Containers (with emphasis on Docker)

Jakub Daněk
Yoso Czech s.r.o.

mailto: jakub.danek@yoso.fi

What are containers?

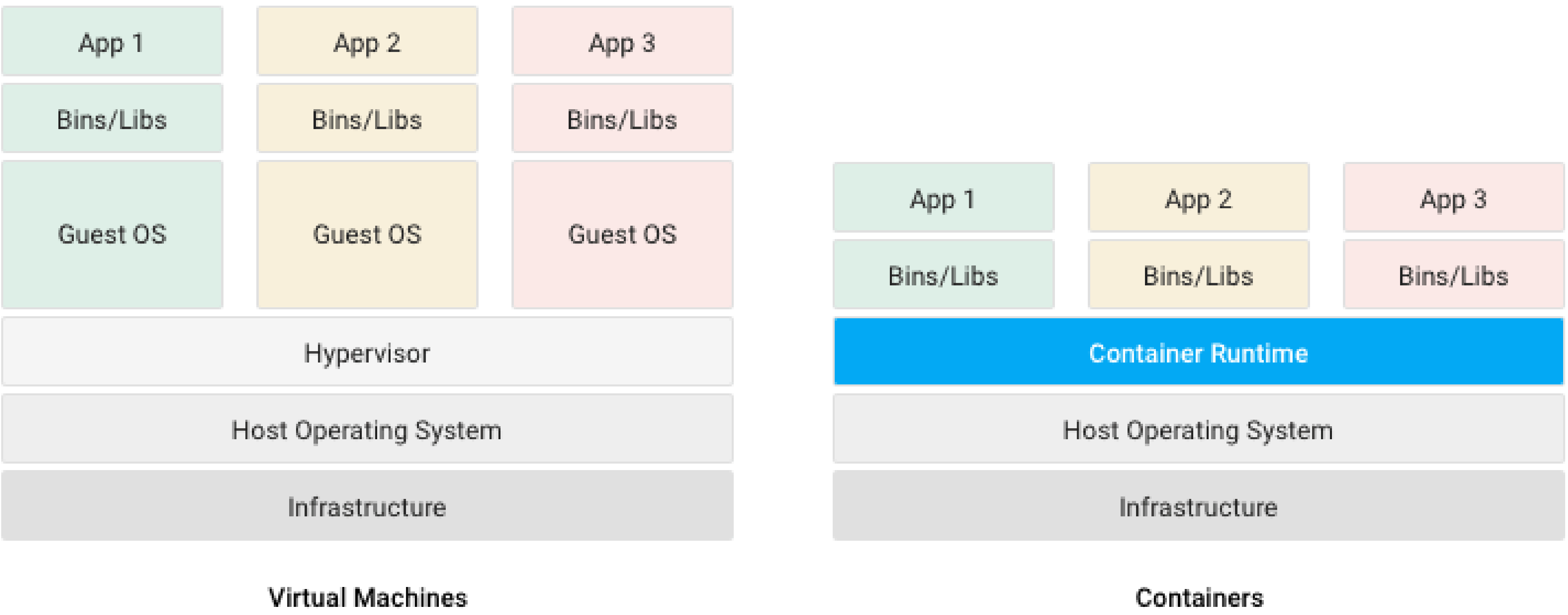
- **Abstraction of applications from the actual runtime environment**

- **Consistent deployment independent on system**
  - Cloud
  - Laptop
  - Own servers

What are containers?

- **Benefits of Consistency**
  - Run anywhere
    - Relatively easy to run on many different platforms
  - Same environment in dev, test and production
    - → easier transmission from one to another
    - → less bugs introduced by environment differences

Container vs Virtual Machine



Src: https://cloud.google.com/containers/

Container vs Virtual Machine

Virtual Machines

- Hardware virtualization → one physical server becomes many virtual servers
- Each VM contains full operating system
- Each VM has own (possibly different kernel)
- Each VM has to boot during start (init scripts, systemd...)

Src: https://cloud.google.com/containers/

Container vs Virtual Machine

Containers

- Process-level virtualization → each container runs as process group in host OS
- All containers on single host use the same kernel
- Process startup is much faster than VM boot (generally)
- Enabled by multiple kernel features for process constraint and isolation

Src: https://cloud.google.com/containers/

Cgroups

Control groups

- **Linux kernel feature allowing setting resource limits for a process group**

- **Tree-structure**

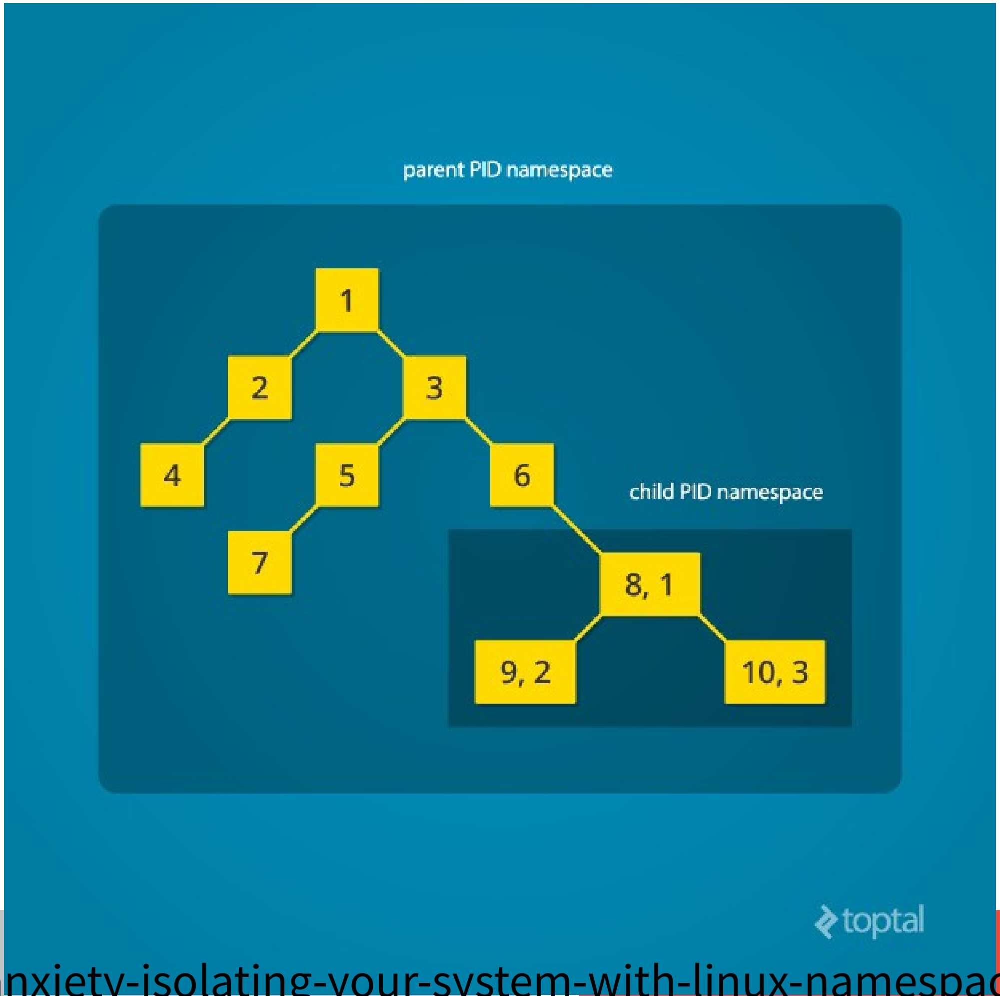
- **Each process belongs to exactly one cgroup**
  - All threads spawned by a process belong to the same cgroup as the parent process

Namespaces

- **Means of isolation of processes running on the same host.**
- **Tree structure**
- **Processes in one namespace do not see resources of another namespace**
  - But processes in parent namespace can see resources in child namespaces

Namespaces - PID

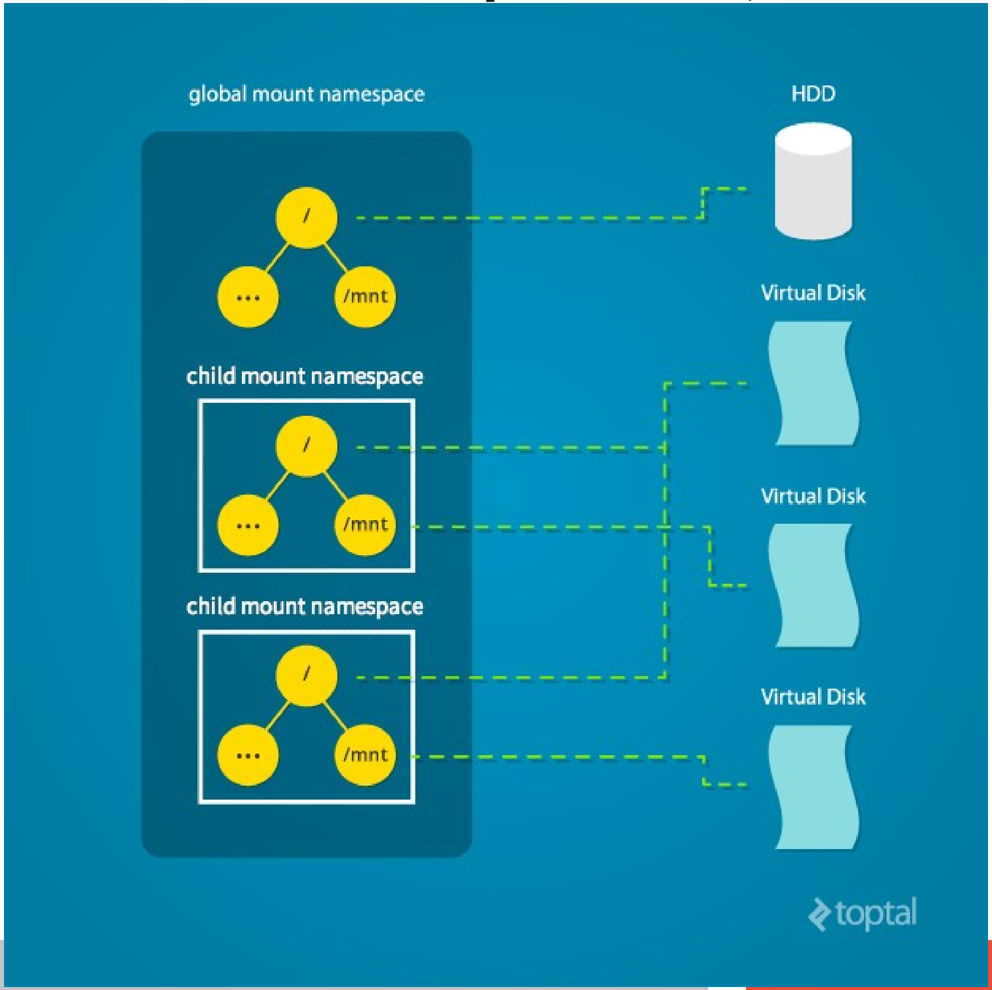
- Process has different ID for each PID namespace to which it belongs



Src: https://www.toptal.com/linux/separation-anxiety-isolating-your-system-with-linux-namespaces

Namespaces - Mount

- When isolated process mounts new disk/partition, it affects only its namespace



Src: https://www.toptal.com/linux/separation-anxiety-isolating-your-system-with-linux-namespaces

Namespaces - types

- cgroups
- user IDs (UID)
- network
- mount points
- interprocess communication (IPC)
- process IDs (PID)
- hostname (UTS)

And some more

- **Seccomp**
  - Preventing processes from making particular system calls
- **Capabilities**
  - Permissions to certain actions generally allowed only to privileged users (root)
  - Can be enabled per-thread for unprivileged processes
- **Apparmor and SELinux**
  - Some more program restricting :)

More reading

Linux Man pages:

http://man7.org/linux/man-pages/man7/capabilities.7.html

https://www.kernel.org/doc/Documentation/cgroup-v2.txt

http://man7.org/linux/man-pages/man7/namespaces.7.html

Under the hood of Docker:

https://pasztor.at/blog/under-the-hood-of-docker

Namespaces:

https://www.toptal.com/linux/separation-anxiety-isolating-your-system-with-linux-namespaces

Rami Rosen (the page is not pretty, but contents are awesome):

http://ramirose.wixsite.com/ramirosen

LXC

- **API for creating linux containers**
- **In general, the containers run full OS**
  - Including init, cron, syslog…
- **→ similar result as with VMs, but implemented as containers**
  - Keeps flexibility of VMs
  - Keeps speed and lower overhead of containers

LXD

- **Container manager built with LXC**
- **Provides common images in remote repository**
- **Scalability, security, resource management**
- **Easy to get into (comparing to low-level LXC API)**
  - https://linuxcontainers.org/lxd/try-it/

Docker

- **Another container engine**
- **Younger than LXC**
- **Different approach: single-application deployment**
    - → docker containers do not run full OS
    - → provide isolated environment for single-application runtime

Docker

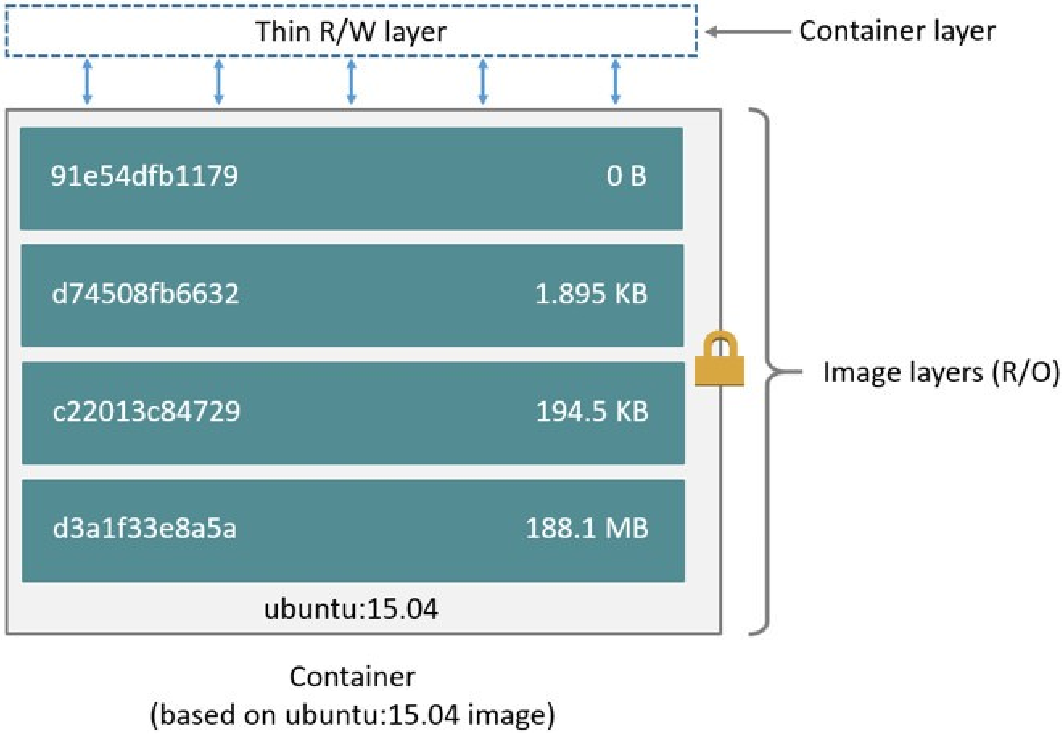
- **First Class Citizens:**
  - Image - binary form of the runtime environment
    - identified by id, name and tag (version)
  - Container - running environment based on an image
    - identified by unique ID and name
- **Images are stored in a repository (private, public, local)**
  - similar mechanism to Maven - if you do not have an image locally, docker will download it for you from remote repositories

Building Docker Images

- Images defined by text specification (Dockerfile)

- Images are built of a series of filesystem layers

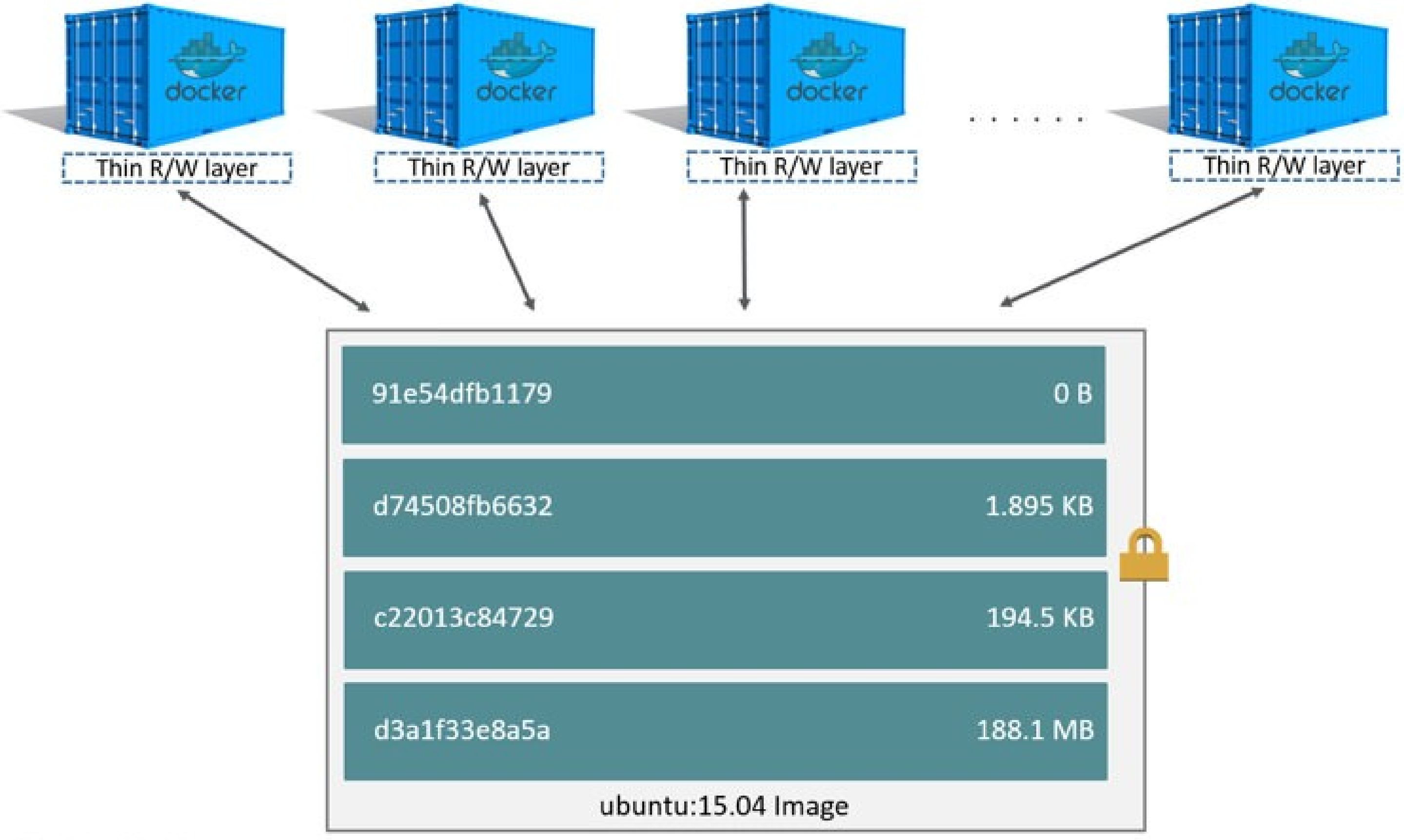
```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



Src: https://docs.docker.com/storage/storagedriver/#images-and-layers

Building Docker Images

- Containers have own R/W layer → can share same image



Src: https://docs.docker.com/storage/storagedriver/#images-and-layers

Docker Containers

- **State of container (changes to the R/W layer) can be commited to create new image**
- **Until then they need to be considered transient**
  - → not good for storing application data
- **Filesystem layers introduce increased complexity to the filesystem implementation**
- **When an update is needed, new image is built and new container started (and old one is killed)**
  - → different philosophy to LXC or VMs where updates are done in runtime

Docker – Storage Options

- **Bind Mounts**
  - older way of storing persistent data
  - mount a folder on host system to a folder inside container
  - -> you need to know which folder you want to mount
    - it must exist before starting the container
- **Example: C:\\Users\danekja\mysql -> /var/lib/mysql**
  - When container runs mysql and it stores data, they can be found on host filesystem

Docker – Storage Options

- **Volumes**

  - Volumes are created on host filesystem and mounted to a path inside the container

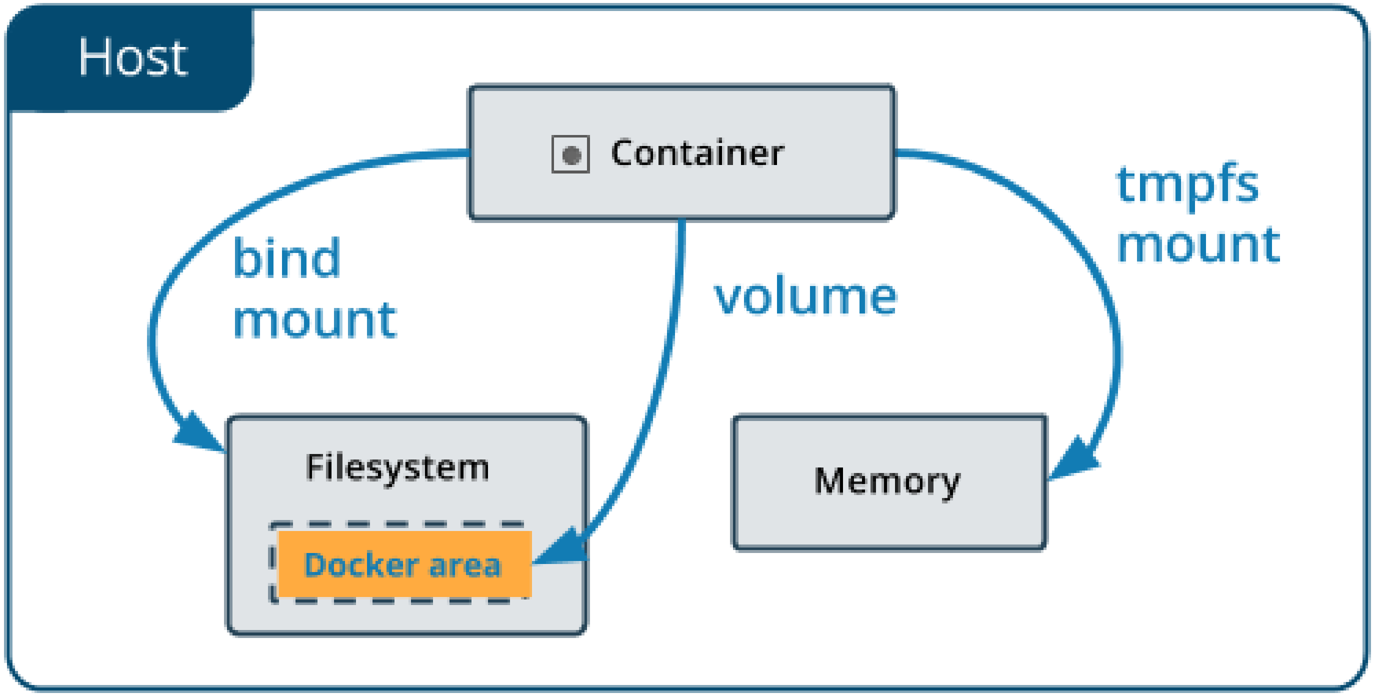
  - Difference to bind mounts: volumes are created by docker in space managed by docker (usually in /var/lib/docker on Linux hosts)
    - on Windows hosts this is a bit complicated because Docker actually runs in Linux VM

Docker – Storage Options

- **Volumes**

  - Volumes can be created independently of containers
    - as most Docker objects are identified by hash ID and a name
  - -> Volumes can be shared among multiple containers
  - Volumes can be remote

Docker – Storage Options



Src: https://docs.docker.com/storage/volumes/

Docker - networking

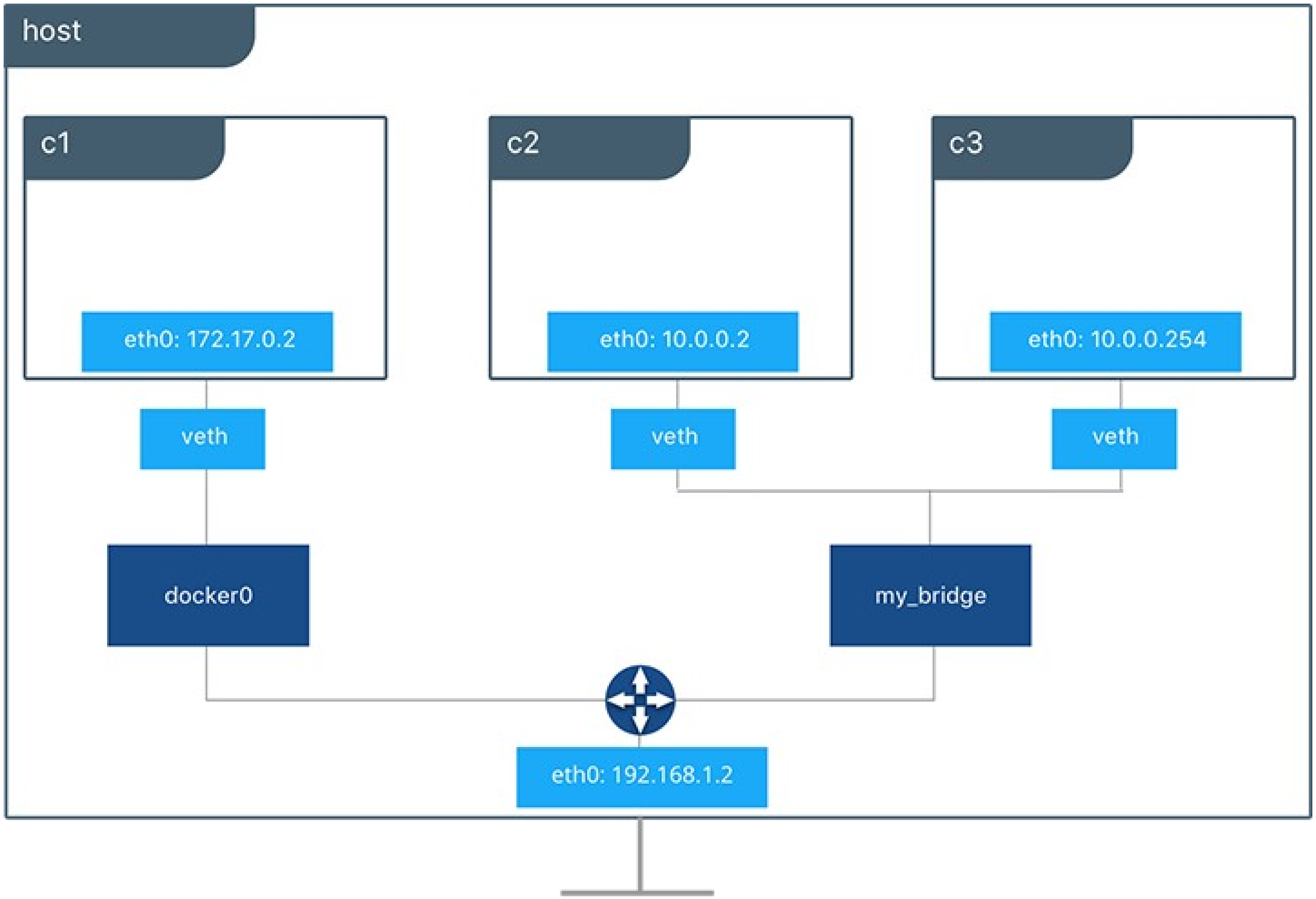
- **Docker provides multiple networking options**
  - Full overview: https://docs.docker.com/network/
  - All manipulate host‘s network configuration (by adding network devices, manipulating iptables) to:
    - create internal networks
    - connect containers to the internet
    - make containers to look like a physical device to the host OS
- **We will discuss only Bridge networks**

Docker - networking

- **Bridge network**
  - Created in terms of one local docker daemon (not for clusters)
  - Containers within same bridge network can communicate with each other
  - Containers in different networks cannot communication with each other
  - Containers can call each other either with IPs or their names

  - Default bridge network is created when Docker starts
    - All new containers are assigned to it unless specified otherwise
    - In the default bridge network containers may communicate only via IPs

Docker - networking



src: https://success.docker.com/article/networking

Docker - networking

- By default, all containers within a network expose all their ports to each other, but no ports to the outside world

- -> unless you specifically allow it, containers cannot be reached from the outside world

- Ports are exposed by binding HOST ports to container port

Docker - networking

- **Ports are exposed by binding HOST ports to container port**

- **Example:**
  - Server hostname: myserver.com
  - Container running tomcat on port 8080
  - Port mapping 80 -> 8080

  - -> calls to http://mysever.com/ will reach the Tomcat inside the container
  - -> calls to http://myserver.com:8080/ will respond with „Server not available“

Some more reading

https://www.docker.com/resources/what-container

https://linuxcontainers.org/lxc

https://linuxcontainers.org/lxd/

https://pasztor.at/blog/lxc-vs-docker

https://goto.docker.com/rs/929-FJL-178/images/IDC-containerplatform-wp.pdf

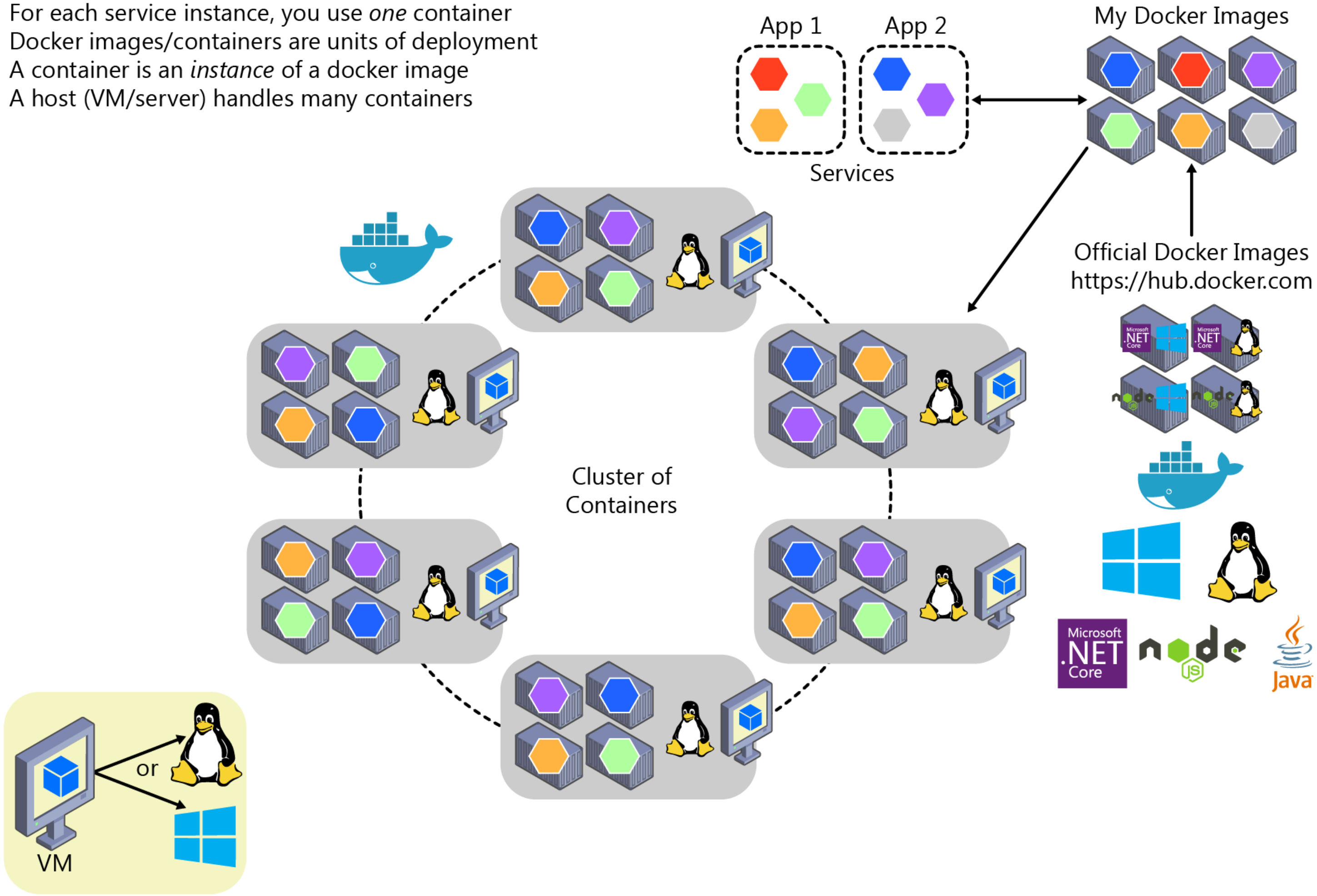
https://success.docker.com/article/networking

Dockerizing Application

- **Application needs to be configurable via environment variables**
  - in theory it is possible to load configuration files into container via bind mounts, but that becomes somewhat complicated when running in cluster
- **That‘s it, you are good to go :)**
- **General rule is to have one process per docker container**
  - can be resolved by custom shell script or projects like http://supervisord.org/
  - but generally recommended approach is one process per container

Dockerizing Application

- For each service instance, you use *one* container
- Docker images/containers are units of deployment
- A container is an *instance* of a docker image
- A host (VM/server) handles many containers



src: https://docs.microsoft.com/en-us/dotnet/standard/containerized-lifecycle-architecture/design-develop-containerized-apps/orchestrate-high-scalability-availability

Container Orchestration

- **main strength of Docker ecosystem comes with easy creation of multi-service deployment and high-availability setup**

- **Many projects approach this**

    - Docker Compose

    - Kubernetes, Docker Swarm

    - AWS, Google Cloud, Azure…

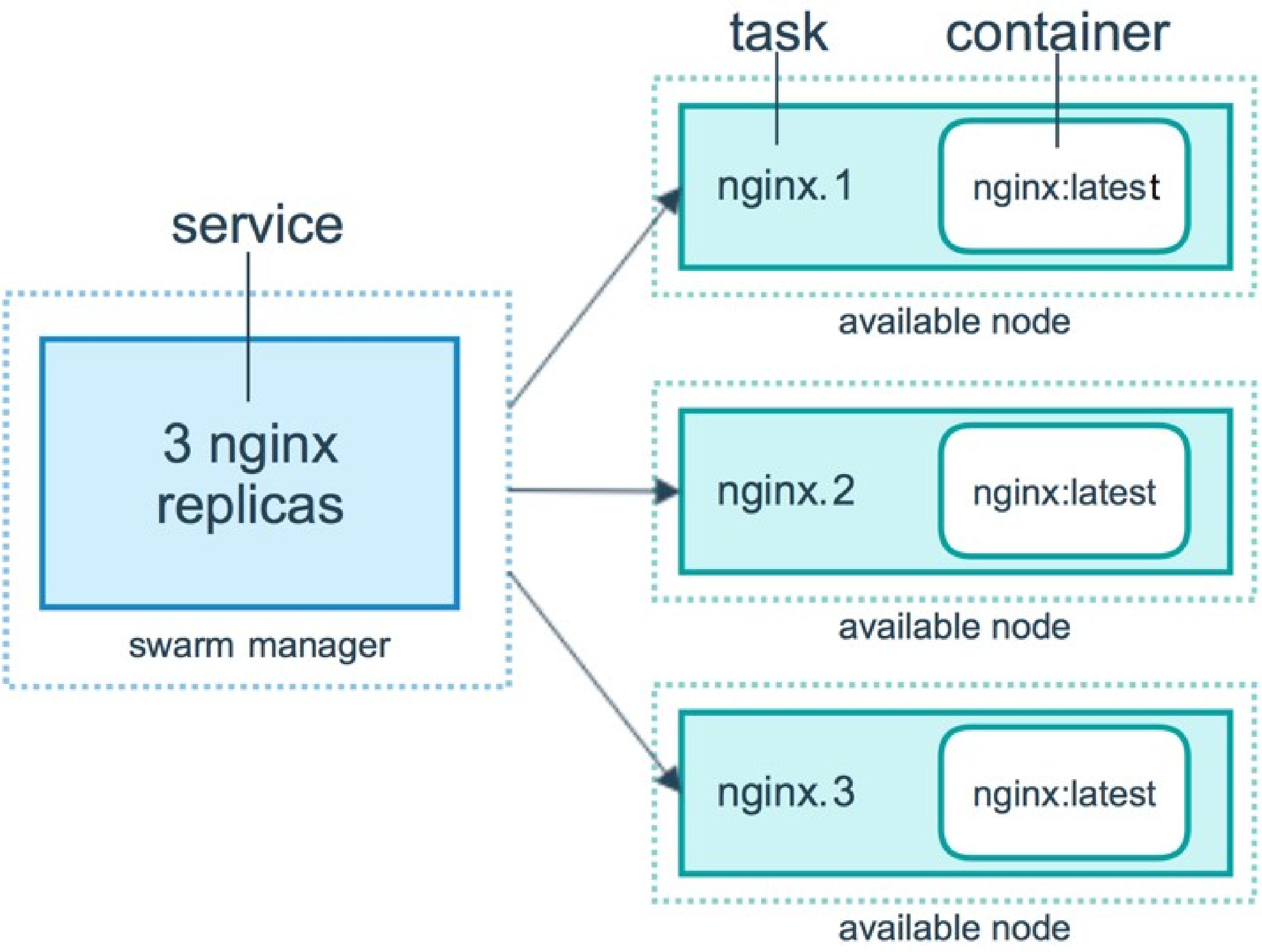
Docker Compose

- **Tool for orchestrating containers based on single configuration file**
- **Easy to setup, easy to learn**
  - Can be used to configure services, networking, volumes, environment variables...
  - Good for development
  - Can be used for production on a small scale (single host)
    - No load-balancers
    - Needs OS tools to support restart in case of crash/server reboot
  - On larger scale, must be used together with Docker Swarm

Docker Swarm

- **Cluster mode from Docker**
  - Connects multiple Docker nodes into a cluster and allows their manipulation via single-point-of-entry API
    - Developers use same commands (or docker-compose configuration) to deploy containers to the cluster
    - Swarm handles deployment of individual container replicas to nodes

Docker Swarm - Services



https://docs.docker.com/engine/swarm/how-swarm-mode-works/services

Docker Swarm - Services

- **Swarms runs internal DNS server**
- **Each service is given a record in DNS via which it can be reached**
- **Swarm handles load-balancing between the running replicas**
- **Configurable health-checks on service level**
- **This is a common approach provided by most of the orchestration tools**
  - -> load-balancing out-of-the-box

Kubernetes

- **More mature, based on Google‘s experience with containers**
  - also more widely used -> easier to find information
- **Provides extended features comparing to Docker Swarm**
  - auto-scaling, health-checks on container-level
  - is not restricted to Docker API (unlike Swarm which has to have the same API as Docker engine)

Kubernetes vs Docker Swarm

- **Kubernetes cons**
  - Do-it-yourself installation is rather complex
    - But there is a simple minikube tool for running on workstations and testing configurations
  - Need to learn another tool
- **Docker Swarm pros**
  - Simpler, same API as Docker
- **Still, Kubernetes is widely used and supported by main cloud providers**

Openshift, GappEngine, AWS...

Where can you host your Docker clusters?

- **Own VMs/servers**
  - -> awfull lot of work if you are serious about your data and availability

- **Openshift (IBM - RedHat)**
  - Cloud platform built on top of Kubernetes
  - Significant changes/improvements - support for builds, Jenkins integration, image registry and management
  - Very good visualisation
  - But: vendor lock-in

Openshift, GappEngine, AWS...

Where can you host your Docker clusters?

- **Google Cloud, AWS, Azure**
  - all provide options to run either plain Docker images with custom deployment configurations for the particular cloud
  - or they provide Kubernetes cluster hosting
- **Main con of all (including Openshift): price is much higher than with regular VMs**

Final Thoughts

- **Note 1: The whole ecosystem evolves quickly, always check for the newest information**
  - Official documentation is your friend
- **Note 2: There is no such thing as silver bullet**
  - Not every application is suitable to run in container
  - But most of the time it is great at least for prototyping integration scenarios
- **Note 3: With the momentum given, containers seem to be the future, pay attention to them**
  - (this is author‘s personal opinion)

Some more reading

https://kubernetes.io/

https://cloud.google.com/kubernetes-engine/

https://aws.amazon.com/eks/

https://azure.microsoft.com/en-us/services/kubernetes-service/

Questions?