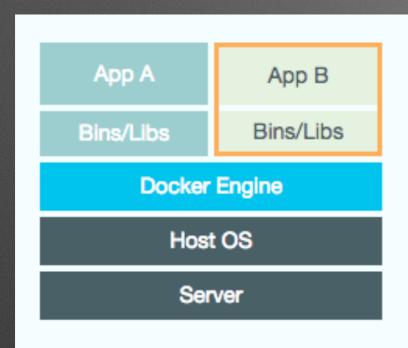
# Docker

# Docker Koncepcja

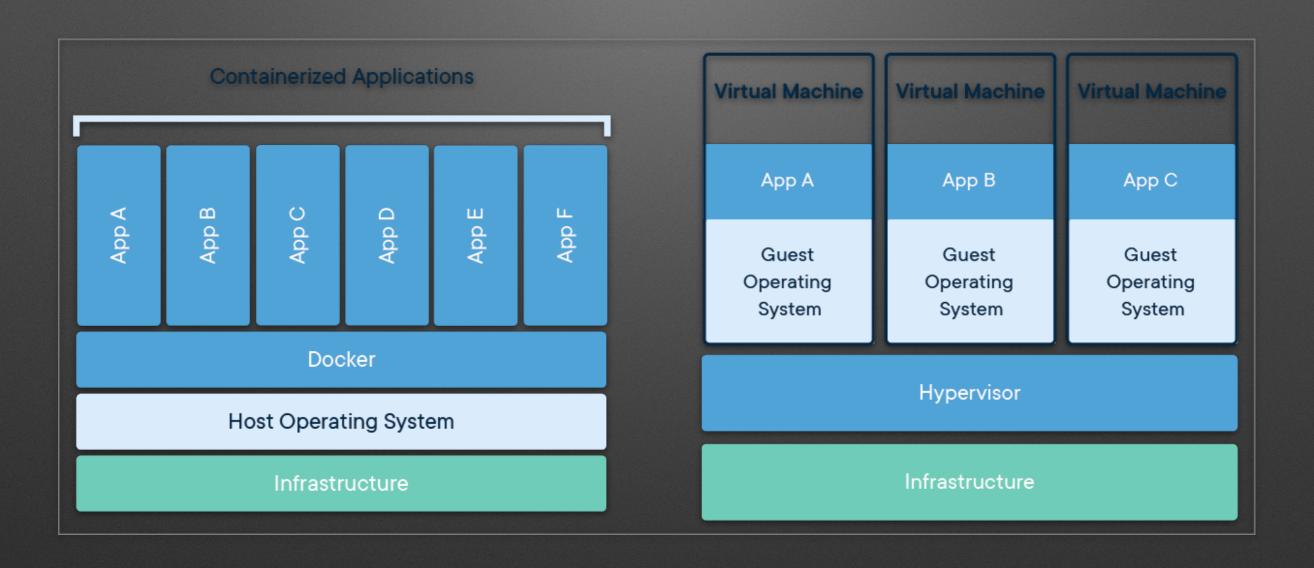


#### Docker

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

- Oparty o Linux
- Izolacja
- łatwość używania

### Linux Kernel Sharing



### Kontener

- Podstawowy byt w Dockerze
- Odizolowane środowisko w Dockerze (konfiguracja, podsieć, system) na którym uruchamiamy nasz aplikacjecje
- Service per Container
- definiwany przez w .Docker file
- Co można uruchomić w kontenerze? Wszystko
- Nazwa: [username]/[imagename]:[tags]

## Docker File

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim
# Set the working directory to /app
WORKDIR /app
# Copy the current directory contents into the container at /app
COPY . /ap
# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt
# Make port 80 available to the world outside this container
EXPOSE 80
# Define environment variable
ENV NAME World
# Run app.py when the container launches
CMD ["python", "app.py"]
```

## Docker Image

- Plik zapisany w machine's local Docker image registry
- Docker Image uruchamiamy na kontenerze
- Każdy image dziedziczy po Base Image
- Można zrobić image z działającego kontenera
- Posiada image ID
- Można zrobić Docker Image budując z Docker File:

docker build --tag=friendlyhello .

#### **Docker Compose**

- Narzędzie do definiowania zestawu kontenerów, konfiguracji, sieci itd. w plikach .yml
- UWAGA!!! docker stack który też używa plików .yml jednak nie wszystko jest tak samo traktowane
- Łatwe szybkie startowanie stopowanie docker-compose up, docker-compose down

```
app:
  build: ./
  volumes:
    - ./:/var/www/app
  ports:
    - 3000:3000
  environment:
    - NODE_ENV=development
    - PORT=3000
  command:
    sh -c 'npm i && node server.js'
mongoDb:
    image: "mongo:mongo"
```

docker buildtag=friendlyhello .	Buduje obraz z pliku DockerFile – musi byc w akutalnym
docker image ls	Listuje obrazy
docker run -p 4000:80 friendlyhello	Uruchamia konetner z obrazu z określonym linkowaniem
docker container <mark>ls</mark>	Listuje kontenery
docker container stop 1fa4ab2cf395	Stopuje kontener
docker tag friendlyhello gordon/get-started:part2	Taguje obraz
docker run -p 4000:80 username/repository:tag	Uruchamia konetner z obrazu ze zdalnego repo
docker stack deploy -c <composefile> <appname></appname></composefile>	Deploy stacka z docker-compose.yml
docker-compose up	Start kontenerów z docker-compose.yml
docker-compose down	Położenie kontenerów z docker-compose.yml
docker swarm join	Dołącza do klastra jako node
docker swarm init	Inicjalizuje cluster jako maszyna manager
docker stack deploy -c docker-compose.yml getstartedlab	Deploy aplikacji na cały klaster
docker stack ps getstartedlab	Listuje stacki
docker stack rm getstartedlab	Usuwa stack

### Docker Swarm

- swarm służy do zarządzania grupą maszyn tworząca klaster
- Zawsze jest maszyna ktora jest managerem clustra
- Do clustra dołczają się też maszyny slave' autoryzując
- się tokenem
- Jest to bardzo proste i wygodne

%inicjalizacja managera klastra docker swarm init- in %sparwdzenie tokena managera Docker swarm join-token manager

#### Dolaczenie do klastra przez noda

```
docker swarm join \
--token SWMTKN-1-5bg7or88h8t8hclr0zdy228rydj3eomo29n6xldto3fdrgutka-bqzcley2zg2l67r8hssy3ci2o \
192.168.65.2:2377
```

%deployment aplikacji na klastrze docker stack deploy -c docker-compose.yml getstartedlab