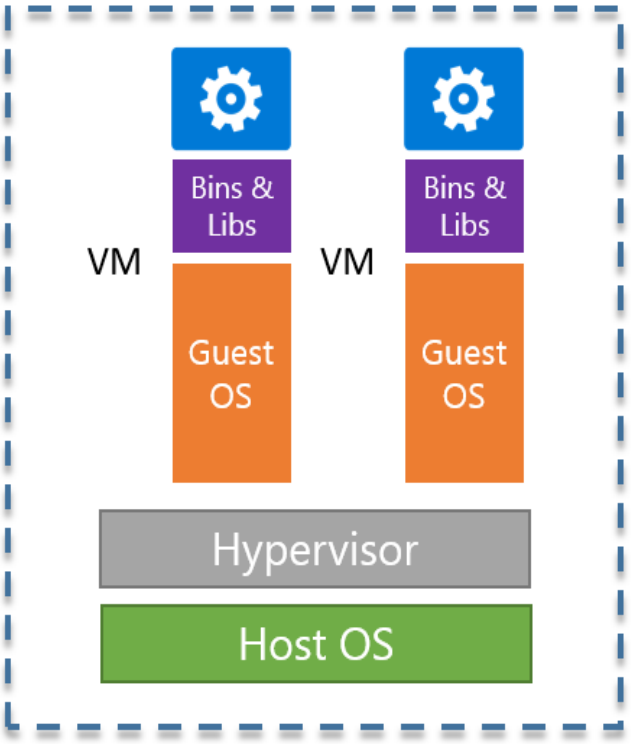
Introduction to Docker & Containerization

What is Docker?

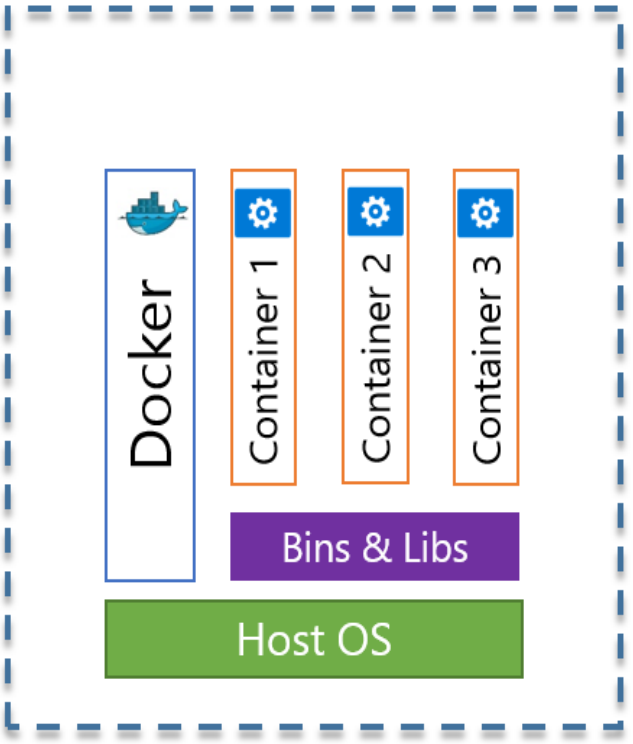
- It's a Linux software (now not only on linux)
- It's not a programming language, nor a framework for building software
- It solves problems like installing, removing, upgrading, distributing, trusting and managing software using a containerization technology
- It simplifies life to sysadmins and developers
- It accomplishes this using a UNIX technology called containers
- A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Containers vs VMs

- VMs use hardware virtualization (OS + programs)
- Docker containers don't use hardware virtualization: they directly interface with host's Linux kernel



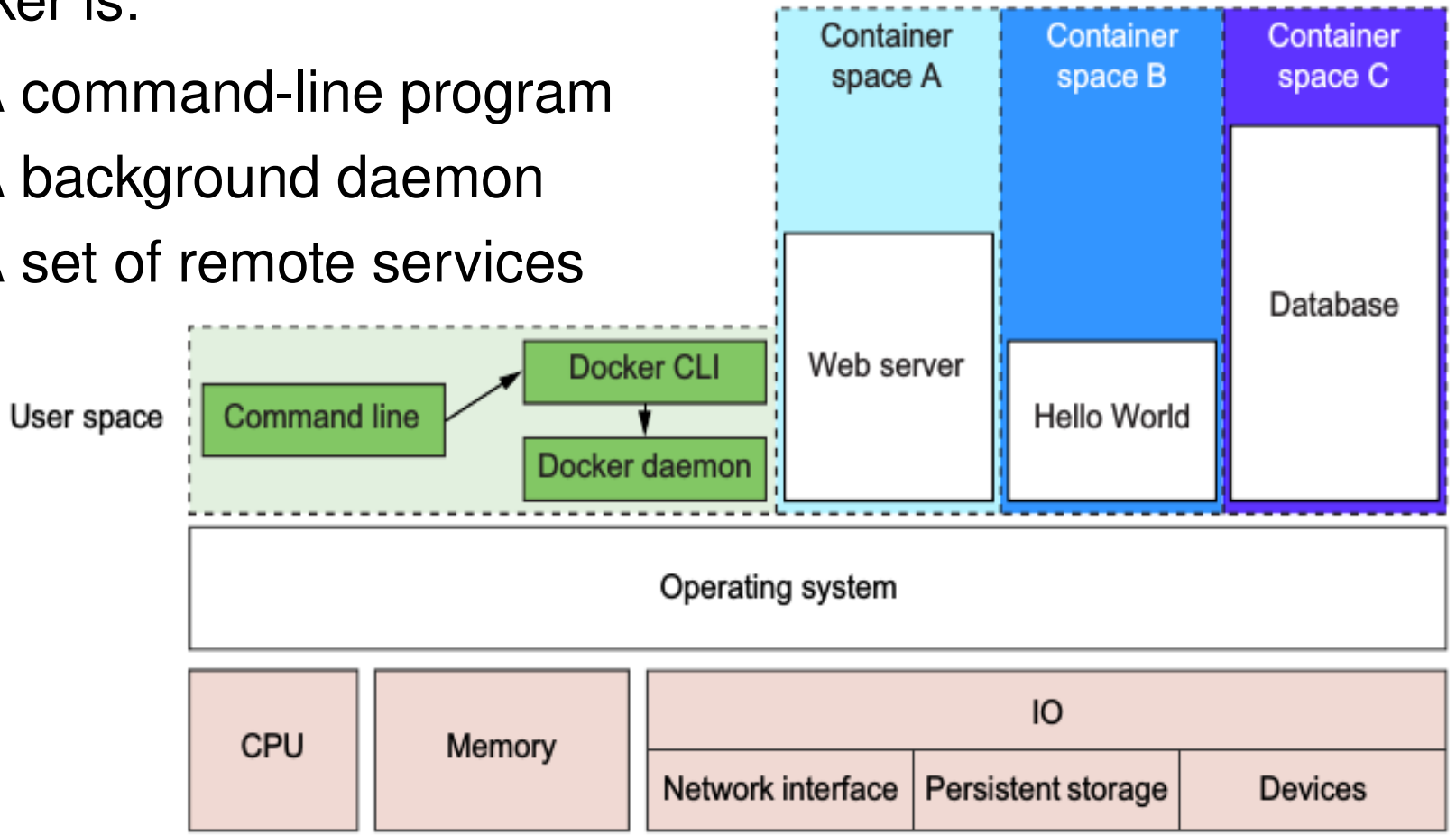
Server with Virtual Machines



Server with Docker Containers

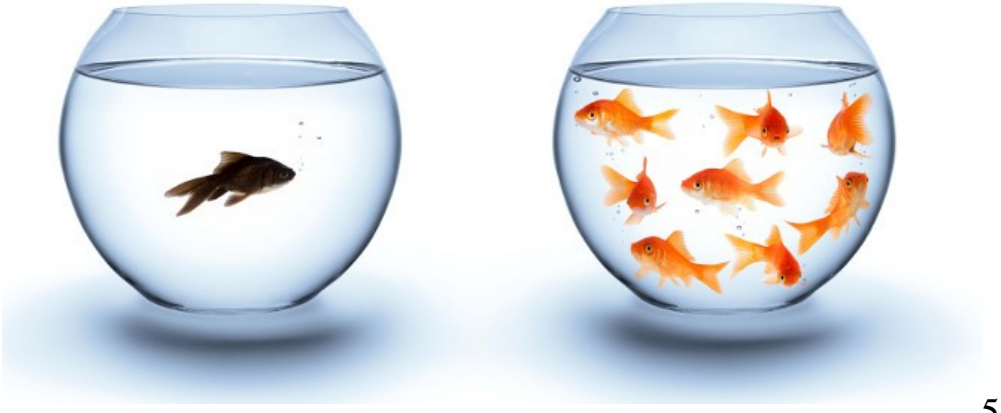
- Example with Docker and 3 containers
- Programs running in a container can access only memory and resources as scoped by the container

- Docker is:
  - A command-line program
  - A background daemon
  - A set of remote services



- Containers that Docker builds are isolated with respect to 5 aspects:

1. Process identifiers and capabilities
2. Host and domain name
3. File system access and structure
4. Network access and structure (interface)
5. Resource protection (CPU, RAM, disk, ...)



Docker containers

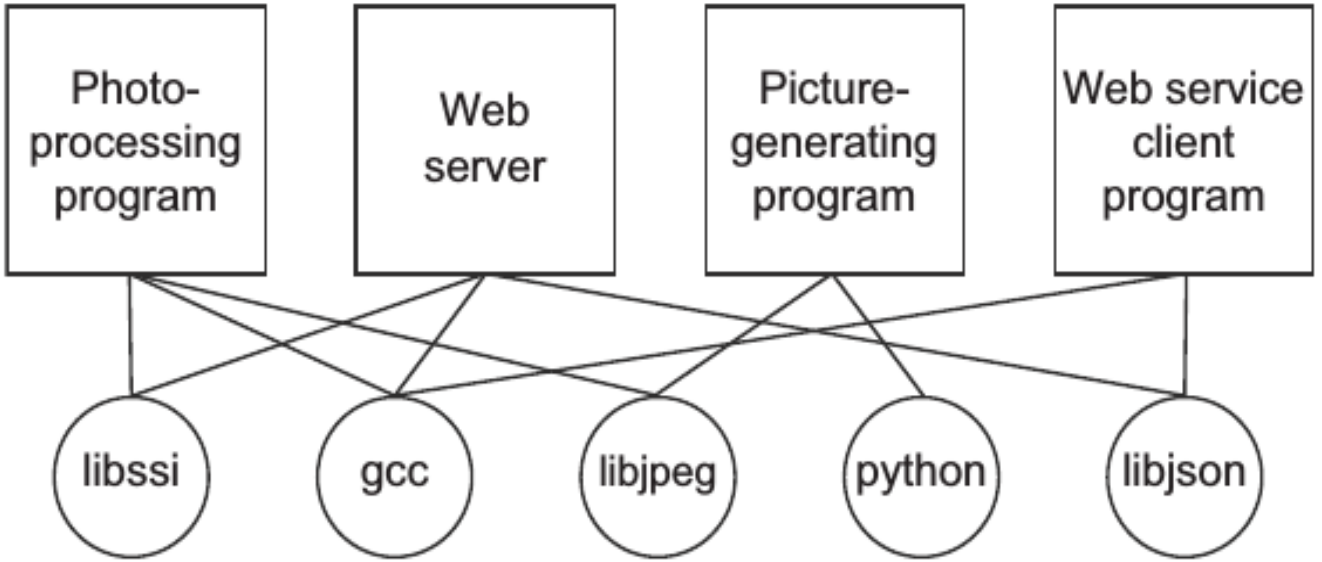
- A docker container is like a physical shipping container
- Docker can run, copy and distribute containers
- A container is filled with an image (snapshot of all files available to a running program inside a container)
- Images are distributed using registries (public or private) like hub.docker.com

Why Docker?

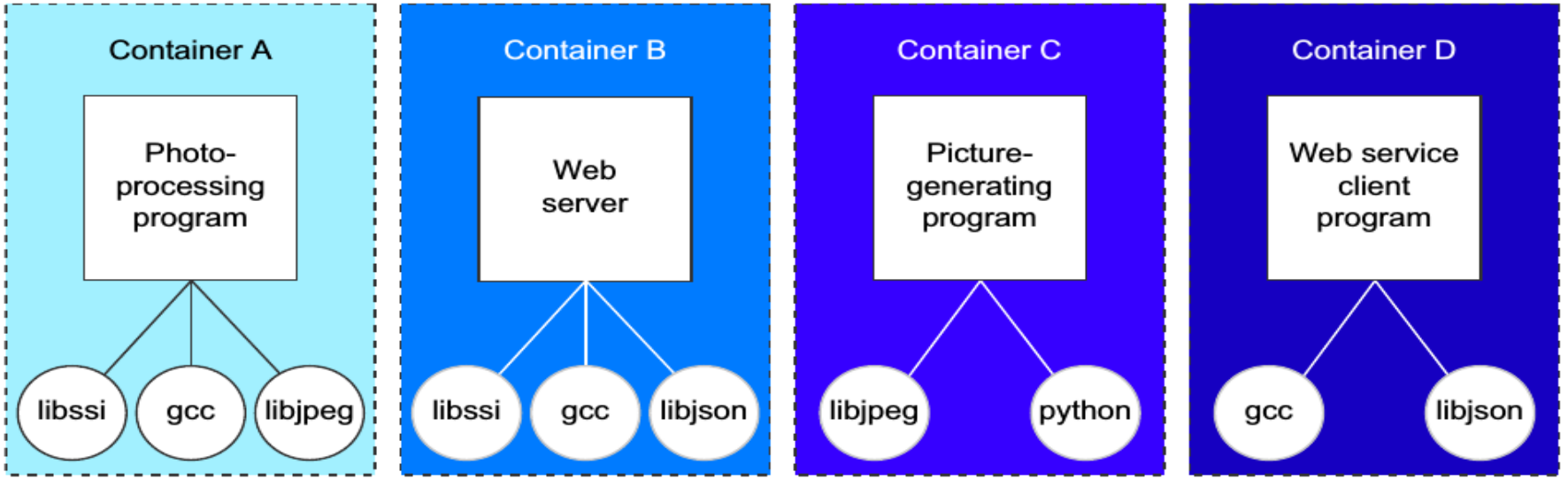
- Software deployment is a complex task
- A lot of considerations before acting:
  - How many resources?
  - Which dependencies?
  - What about other running applications?
  - Security?
  - Updates?

- The more software, the more difficults to manage

Before Docker...

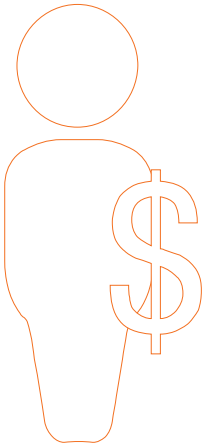


With Docker...



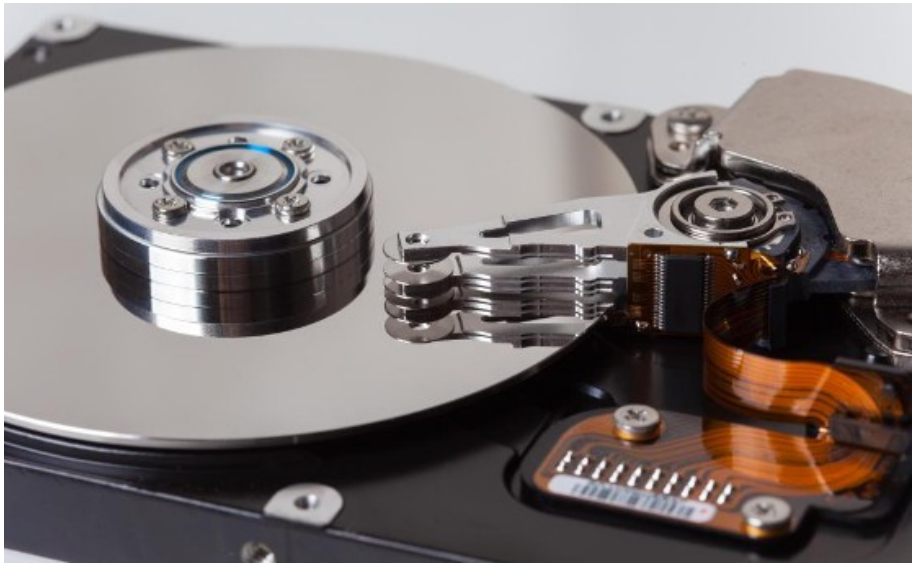
Advantages with Docker

- Improving portability
- More abstraction (containers are movable, resizable)
- Consistent and specific environments
- More and better focus on software development
- Company infrastructure as a cloud
- More protection because of isolation:
  - Anything inside a container has access only to a limit set of resources
  - In case of failure or malicious software, the problem is limited to a single container
- Save time, money, energy
- Companies like Amazon, Google, Microsoft contribute, support and push Docker adoption (instead of developing their own solutions)



Volumes

- So far we've seen ephemeral containers: data are not stored on a phisical support and are lost when the container is turned off
- In Docker volumes are used to persist data
- A volume is a mount point on the container's directory tree



Images vs Volumes

	Type of data	Specialization
Images	static	not specialized
Volumes	dynamic	specialized

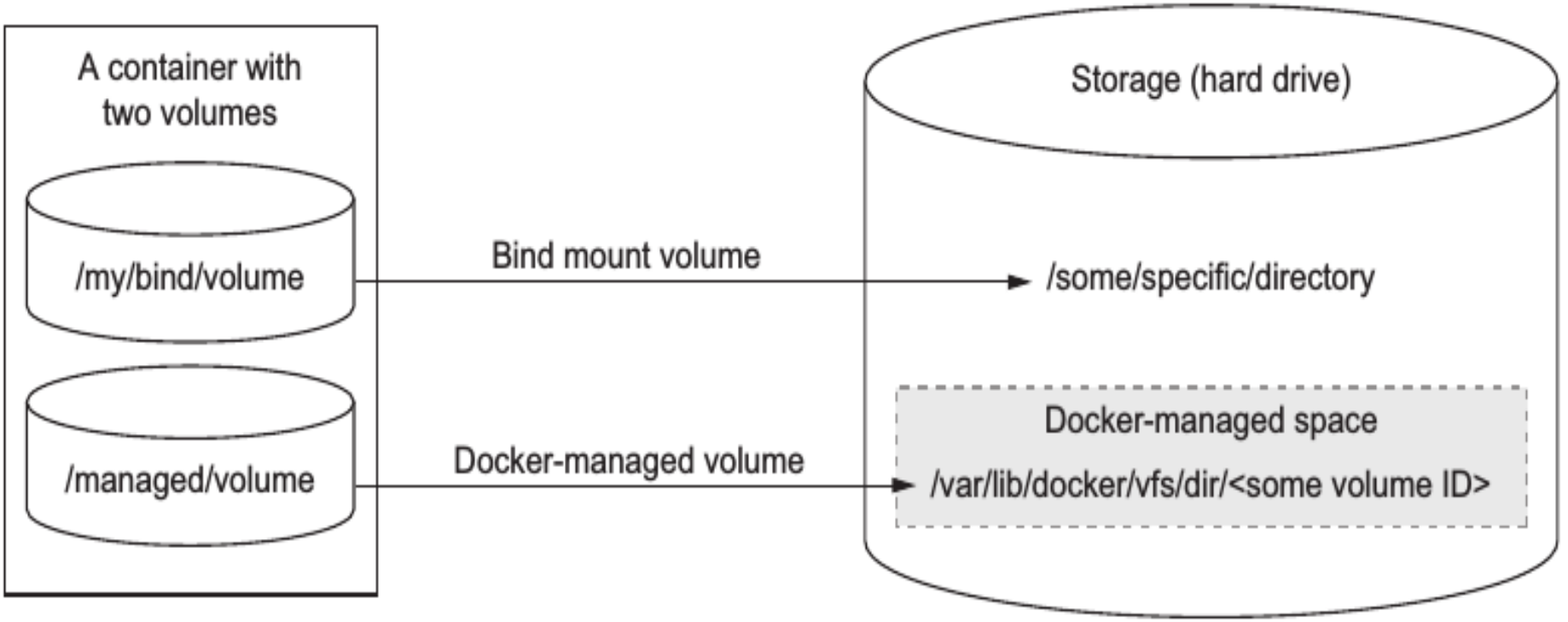
Volumes:

- enable the separation of application and host concerns
- allow implementation of advanced patterns like polymorphic and composable tools

Same interface,
but different implementations

Volume types

- 2 types of volumes:
  - Bind mount: directory or file on the host OS
  - Docker-managed: space controlled by Docker daemon

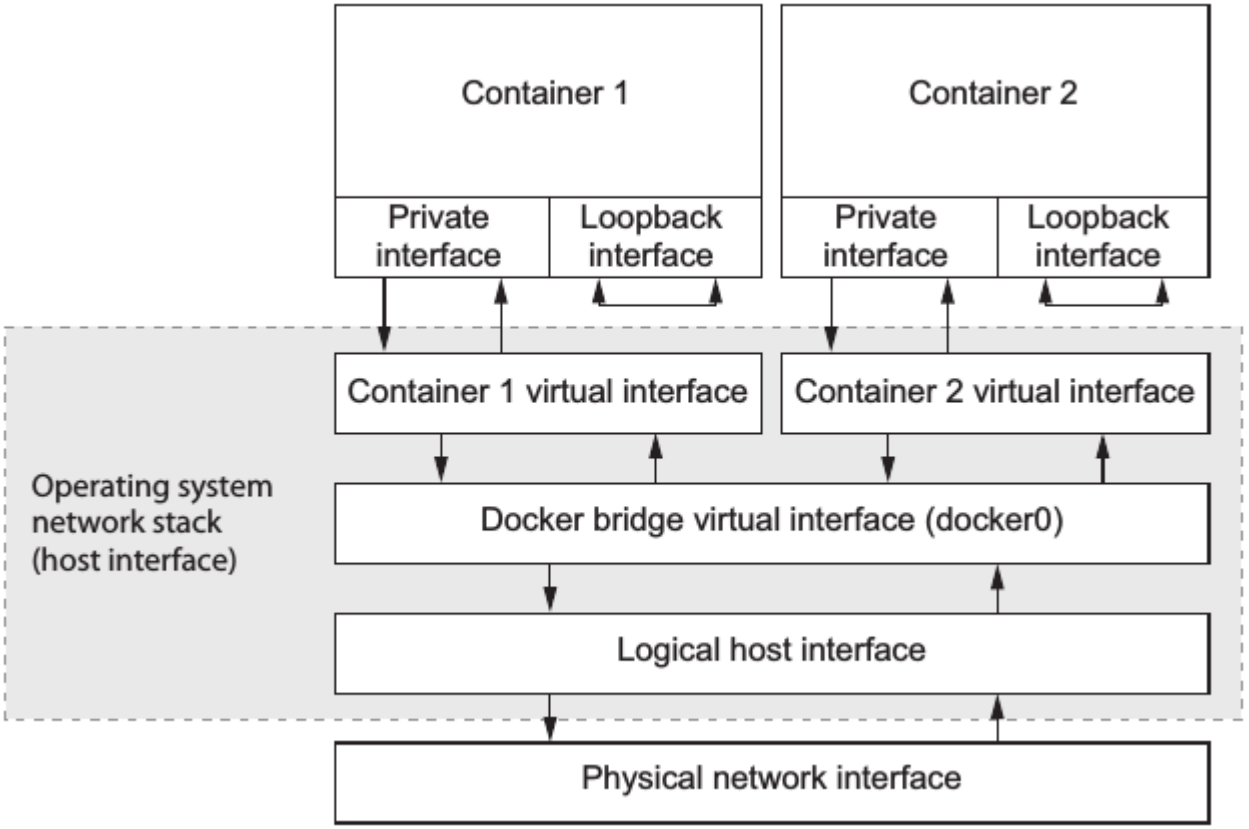


Sharing volumes

- You can share a volume between containers
- 2 types of sharing volumes:
  - **Host-dependent**: two or more containers mount on the same host's folder (-v option, as seen before)
  - **Generalized sharing**: using --volumes-from <container>
- You can copy volumes directly or transitively
  % **docker run --name D --volumes-from C alpine:latest**
- Data-only containers are used by other containers to mount data volume (even if DOc are in stopped state)

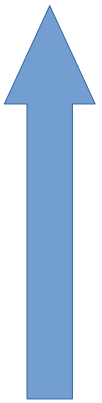
Networking

- How to connect a container to a network?
- Docker uses underlying OS to build a virtual network



Network archetypes

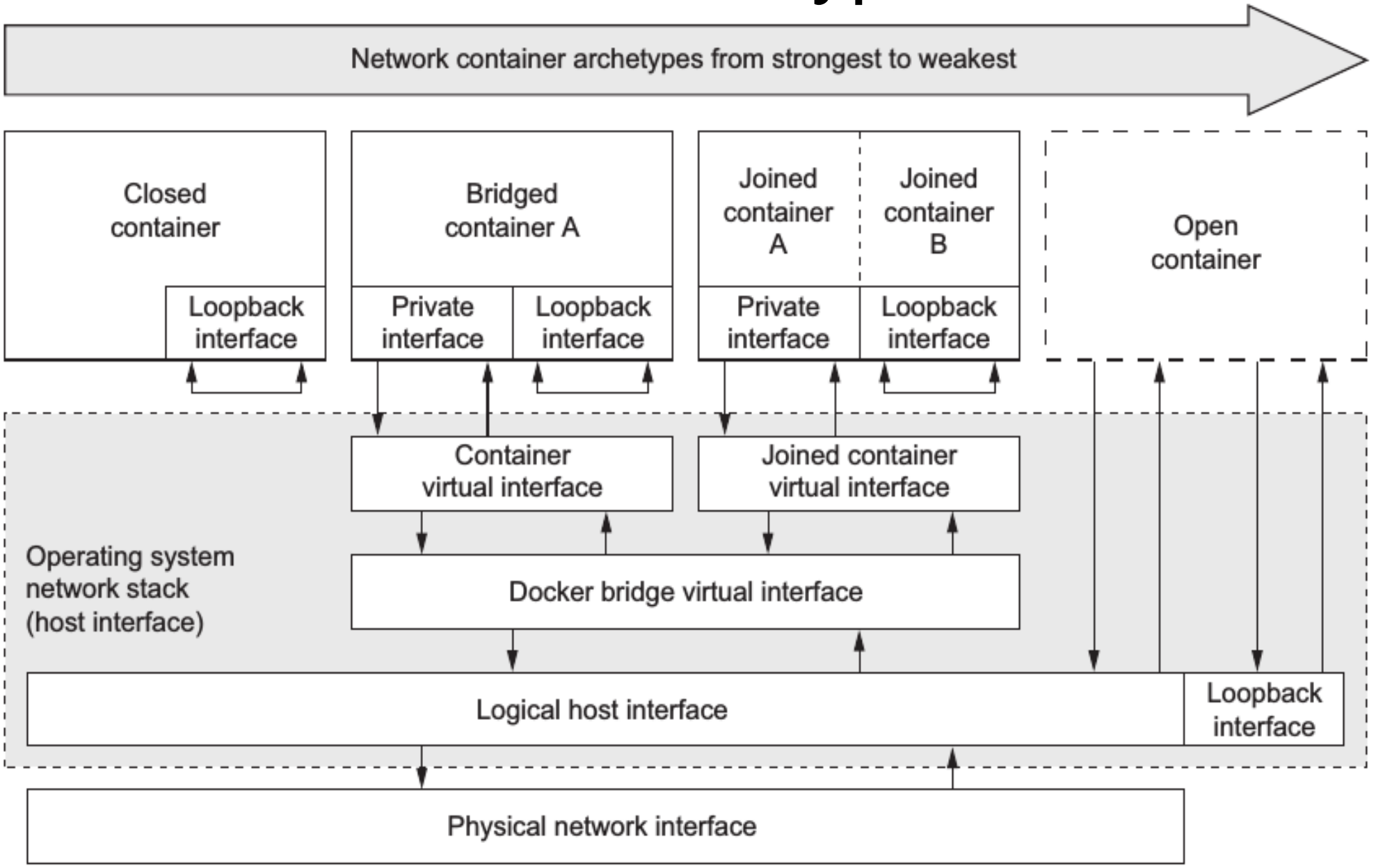
- How a container interacts with local and host's network is defined by 4 archetypes:
  - Closed container
  - Bridged container
  - Joined container
  - Open container



Level of isolation

- They define a different level of isolation

Four archetypes

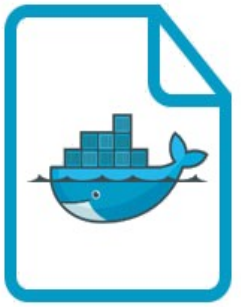


Inbound communication

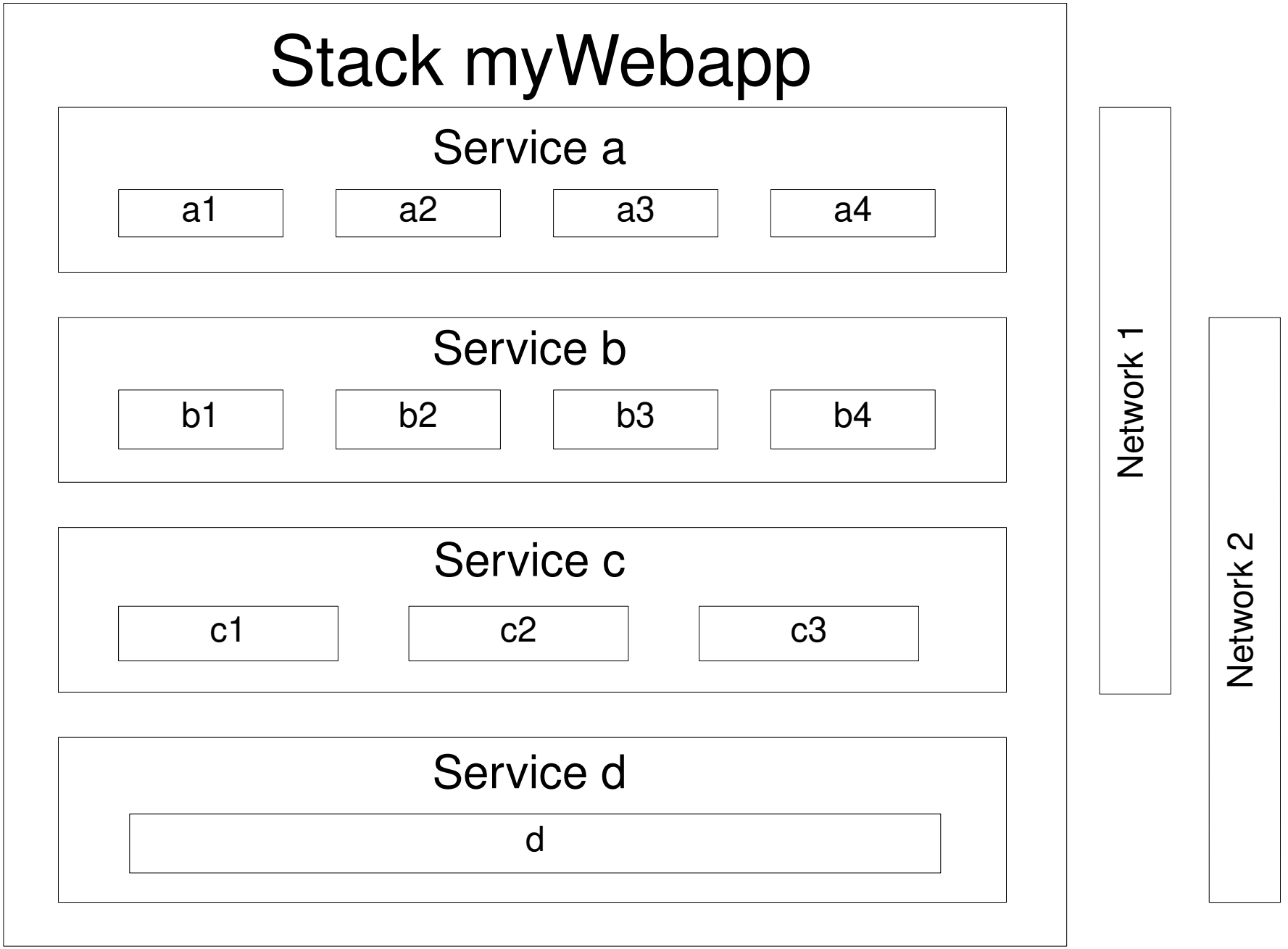
- By default a bridged container is not accessible from the network
- Option **--publish = [ ]** (or -p=[ ]) creates a mapping between a network port's host and container's interface
- 4 formats for the mapping:
  - <containerPort>
    % **docker run -p 6789**
  - <hostPort>:<containerPort>
    % **docker run -p 4444:5555**

Build automation

- Dockerfile is a file containing instructions for building an image
- ### Dockerfile example ###
  FROM php:latest
  COPY myapp/* /var/www/html/
- **docker build --tag myapp:1.0 .**

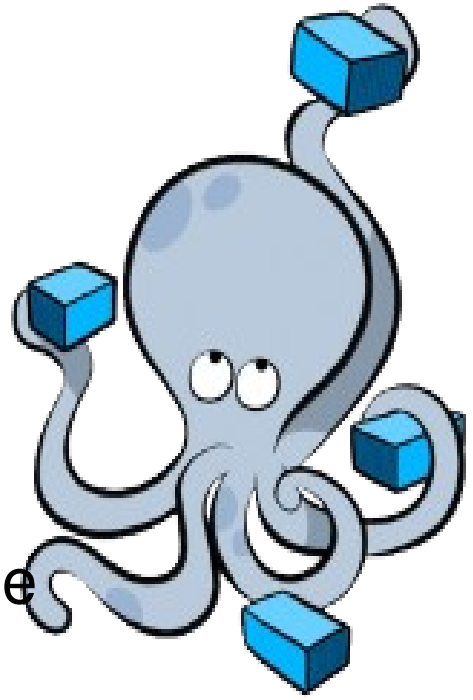


Multiple containers (service + stack)



Docker Compose

- It's a tool for defining and running multi-container Docker applications
- Systems of services are defined in YAML files
- **docker-compose** let you:
  - Build docker images
  - Launch containerized applications
  - Launch systems of services
  - Manage the state of a service
  - Scale up/down
  - View logs
- Project on https://github.com/docker/compose



Orchestration

- Some questions to solve:
  - Selection of a machine → scheduling
  - Advertise the availability of a service → registration
  - Resolve the location of a service → service discovery
- Specific tools were developed for these tasks:
  - Docker Swarm
  - Kubernetes
  - Apache Mesos

Apache MESOS™

kubernetes