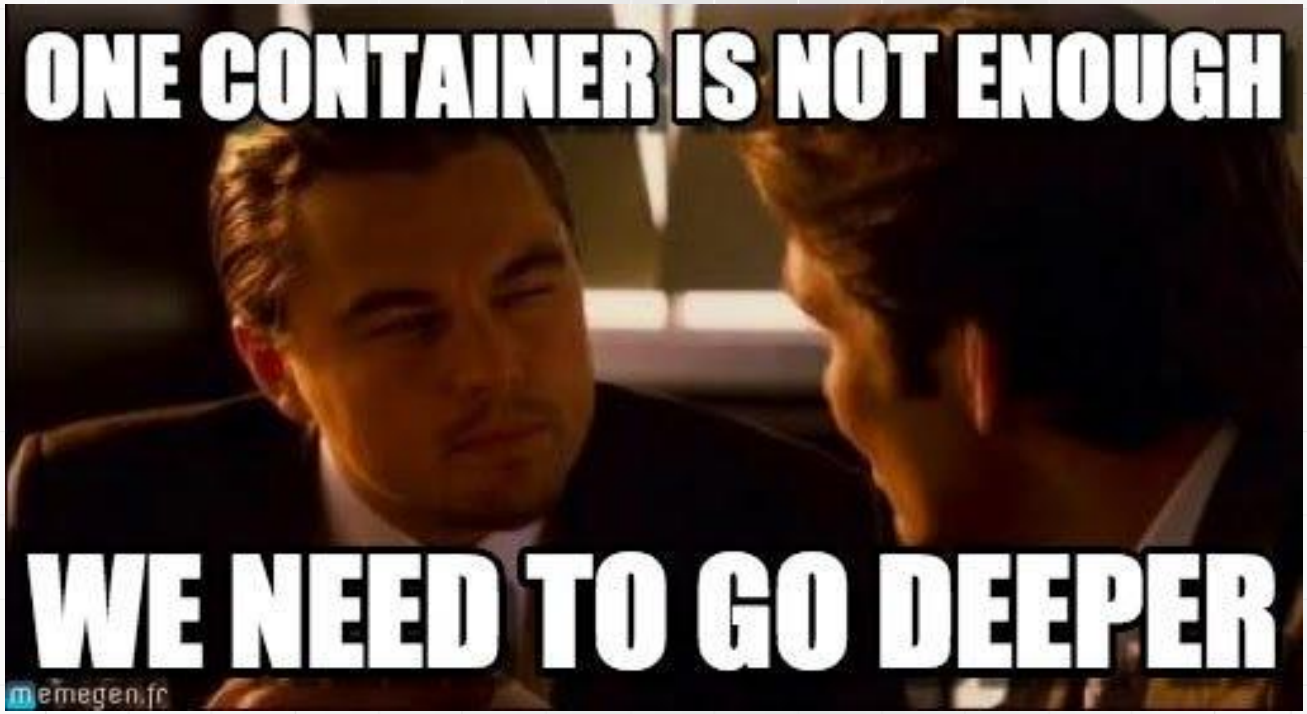
Zarządzanie kontenerami

- To proces kompleksowego **zarządzania życiem i działaniem kontenerów aplikacji w środowisku informatycznym**
- Obejmuje **szeroki zakres działań**, od **rozwoju aplikacji** przez jej **wdrożenie**, **skalowanie**, **aktualizacje**, **monitorowanie**, aż po **zapewnienie bezpieczeństwa**.



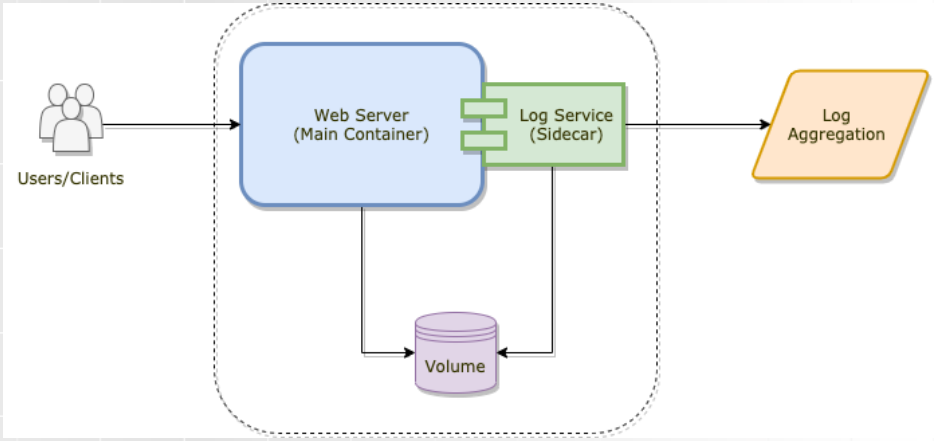
Zarządzanie kontenerami - orkiestracja

- Proces **zarządzania cyklem życia wielu kontenerów**, które razem tworzą aplikacje w złożonych środowiskach.
- Orkiestracja kontenerów **pozwala na automatyczne rozmieszczanie**, **skalowanie** i **zarządzanie stanem kontenerów**.



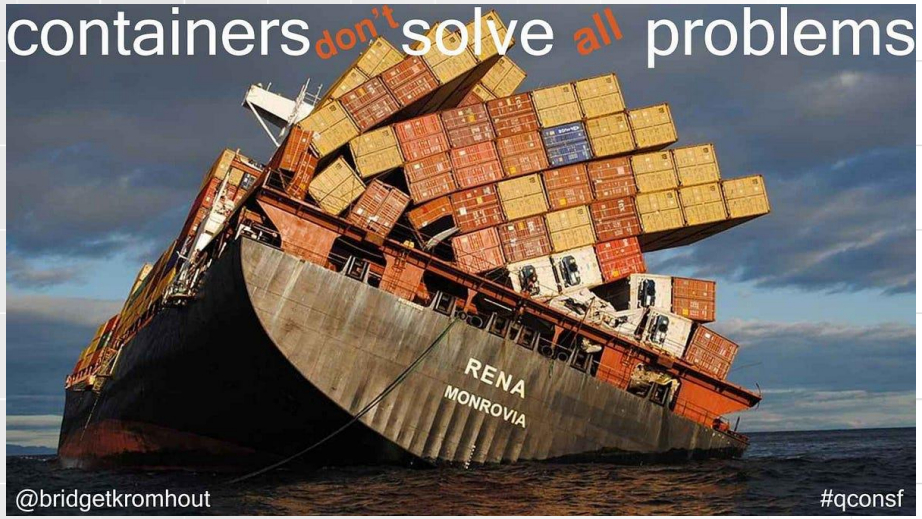
Zarządzanie kontenerami - monitorowanie i logowanie

- Monitoring stanu kontenerów i aplikacji, a także zbieranie i analiza logów, są kluczowe dla utrzymania wydajności, dostępności i bezpieczeństwa aplikacji.
- Narzędzia do zarządzania kontenerami często integrują się z systemami monitorowania i logowania, dostarczając wgląd w działanie aplikacji i ułatwiając diagnostykę problemów.



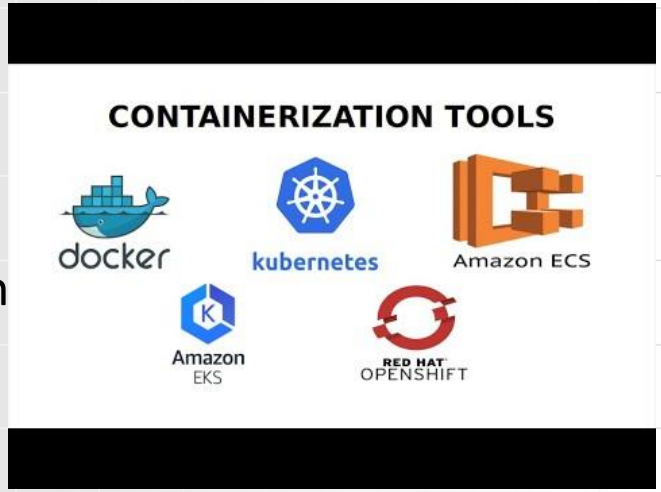
Zarządzanie kontenerami - sieciowanie

- Zarządzanie siecią w środowisku kontenerów, w tym izolacja sieci, balansowanie obciążenia, odkrywanie usług i zarządzanie komunikacją między kontenerami.
- Jest kluczowe dla zapewnienia skutecznej komunikacji wewnątrz złożonych aplikacji mikrousługowych.

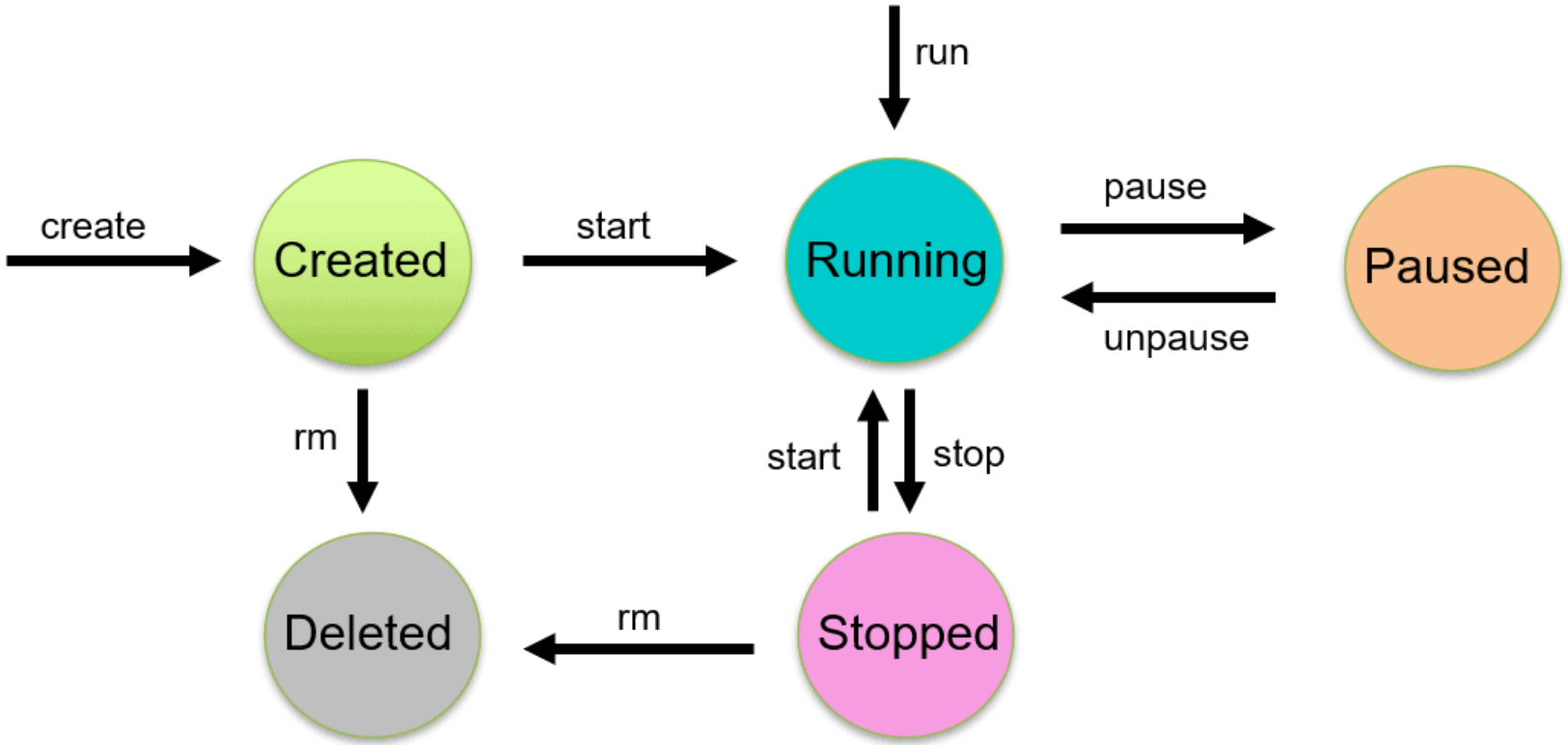


Zarządzanie kontenerami - narzędzia

- **Docker Swarm** - Narzędzie do klastrowania i zarządzania kontenerami Docker, zapewniające natywne zarządzanie klastrem Docker,
- **Kubernetes** – Otwarto źródłowy system do automatycznego wdrażania, skalowania i zarządzania aplikacjami kontenerowymi,
- **OpenShift** - Platforma aplikacji kontenerowych od Red Hat, oparta na Kubernetes, oferująca dodatkowe funkcje zarządzania i bezpieczeństwa,
- **Amazon ECS/AKS/GKE** - Usługi zarządzania kontenerami dostarczane przez głównych dostawców chmury: Amazon Elastic Container Service, Azure Kubernetes Service i Google Kubernetes Engine.

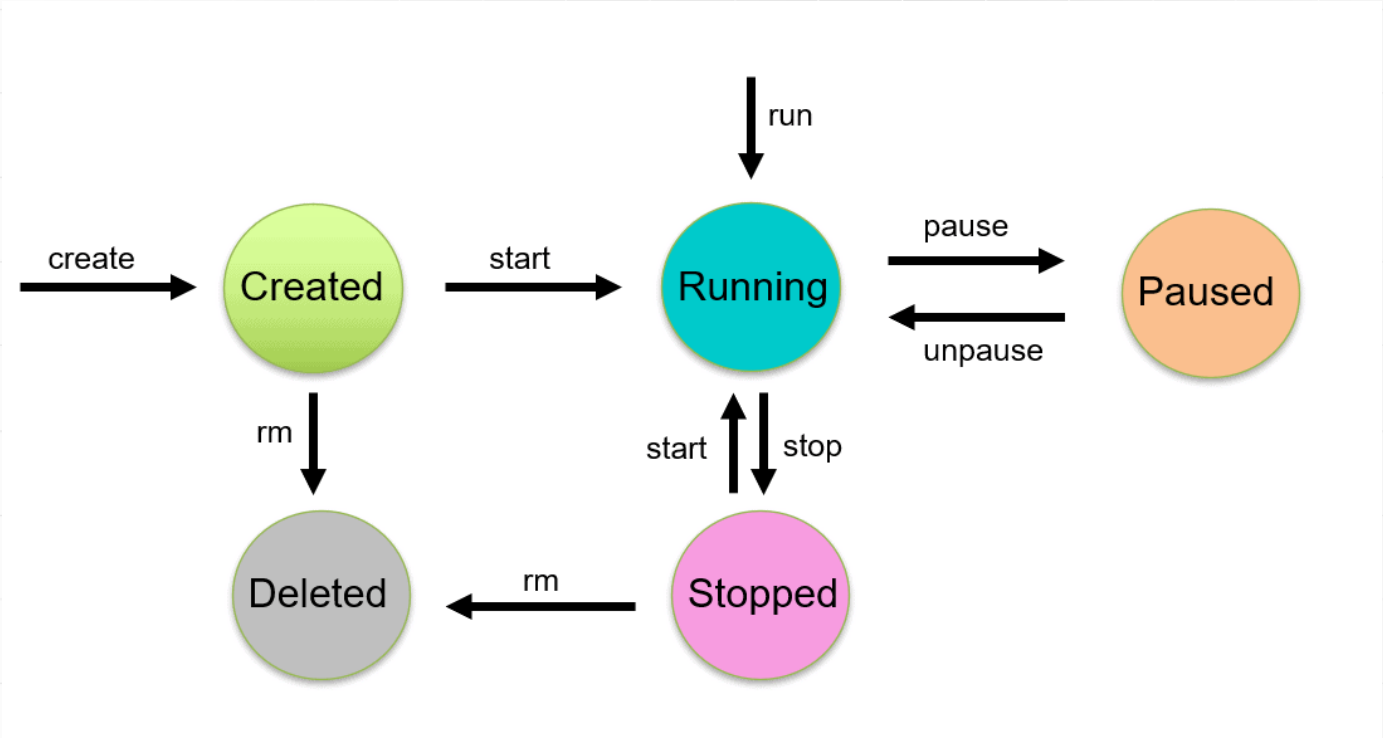


Cykl życia obrazów Docker



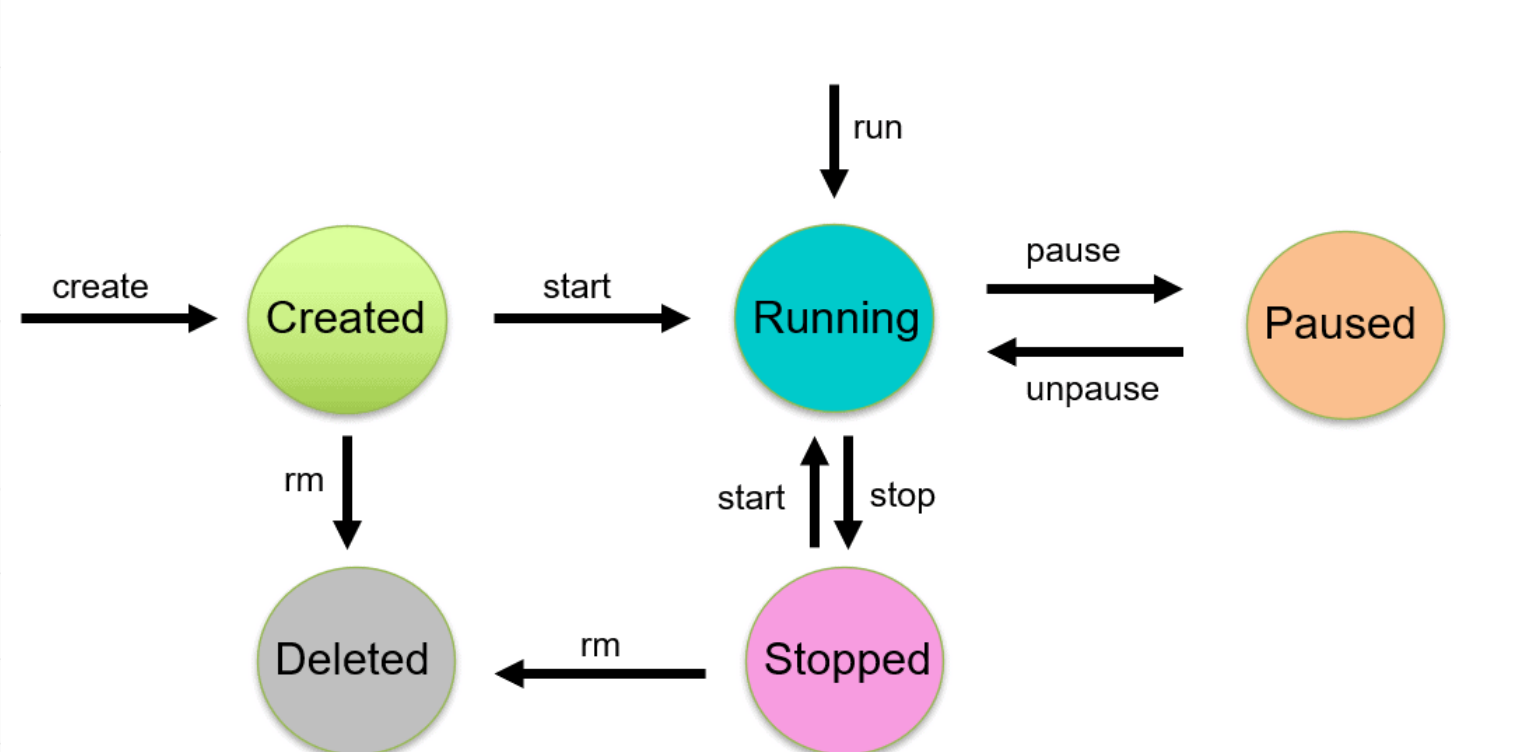
Podstawowe komendy - docker build

- docker build PATH | URL
- Buduje **obraz** Dockera z pliku Dockerfile



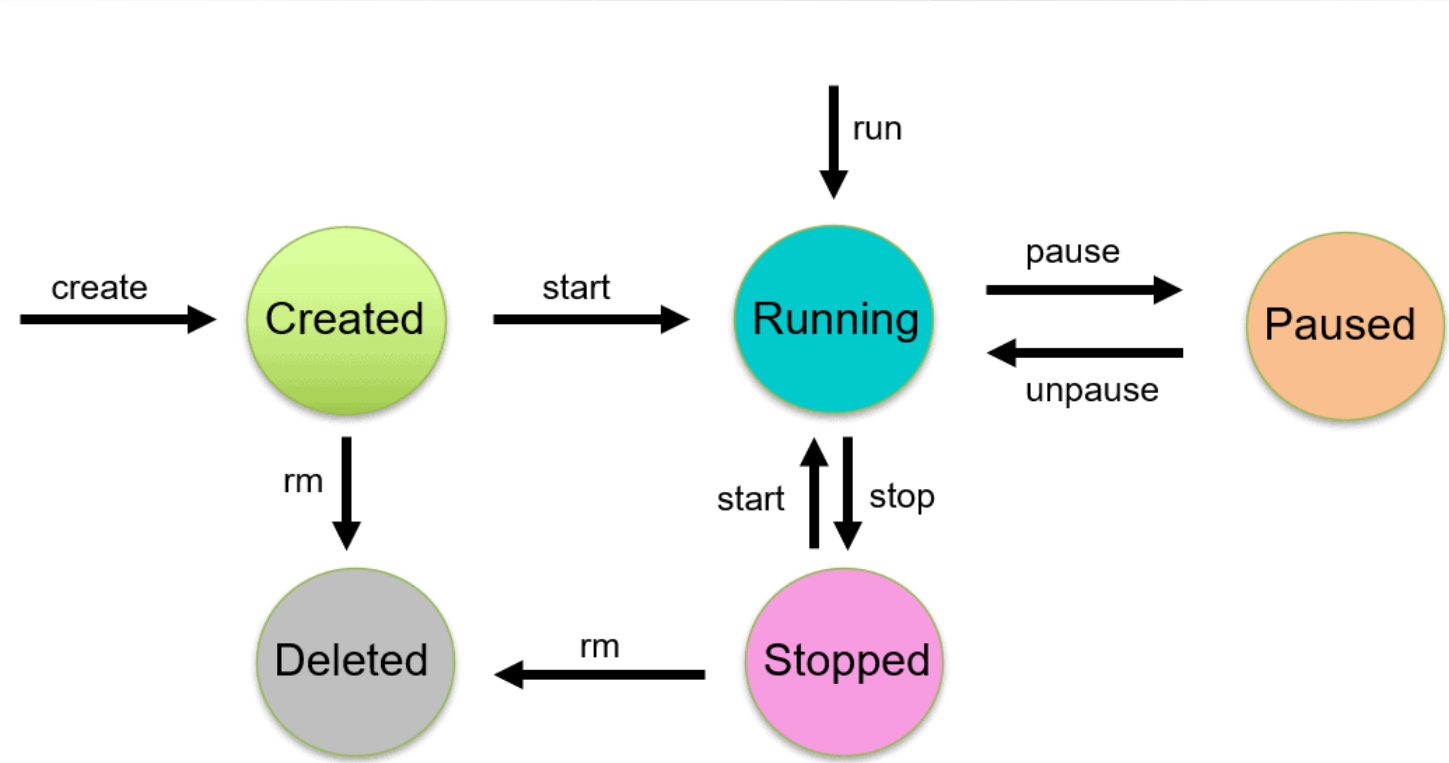
Podstawowe komendy - docker create

- docker create --name <nazwa kontenera> <nazwa obrazu>
- Powoduje utworzenie nowego **kontenera** Docker z określonym obrazem dockerowym.



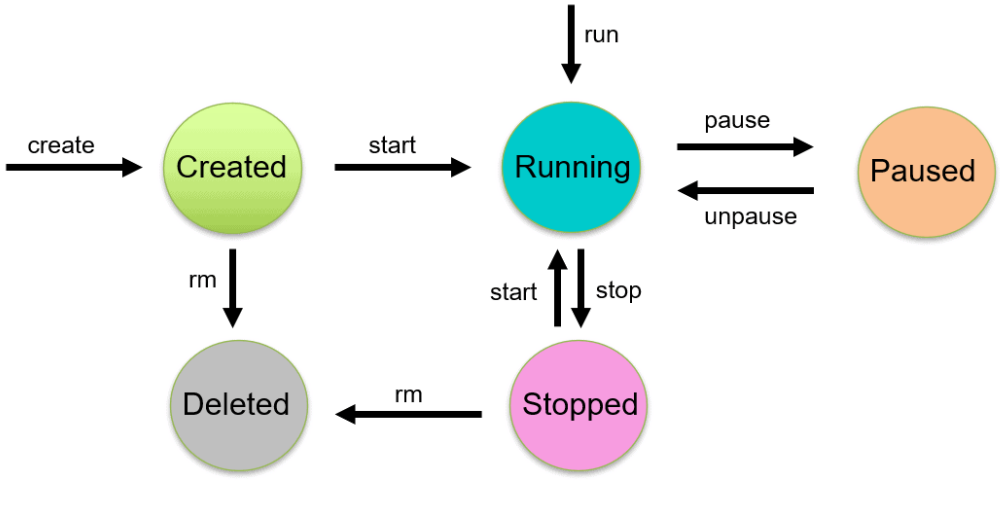
Podstawowe komendy - docker start

- docker start <nazwa kontenera>
- Uruchomia zatrzymany kontener



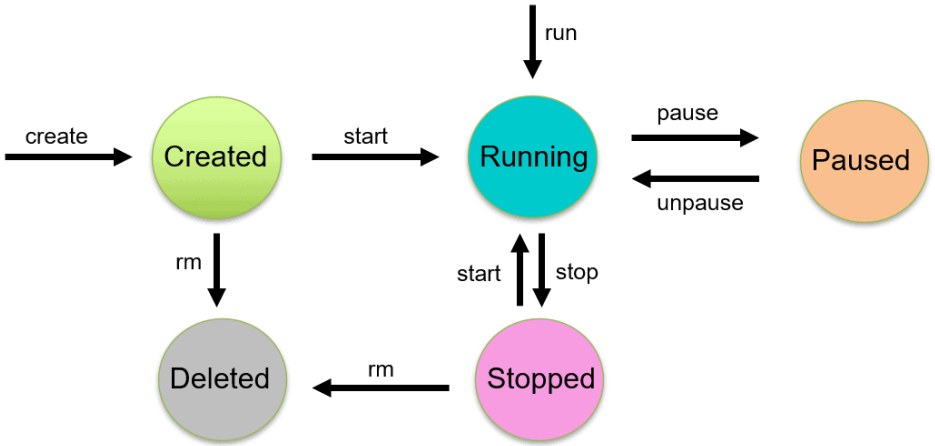
Podstawowe komendy - docker run

- docker run <nazwa_obrazu>
- Pobiera obraz i tworzy z niego kontener.
- Wykona działanie zarówno polecenia „docker create”, jak i „docker start”.
- Utworzy nowy kontener i uruchomi obraz w nowo utworzonym kontenerze.



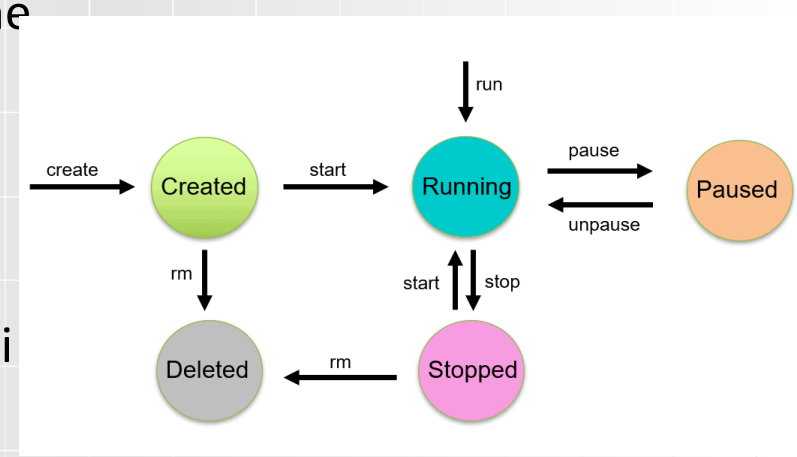
Podstawowe komendy - docker pause/unpause

- docker pause <nazwa kontenera>
- docker unpause <nazwa kontenera>
- Służy do tymczasowego zatrzymania wszystkich procesów wewnątrz określonego kontenera Docker.
- Kiedy kontener jest wstrzymany (paused), wszystkie procesy działające wewnątrz tego kontenera są zamrażane



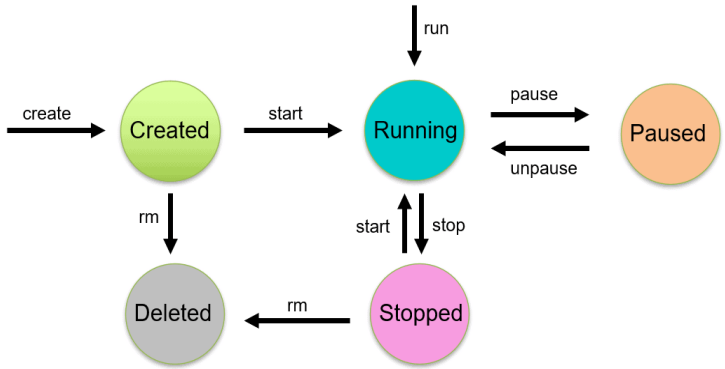
Podstawowe komendy - docker stop

- docker stop <nazwa kontenera>
- Zatrzymuje działający kontener.
- Proces zatrzymania kontenera **odbywa się w kilku etapach** i jest zaprojektowany w taki sposób, aby umożliwić bezpieczne zakończenie działania aplikacji i kontenera.
- **Preferowany i bezpieczny sposobów zatrzymywania kontenerów**, gdyż oferuje aplikacjom możliwość czystego i kontrolowanego zamknięcia, co jest szczególnie ważne w środowiskach produkcyjnych i dla krytycznych aplikacji.
- **Zasoby systemowe** (np. pamięć, CPU), które były używane przez kontener, **zostają zwolnione**.



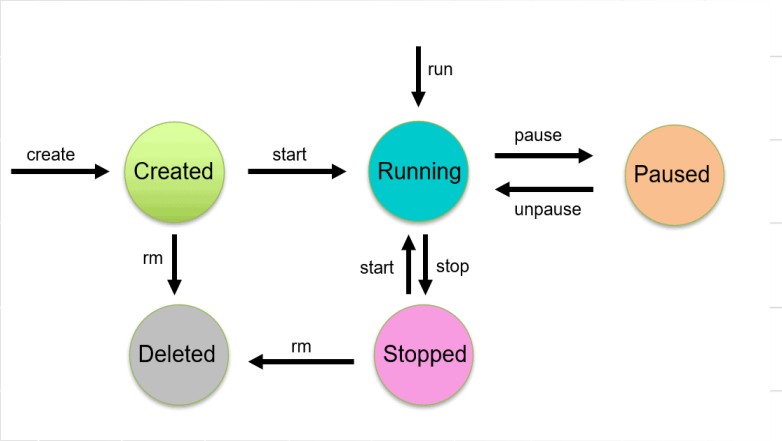
Podstawowe komendy - docker rm

- docker rm <nazwa kontenera>
- Używane do usuwanięcia kontenera z systemu
- Zanim kontener zostanie usunięty, musi być zatrzymany.
- Gdy kontener jest zatrzymany, docker rm usuwa jego pliki z systemu Docker, w tym metadane kontenera i wszystkie pliki danych związane z kontenerem, które nie są przechowywane na woluminach zewnętrznych, chyba że zastosowano opcję --volumes, która usunie także przypisane woluminy.
- Po usunięciu kontenera, wszelkie zasoby systemowe (tj, pamięć, przestrzeń dyskowa itd.) zostają zwolnione



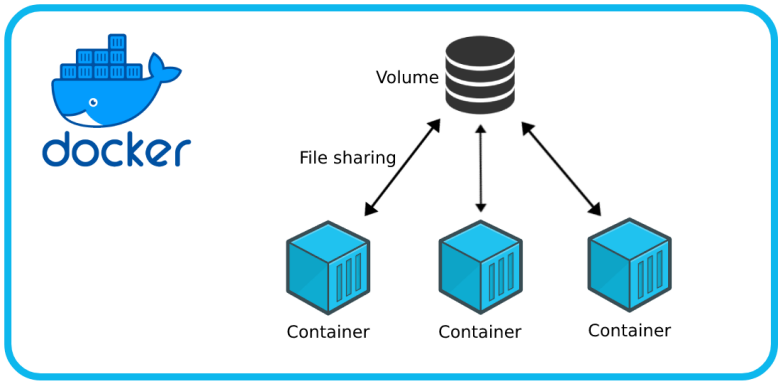
Podstawowe komendy - docker kill

- docker kill <nazwa kontenera>
- Używane do natychmiastowego zatrzymania działającego kontenera
- Używane głównie w sytuacjach, gdy kontener nie reaguje na normalne polecenia zatrzymania lub gdy jest potrzebne szybkie zwolnienie zasobów systemowych zajmowanych przez kontener.



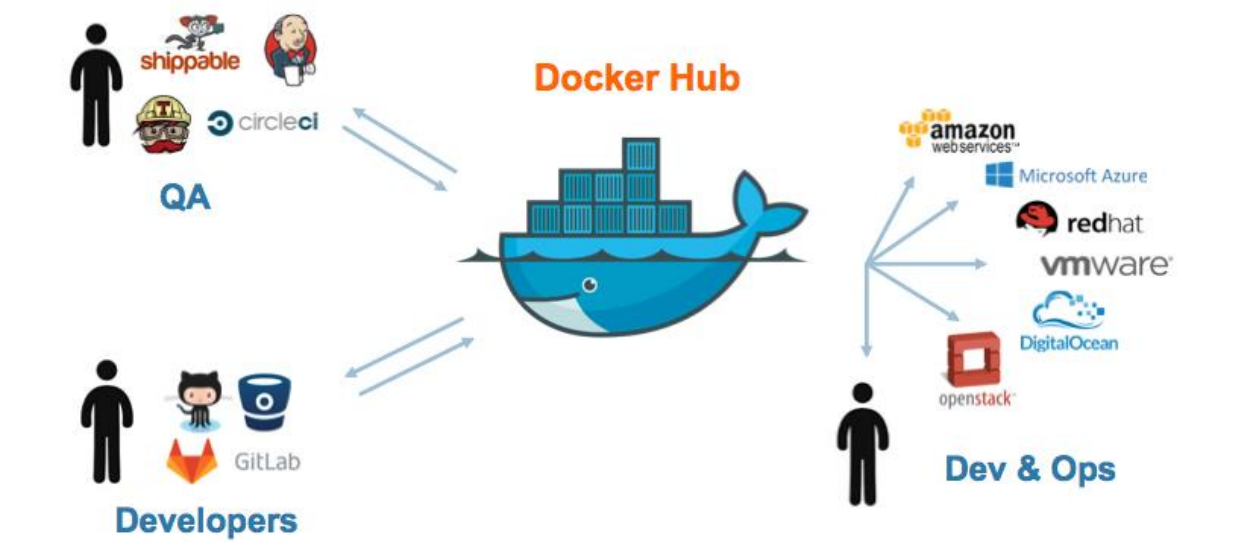
Wolumeny

- docker volume create – tworzy nowy wolumen
- docker volume ls – pozwala przeglądać listę wolumenów
- docker volume inspect – wyświwtla szczegółowe informacje o konkretnym wolumenie.
- docker volume rm – usuwa wolumen
- docker run -v moj_wolumen:/app/data moj_obraz
- To mechanizm trwałego przechowywania danych, używany do zarządzania danymi, które powinny przetrwać usunięcie lub restart kontenerów
- Mogą być bezpiecznie udostępniane między wieloma kontenerami



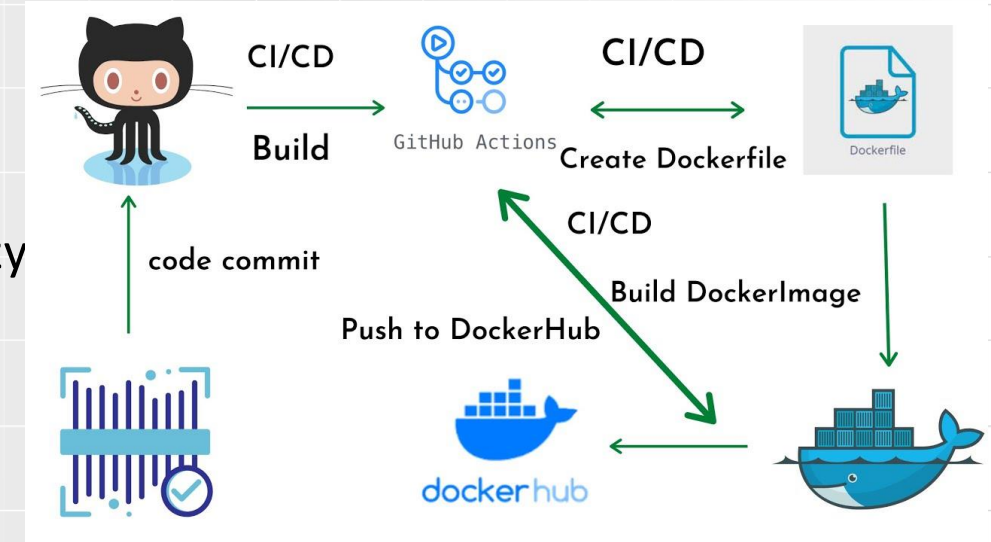
DockerHub

- To usługa chmurowa, która umożliwia **udostępnianie i zarządzanie repozytoriami obrazów** Docker. Jest to oficjalne centrum obrazów dla Docker, które **zawiera** zarówno **publiczne**, jak i **prywatne** repozytoria.



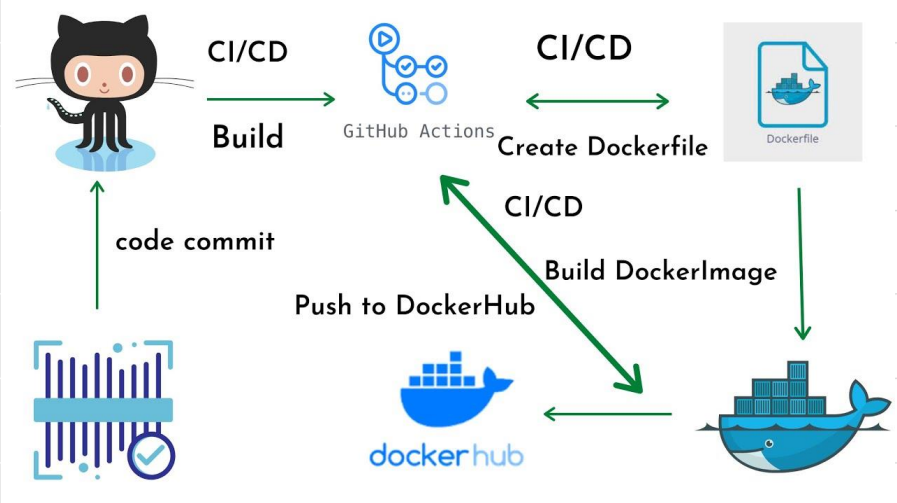
DockerHub – kluczowe funkcje [1]

- **Repozytoria** - Możliwość tworzenia **publicznych i prywatnych repozytoriów** do przechowywania obrazów Docker.
- **Automatyczne Budowanie - Integracja z GitHub i Bitbucket,** umożliwiająca automatyczne budowanie obrazów Docker przy każdej zmianie kodu w repozytorium.
- **Wersjonowanie Obrazów - Utrzymywanie różnych wersji (tagów) tego samego obrazu**, co ułatwia zarządzanie wersjami i rollbacki.



DockerHub – kluczowe funkcje [2]

- **Oficjalne Obrazy -** Dostęp do **szerokiej gamy oficjalnych obrazów Docker**, które są utrzymywane przez twórców oprogramowania i zweryfikowane przez Docker Inc.
- **Społeczność -** Dostęp do **obrazów stworzonych przez społeczność**, które można wykorzystać jako bazę dla własnych aplikacji.



Docker Compose

- Docker Compose to narzędzie do definiowania i uruchamiania wielokontenerowych aplikacji Docker z użyciem plików YAML,
- Umożliwia konfigurację usług, sieci i wolumenów w jednym pliku, upraszczając proces wdrażania złożonych aplikacji,
- Pozwala na automatyczne tworzenie sieci między kontenerami, co umożliwia komunikację między serwisami bez konieczności ręcznej konfiguracji.

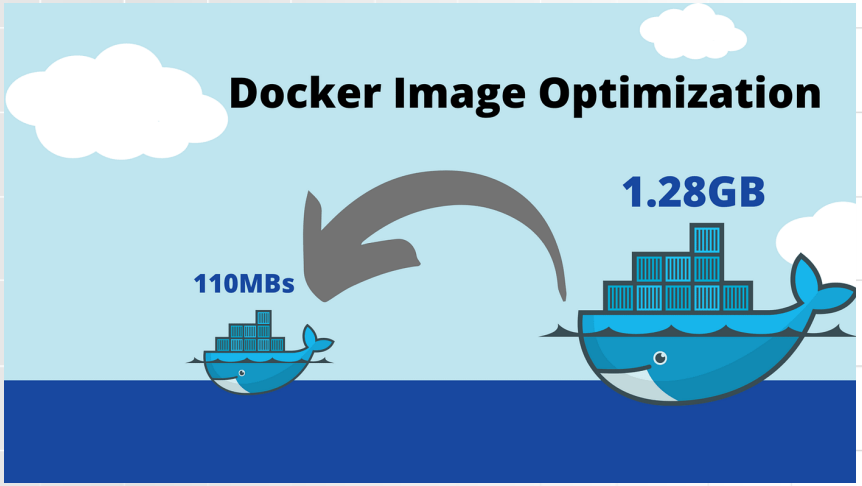
```yaml
version: '3.3'

services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:C:\Users\User\Desktop\dcompose
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: rootwordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

Optymalizacja obrazów Docker

- **Lekkie Obrazy Bazowe -** Używaj lekkich obrazów bazowych, takich jak **Alpine Linux**, które są mniejsze w rozmiarze i zawierają mniej zbędnych plików i bibliotek.
- **Oficjalne Obrazy -** Preferuj **oficjalne obrazy** dostępne na Docker Hub, które są **regularnie aktualizowane i optymalizowane** pod kątem **bezpieczeństwa** i **wydajności**.
- **Minimalizacja Liczby Warstw -** Łącz polecenia RUN w jednym łańcuchu, używając operatorów logicznych (&&) do zmniejszenia liczby warstw.

Optymalizacja obrazów Docker

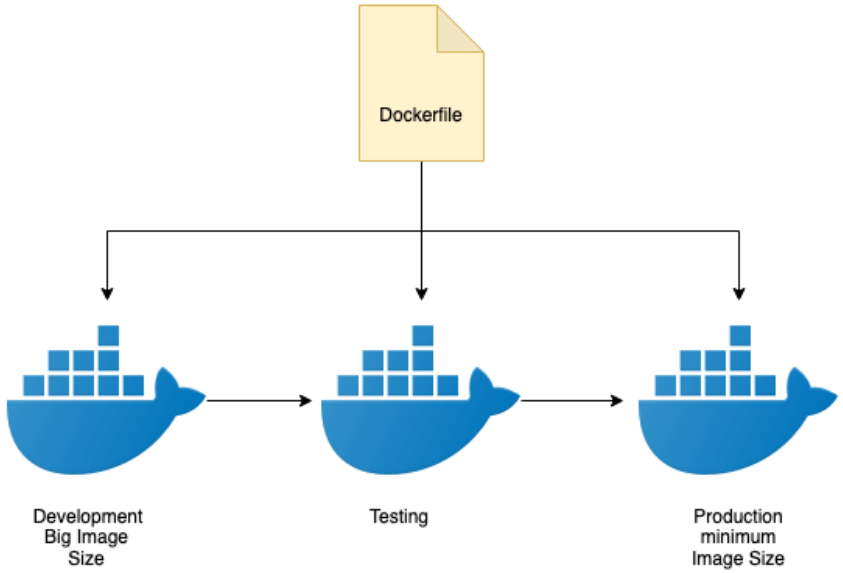
- **Usuwanie Niepotrzebnych Plików -** Oczyść niepotrzebne pliki i pakiety w tym samym poleceniu RUN, które je instaluje, aby uniknąć zapisywania ich w warstwach obrazu.
- **Ignorowanie Niepotrzebnych Plików -** Dodaj plik .dockerignore do swojego projektu, aby wykluczyć pliki i katalogi niepotrzebne w obrazie (np. zależności deweloperskie, pliki tymczasowe).



Optymalizacja obrazów Docker

- **Optymalizacja Budowania** - Użyj multi-stage builds do oddzielenia środowisk budowania i uruchomieniowych. Pozwala to na uwzględnienie tylko finalnych artefaktów w obrazie docelowym, redukując jego rozmiar.
- **Minimalizacja Zależności -** Instaluj tylko niezbędne pakiety i narzędzia, aby zmniejszyć rozmiar obrazu i ograniczyć powierzchnię ataku.

```
1  # Stage - Development
2  FROM Dev-Image as dev
3  . . .          stage 0
4  . . .
5
6  # Stage - Testing
7  FROM Test-Image as test
8  COPY --from=dev /src/app .
9  . . .          stage 1
10 . . .
11
12 # Stage - Production
13 FROM Minimum-Image as prod
14 COPY --from=dev /src/app .
15 . . .          stage 2
16 . . .
```



Bezpieczeństwo kontenerów

- Zapisywanie danych pofnych w zmiennych środowiskowych
- **docker run -e "PASSWORD=mojeHaslo" aplikacja**

Wrocław
University
of Science
and Technology

Dziękuję za uwagę