

Odisee
DE CO-HOGESCHOOL

Big Data – Containers



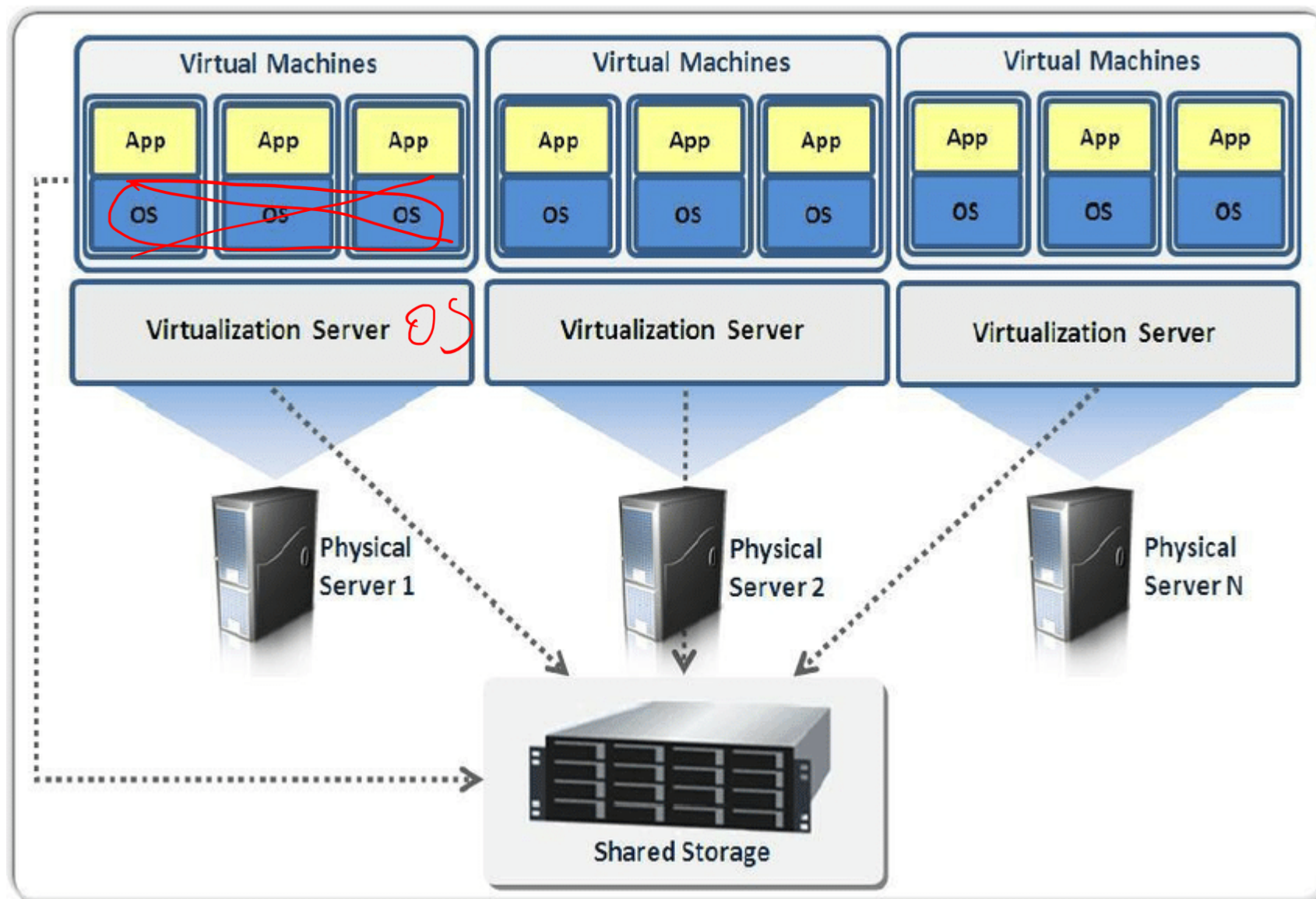
Jens Baetens



Virtualisatie

- ▣ Het virtueel maken van een fysiek toestel
 - Kan zowel op niveau van server, netwerk, opslagmedium, ...
 - In deze cursus vooral het niveau van server relevant
 - Voordelen:
 - Efficiëntie
 - Flexibiliteit
 - Disaster recovery

Server virtualisatie





Containers

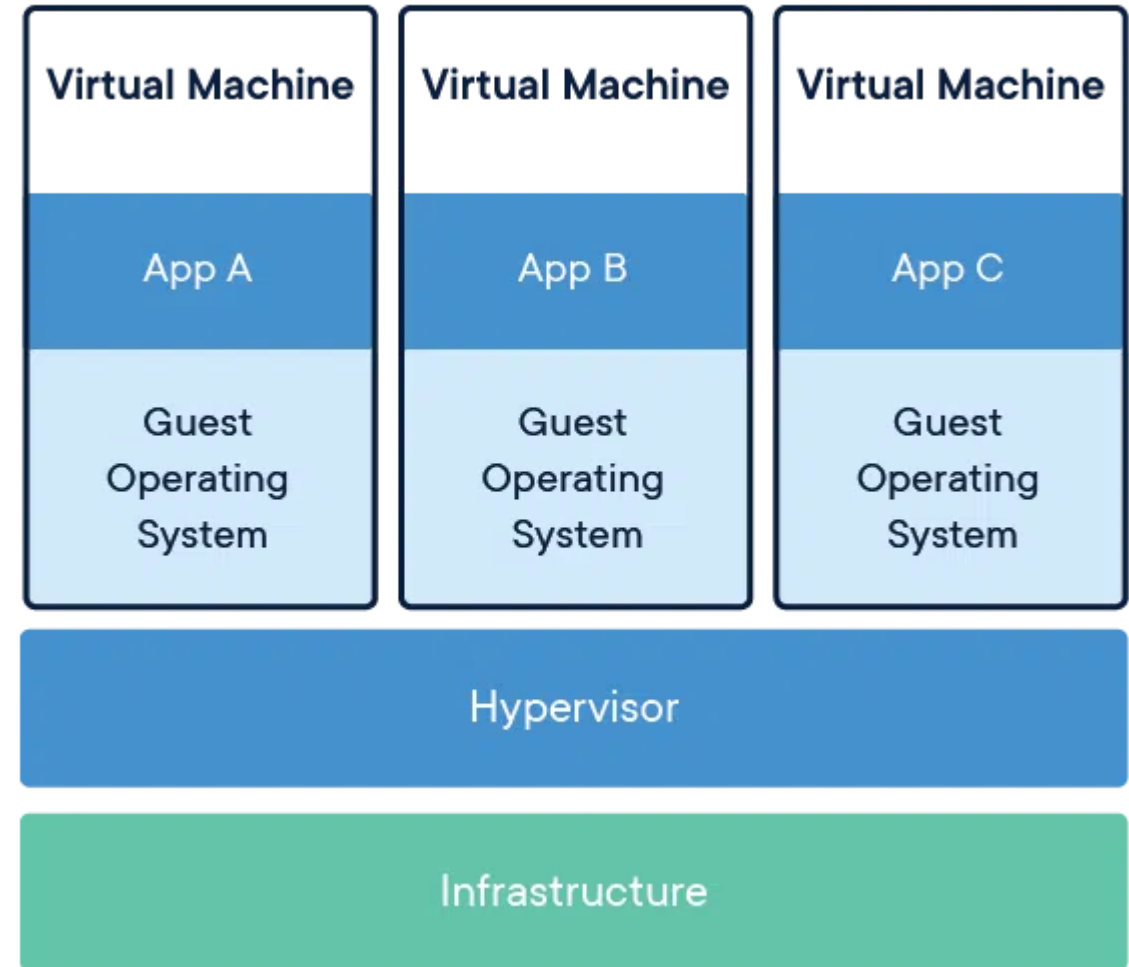
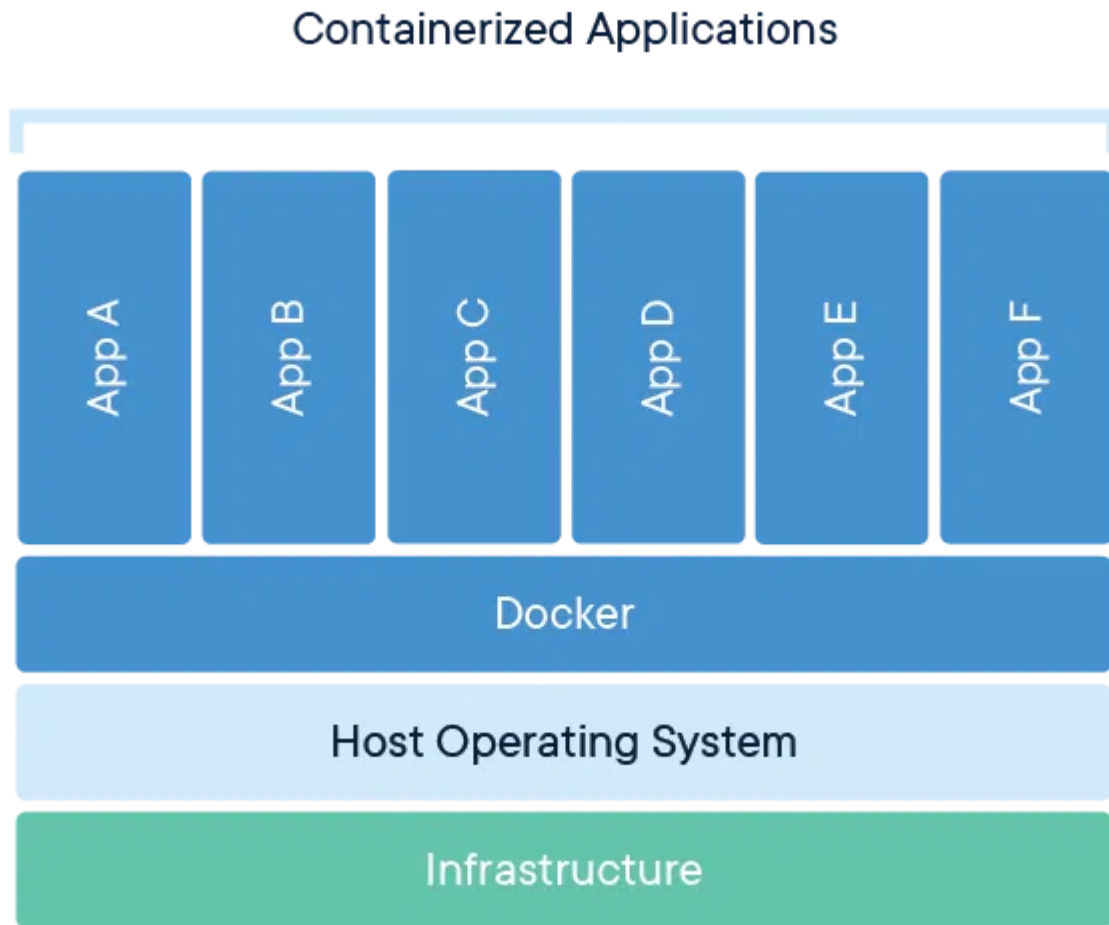
- ▣ Vorm van virtualisatie om applicaties en hun afhankelijkheden te isoleren
- ▣ Lichte, efficiënte en consistente omgeving voor het ontwikkelen, testen en implementeren van applicaties
- ▣ In tegenstelling tot virtuele machines delen containers de OS-kernel
 - ▣ Lichter en sneller te starten



Containers

- ▣ Abstractie van de application layer
 - Groepeert applicatie code en afhankelijkheden
 - Alle containers delen de OS-kernel maar runnen onafhankelijk
- ▣ Containers worden beheerd door container runtimes ipv hypervisor
- ▣ Soorten software om containers te beheren:
 - Docker
 - Kubernetes
- ▣ Standaard voor containers ontwikkeld door Docker

Containers vs Virtuele machines





Voordelen van containers

- ▣ Zelfde voordelen als VM's maar
 - Minder groot (MB's per image ipv 10 GB of meer)
 - Starten sneller (omdat ze minder groot zijn)
 - Schaalbaarder



Kubernetes

Kubernetes

termen niet te kennen

■ Orchestration tool

- ▬ Automatiseren van de deployment, scaling en management van containerized applicaties

■ Basisconcepten:

- ▬ Pods: De kleinste eenheid in Kubernetes, bestaande uit een of meer containers.
- ▬ Services: Definiëren hoe pods met elkaar en met andere services communiceren.
- ▬ Deployments: Beheren de declaratieve updates voor pods en replica sets.

■ Configuratie via een yaml-file

- ▬ Gelijkaardig aan de compose-file bij docker



Use cases



Toepassingen van containers

- ▣ Web development
 - ontwikkelomgevingen te standaardiseren.
- ▣ Big Data
 - schalen van data-analyse workloads.
 - Data analyse pipelines
 - Gedistribueerd trainen van ml-modellen
- ▣ Microservices
 - implementeren en beheren van microservices.



Docker terminologie

Basic taxonomy in Docker



A **Registry**
Stores many
static images



Images
Static, persisted container image

*→ dit is nog niet
gestart.*



Container = image die runt / gestart is
Image-instance running
an app process (service/web)

Hosted Docker
Registry

Docker Trusted
Registry on-prem.

On-premises
(‘n’ private organizations)

Docker Hub
Registry

Docker Trusted
Registry on-cloud

Azure Container
Registry

AWS Container
Registry

Google
Container
Registry

Quay
Registry

Other Cloud

Public Cloud
(specific vendors)

*→ hier staan
mijn containers*

Basic taxonomy

▣ Registry

- ▬ Repository voor bewaren en verdelen van container images
- ▬ Bvb: Docker hub

▣ Container image

- ▬ Een lichtgewicht, standalone en uitvoerbare software package
- ▬ Bevat applicatiecode en afhankelijkheden maar geen OS

▣ Container

- ▬ Een container image dat uitgevoerd wordt

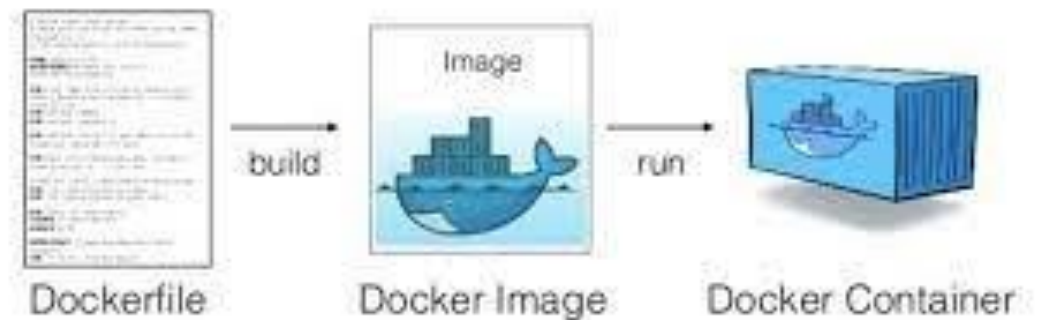
▣ Volume

- ▬ Een manier om data te bewaren buiten de container of data te delen tussen host en/of containers

Docker file

■ Een set van instructies om een container image te bouwen

- Voorbeeld: dockerfiles voor onze containers
- Belangrijke commando's:
 - FROM: van welke image te starten (uit een repository)
 - RUN: execute command in shell
 - ADD/COPY: Copy file from build directory to image
 - EXPOSE: open netwerkpoort
 - CMD/ENTRYPOINT: wat er gestart moet worden
 - ENV: set environment variabele
 - ARG: set build-time variabelen





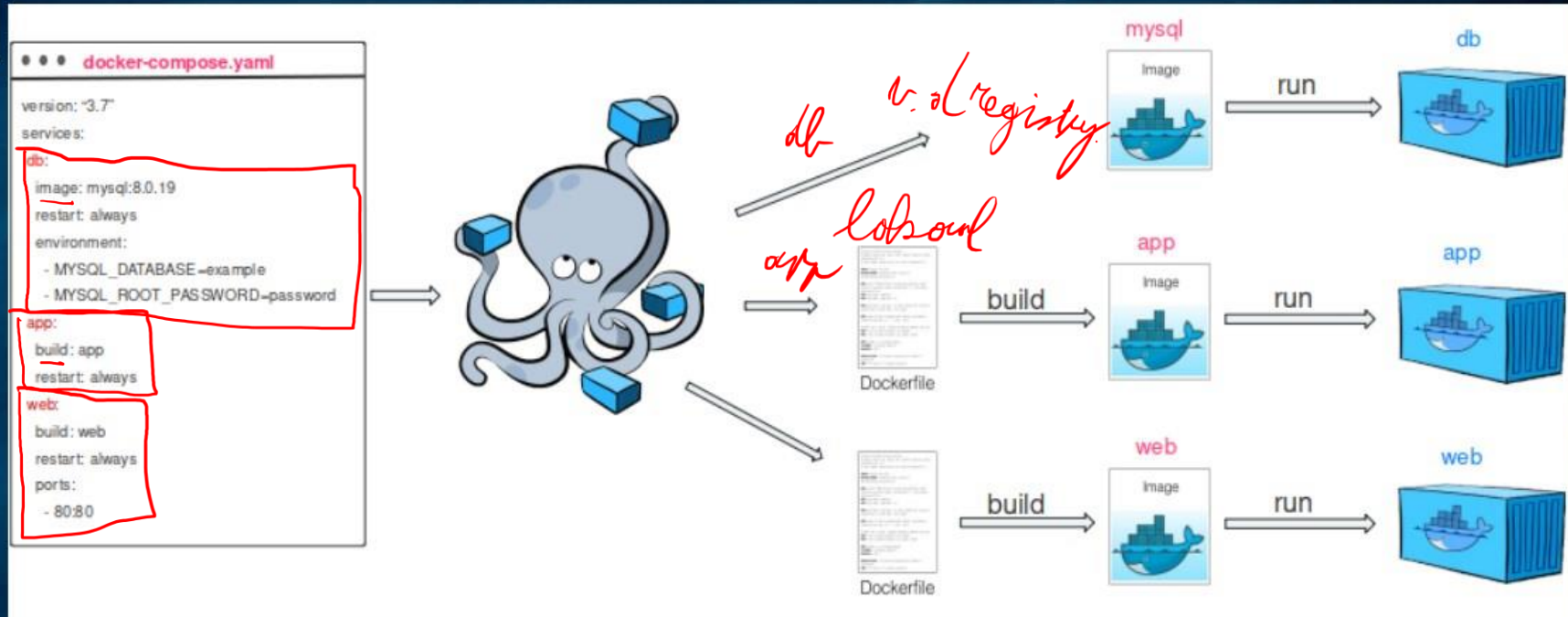
Docker compose

- ▣ Een software applicatie kan uit meerdere containers bestaan
 - Website bestaat uit een server en een database (2 containers bvb)
 - Onze cluster bestaat uit onder andere een namenode en 4 datanodes
- ▣ Is een tool om multi-container applications te definiëren en te starten
 - Maakt gebruik van een yaml file
- ▣ Ter voorbeeld: zie compose hadoop

Docker compose



Docker Compose



Run multiple containers as a service from 1 .yaml file



Demo

Een eenvoudige container

- ▣ Schrijf een Dockerfile voor de applicatie.

```
FROM python:3.8-slim
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

- ▣ Bouw de Docker image

- Docker build . --tag demo1

(punt is belangrijk voor deze demo)

- ▣ Start de container

- Docker run demo1

Demo website development met docker-compose

▣ Docker-compose file

- ▬ Backend
 - Postgres database
 - Flask api

▣ Bouw de twee docker-containers

- ▬ Docker-compose build

▣ Start de containers

- ▬ Docker-compose up

▣ Surf naar localhost:5000

```
db:
  image: postgres:13
  environment:
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: password
    POSTGRES_DB: mydatabase
  volumes:
    - postgres_data:/var/lib/postgresql/data
```

```
web:
  build: .
  ports:
    - "5000:5000"
  depends_on:
    - db
  environment:
    - DATABASE_URL=postgresql://postgres:password@db:5432/mydatabase
```

Hello, World!

Oefening - Dockerfiles

- Maak de volgende containers:
 - ▬ Container met een Hello World script
 - ▬ Container waarbij het requests package geïnstalleerd worden
 - Print de response uit van een request naar <https://api.github.com>
 - ▬ Container waarbij een data.txt met wat testdata (random mag je zelf invullen) toegevoegd wordt aan de containers
 - Print in de container de inhoud van het bestand uit



Oefening – Docker compose

- Maak een docker-compose file dat de vorige containers allemaal toevoegd en uitvoert.