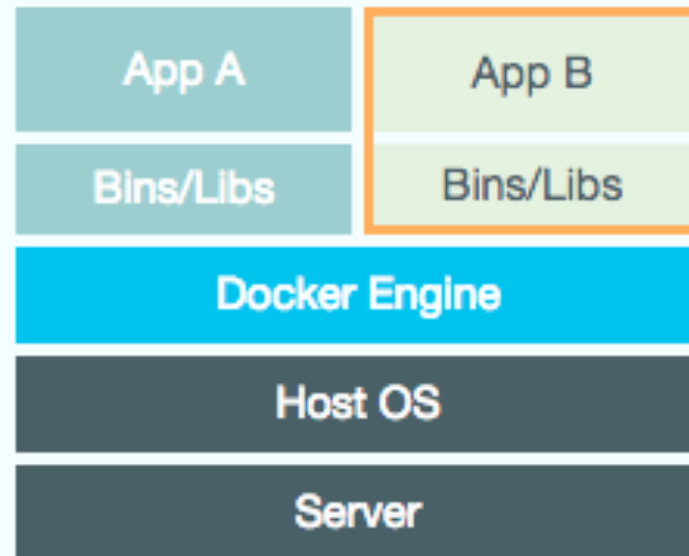


# Docker



# Docker Koncepcja



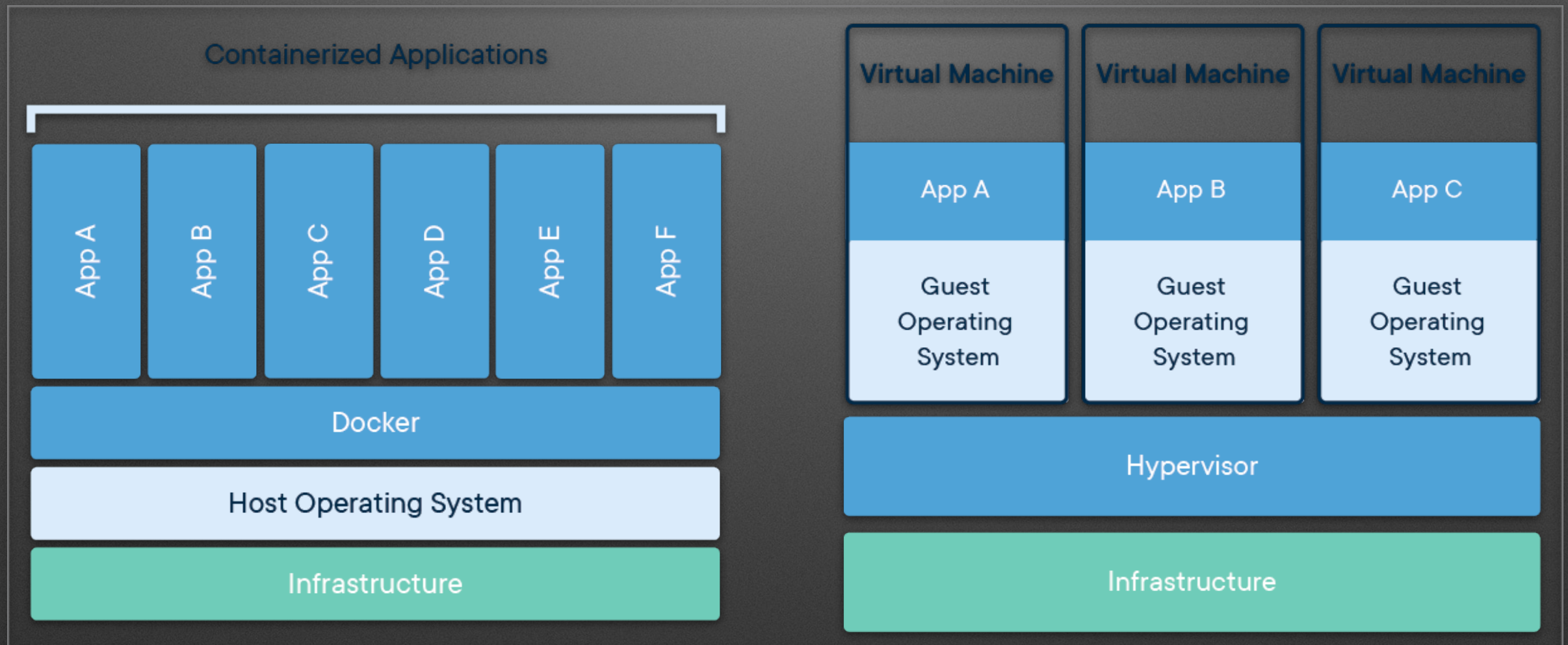
## Docker

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 naszą aplikację
- 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 . /app

# 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 .yaml
- UWAGA!!! docker stack który też używa plików .yaml 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”
```



[illegible]



# Docker Swarm

- swarm służy do zarządzania grupą maszyn tworzącą klaster
- Zawsze jest maszyna która jest managerem klastra
- Do klastra dołączają 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
```