EGNYTE

Ewolucja infrastruktury: od monolitu do rozproszonej aplikacji w chmurze.

Michał Gręda
mgreda@egnyte.com

Od MVP do gotowego produktu

Michał Gręda
mgreda@egnyte.com

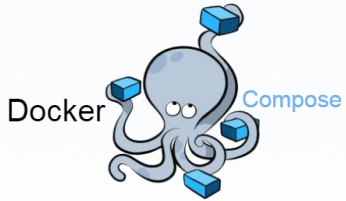
Agenda

01 **Kontenery**

02 **Docker**
Demo

03 **Docker Compose**
Demo

04 **Kubernetes**
Demo



Gdzie jesteśmy z naszym biznesem?

To już mamy

Pomysł

To już mamy

Pomysł

Wymagania

To już mamy

Pomysł

Wymagania

HLE

To już mamy

Pomysł

Wymagania

HLE

Refinement

To już mamy

Pomysł

Wymagania

HLE

Refinement

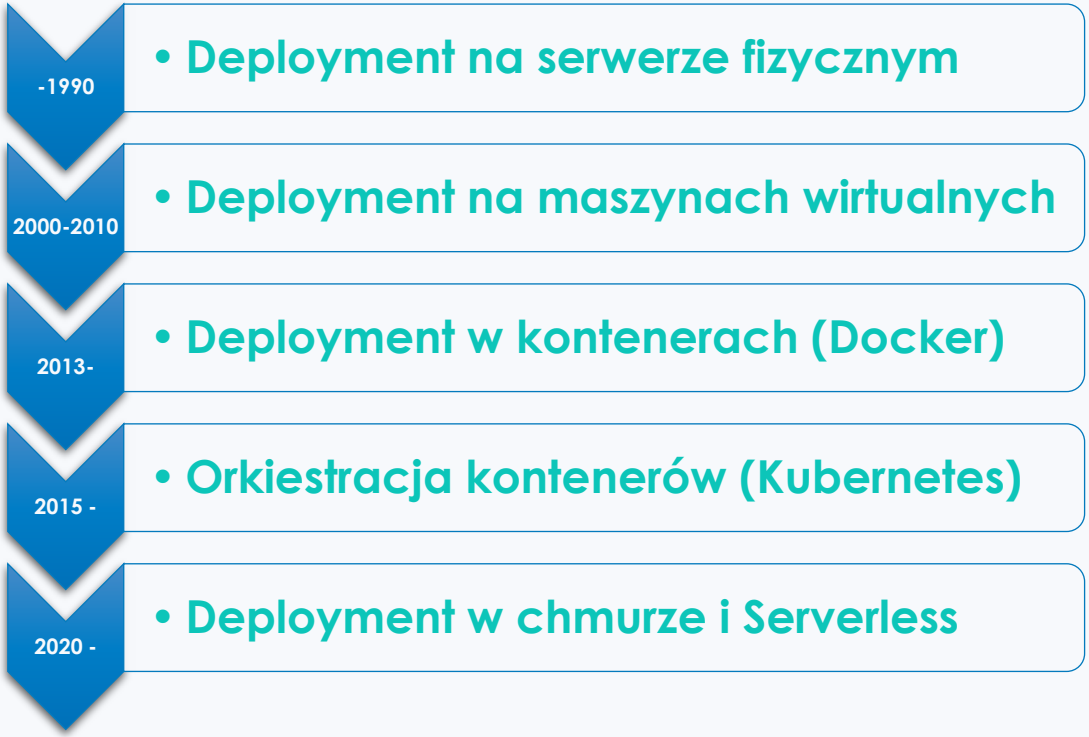
Development

MVP - DEMO

branch: main

Deployment aplikacji webowych - historia

Deployment aplikacji webowych – odrobina historii



SOA
Service-Oriented Architecture

Konteneryzacja

Dlaczego kontenery?

Powtarzalność środowiska

Ten sam kod działa tak samo na różnych systemach

Łatwość skalowania

Możliwość dynamicznego zwiększania liczby instancji aplikacji

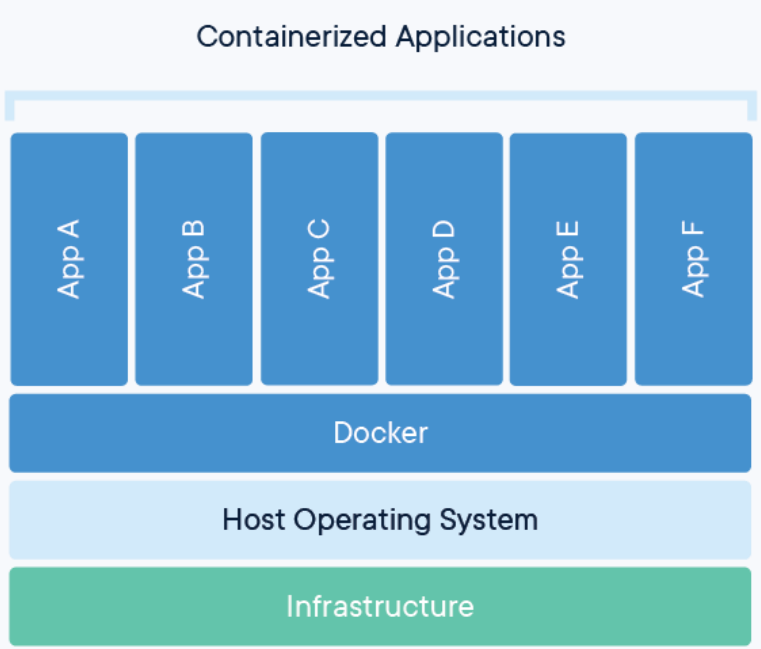
Izolacja aplikacji

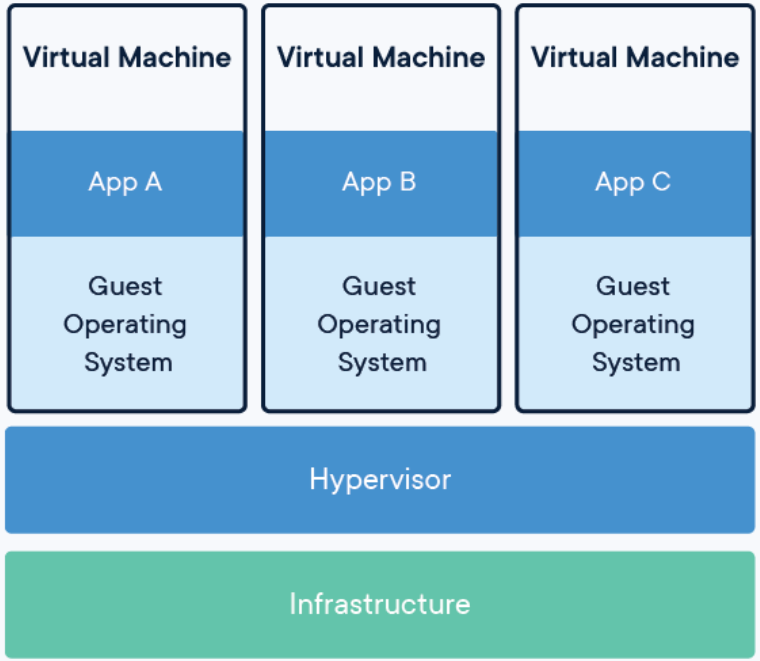
Brak konfliktów między zależnościami różnych usług

Wydajniejsze wykorzystanie zasobów

Mniejsze zużycie pamięci w porównaniu do maszyn wirtualnych

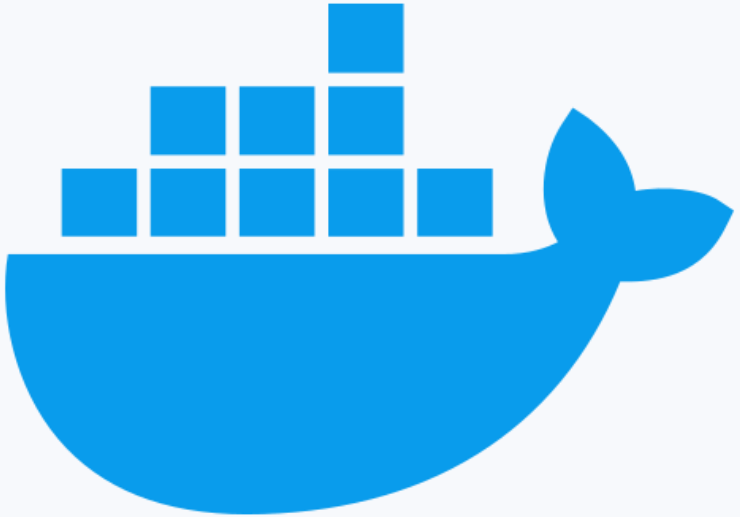
Dlaczego kontenery?





https://learning.rc.virginia.edu/notes/containers/overview-vms/

Docker



Czym jest Docker?

Czym jest Docker?

Docker to platforma do tworzenia, wdrażania i uruchamiania aplikacji w kontenerach.

DOCKERFILE

Plik zawierający instrukcje budowania obrazu.

OBRAZ

Niezmienny szablon aplikacji, zbudowany na podstawie Dockerfile.

KONTENER

Uruchomiona instancja obrazu, zawierająca aplikację.

REGISTRY

Repozytorium obrazów (np. Docker Hub, Amazon ECR).

DOCKERFILE

```
# 1. Wybór obrazu bazowego
FROM openjdk:17-jdk-slim

# 2. Zmienne środowiskowe
ENV SPRING_PROFILES_ACTIVE=prod \
APP_PORT=8080

# 3. Ustawienie katalogu roboczego
WORKDIR /app

# 4. Kopiowanie aplikacji do kontenera
COPY target/myapp.jar /app/myapp.jar
```

DOCKERFILE

```
# 5. Instalacja zależności (opcjonalnie)
RUN apt-get update && apt-get install -y curl && rm -rf
/var/lib/apt/lists/*

# 6. Wystawienie portu aplikacji
EXPOSE 8080

# 7. Tworzenie wolumenu na dane (opcjonalnie)
VOLUME /app/data

# 8. Ustawienie domyślnej komendy startowej
ENTRYPOINT ["java", "-jar", "/app/myapp.jar"]
```

Budowanie obrazu i uruchamianie kontenera

```
docker build -t myapp .

docker run -p 8080:8080 myapp
```

Sieci w dockerze

Bridge (domyślna)

Każdy kontener dostaje własny adres IP i może komunikować się z innymi kontenerami w tej samej sieci bridge.

```
docker network create my_bridge
docker run --network=my_bridge myapp
```

Host

Kontener korzysta bezpośrednio z sieci hosta, nie ma izolacji sieciowej.

```
docker run --network=host myapp
```

None

Kontener nie ma dostępu do żadnej sieci – pełna izolacja.

Overlay

Używane w Docker Swarm i Kubernetes, pozwala na komunikację między kontenerami na różnych hostach.

DOCKER - DEMO

branch: devops_1_docker_image

Pierwszy klient

- Zatrudniamy sprzedawcę

Pierwszy klient

- Zatrudniamy sprzedawcę
- Klient kupuje oczami

Pierwszy klient

- Zatrudniamy sprzedawcę
- Klient kupuje oczami
- Zmieniamy design

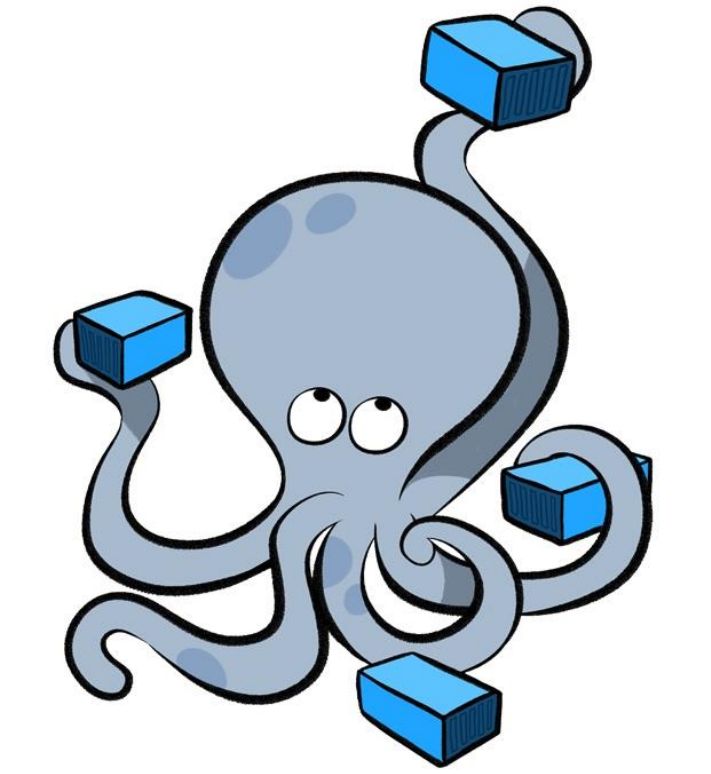
Pierwszy klient

- Zatrudniamy sprzedawcę
- Klient kupuje oczami
- Zmieniamy design
- Zatrudniamy programistów frontendowych

Pierwszy klient

- Zatrudniamy sprzedawcę
- Klient kupuje oczami
- Zmieniamy design
- Zatrudniamy programistów frontendowych
- Powstaje nowa aplikacja frontendowa

Docker Compose



Docker Compose

- Docker Compose to narzędzie do definiowania i zarządzania wieloma kontenerami

- Plik docker-compose.yml pozwala określić usługi, sieci i wolumeny

- Ułatwia zarządzanie zależnościami aplikacji
  (np. baza danych, backend, frontend)

```
services:
  frontend:
    image: nginx:latest
    container_name: frontend
    volumes:
      - ./frontend/nginx.conf:/etc/nginx/nginx.conf:ro
    ports:
      - "80:80"
    depends_on:
      - backend

  backend:
    build: ./backend
    container_name: backend
    environment:
      SPRING_DATASOURCE_URL:
jdbc:mysql://db:3306/app_db
      SPRING_DATASOURCE_USERNAME: user
      SPRING_DATASOURCE_PASSWORD: password
    ports:
      - "8080:8080"
    depends_on:
      - db
```

```
db:
    image: mysql:8
    container_name: db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: app_db
      MYSQL_USER: user
      MYSQL_PASSWORD: password
    volumes:
      - db_data:/var/lib/mysql
    ports:
      - "3306:3306"
```

Sieci w docker-compose

Domyślna sieć Bridge – kontenery w tej samej sieci mogą się komunikować po nazwach usług.

Możliwość definiowania własnych sieci dla lepszej izolacji.

```yaml
version: "3.8"
services:
  app:
    image: myapp
    networks:
      - mynetwork
  db:
    image: postgres
    networks:
      - mynetwork
networks:
  mynetwork:
    driver: bridge
```

DOCKER COMPOSE - DEMO

branch: devops_2_frontend_app

Frontend – port 3000
Backend – port 8080

?

Frontend – port 3000
Backend – port 8080

CORS (Cross-Origin Resource Sharing)

CORS (Cross-Origin Resource Sharing)

CORS (Cross-Origin Resource Sharing) to mechanizm bezpieczeństwa przeglądarki, który ogranicza, jakie zasoby mogą być ładowane z innych domen. Jest to zabezpieczenie, które zapobiega nieautoryzowanemu dostępowi do danych z innych stron internetowych.

- Jak działa CORS?

  Przeglądarka domyślnie blokuje żądania wysyłane z jednej domeny do innej (np. z frontend.com do api.backend.com).Serwer musi jawnie zezwolić na takie żądania, dodając odpowiednie nagłówki w odpowiedzi.

HAProxy

HAProxy (High Availability Proxy) to popularny, wydajny i lekki load balancer oraz proxy reverse'owy. Jest często stosowany w systemach o wysokiej dostępności i dużym obciążeniu, gdzie umożliwia równoważenie ruchu między wieloma serwerami backendowymi.

Kluczowe cechy HAProxy:

- Równoważenie obciążenia (Load Balancing).
- Reverse Proxy – ukrywa serwery backendowe przed klientami.
- Obsługa warstwy 4 i 7 – działa zarówno na poziomie TCP (Layer 4), jak i HTTP (Layer 7)
- Wsparcie dla SSL/TLS – umożliwia terminację SSL i przekierowywanie ruchu HTTP na HTTPS.
- Monitorowanie i statystyki.

HAPROXY - DEMO

branch: devops_3_haproxy

Kolejny klient – firma finansowa

- Duży klient
- Duży ruch
- Nowe wymaganie – kursy walut
- Dostępność 24/7

KURSY WALUT - DEMO

branch: devops_4_exchange_rates
devops_5_separate_service

Kubernetes

Podstawowe komponenty Kubernetes

Node

Fizyczny lub wirtualny serwer w klastrze Kubernetes, na którym uruchamiane są Pody

Deployment

Zarządza replikami aplikacji i aktualizacjami

Pod

Najmniejsza jednostka zarządzania, może zawierać jeden lub więcej kontenerów

Kontener

Lekka jednostka zawierająca aplikację i jej zależności, uruchamiana wewnątrz Poda

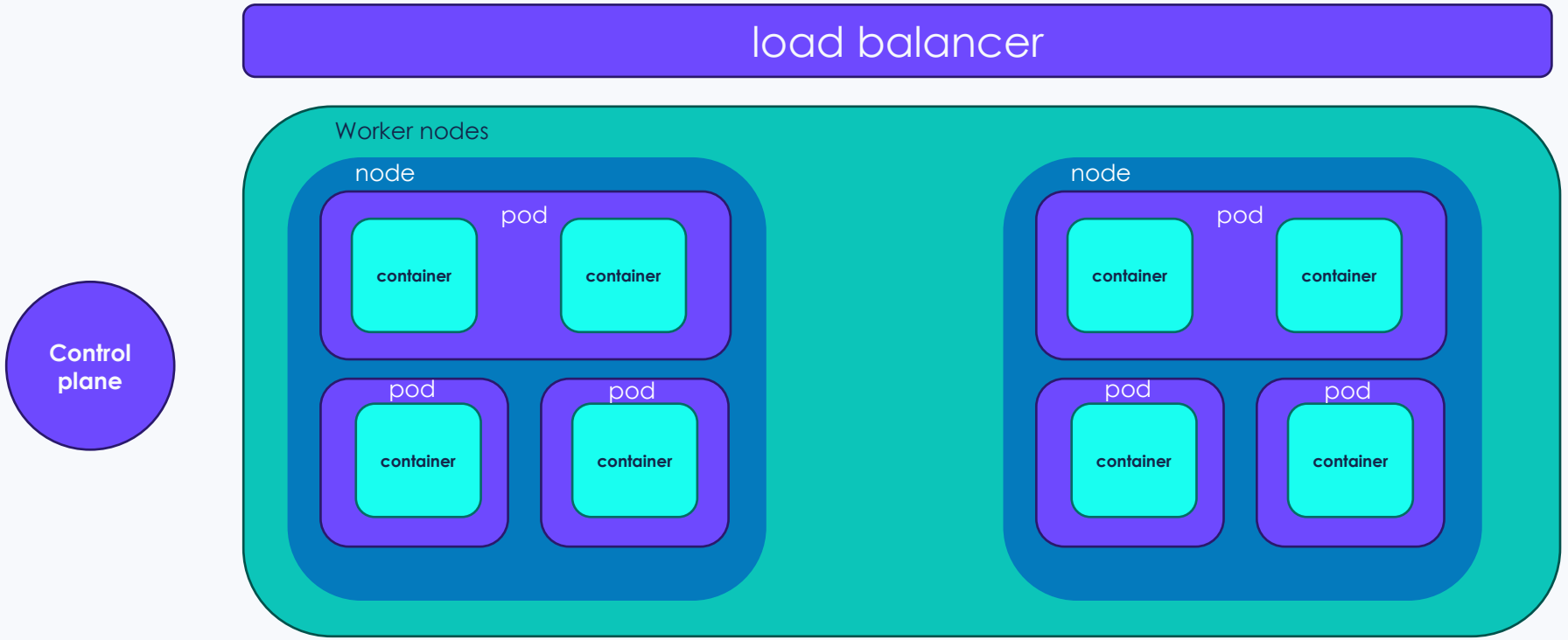
Service

Zapewnia dostęp do aplikacji, balansowanie ruchu

Ingress

Umożliwia dostęp do aplikacji z internetu (reverse proxy dla Kubernetes) – zastępuje Haproxy

Kubernetes - diagram



KUBERNETES - DEMO

branch: devops_6_kubernetes

Kiedy używać?

Docker

Gdy potrzebujemy izolowanego środowiska dla aplikacji.

W praktyce najczęściej na potrzeby developerskie

Docker Compose

Gdy aplikacja składa się z wielu współpracujących usług.

W praktyce na potrzeby deweloperskie gdy aplikacja składa się z wielu współdzielonych usług

Kubernetes

Gdy wymagamy automatycznego skalowania i zarządzania dużą liczbą kontenerów.

W praktyce - produkcja

Pytania