

Docker



Level 2

Présentation disponible à l'URL: <https://dduportal.github.io/slides/2020-arexo-docker-level2>








Présentation disponible en PDF: <https://dduportal.github.io/slides/2020-arexo-docker-level2/slides.pdf>

Comment utiliser cette présentation ?

- Pour naviguer, utilisez les flèches en bas à droite (ou celles de votre clavier)
 - Gauche/Droite: changer de chapitre
 - Haut/Bas: naviguer dans un chapitre
- Pour avoir une vue globale : utiliser la touche "  " (pour "**O**verview")
- Pour voir les notes de l'auteur : utiliser la touche "  " (pour "**S**peaker notes")

Whoami

- Damien DUPORTAL:
 -  Tooling Engineer at OpenIO-OVH & Freelancer
 -  Former Traefik's Developer Advocate @ Containous
 -  Former Jenkins' Training Engineer @ CloudBees
-  damien.duportal+pro <chez> gmail.com
- @DamienDuportal
-  dduportal

Menu

- Prerequisites from Level 1
- Docker Networks
- Docker Volumes
- Docker CLI

Prerequisites from Docker Level 1

Container vs. Image

- What is the difference between a container and an image?
- What is the link between image and container?

Container

- How to run a container?
- Container lifecycle? How to check for running containers? And stoped containers?
- Difference between `-it` and `-d`?
- How to access a running container?

Image

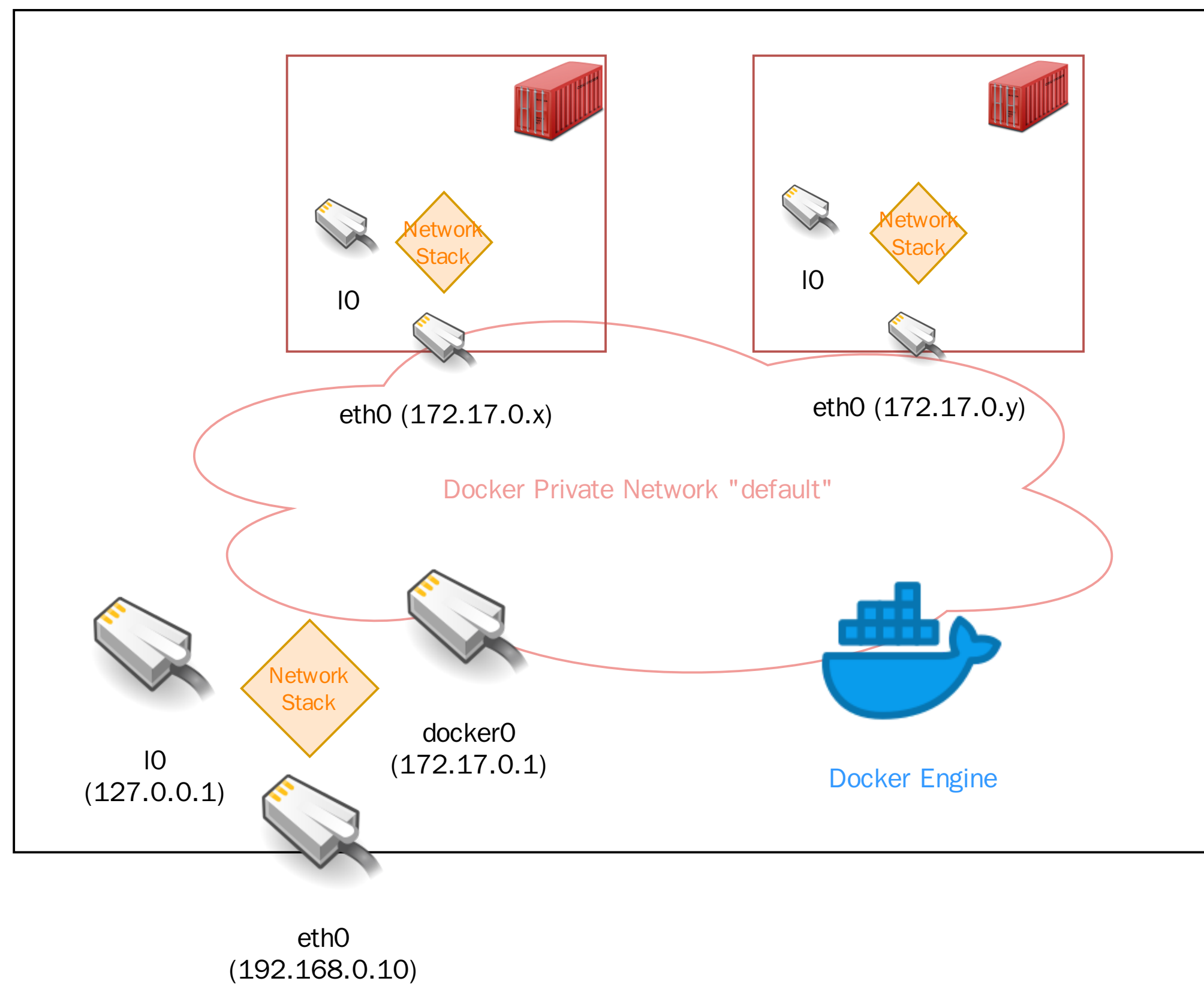
- How to list images?
- Image naming? Where to check these image?
- How to build an image?

Docker Networks

Container Default Network

- A container has its own network stack,
- and is attached to a default private network,
- with a default gateway, which is the host's `docker0` interface

Default Network Diagram



Exercise: Container Default Network

```
docker run --name=web1 --detach nginx:alpine
```

```
docker inspect # | grep IPA
```

```
curl -v 172.17.0.x # Result ?
```

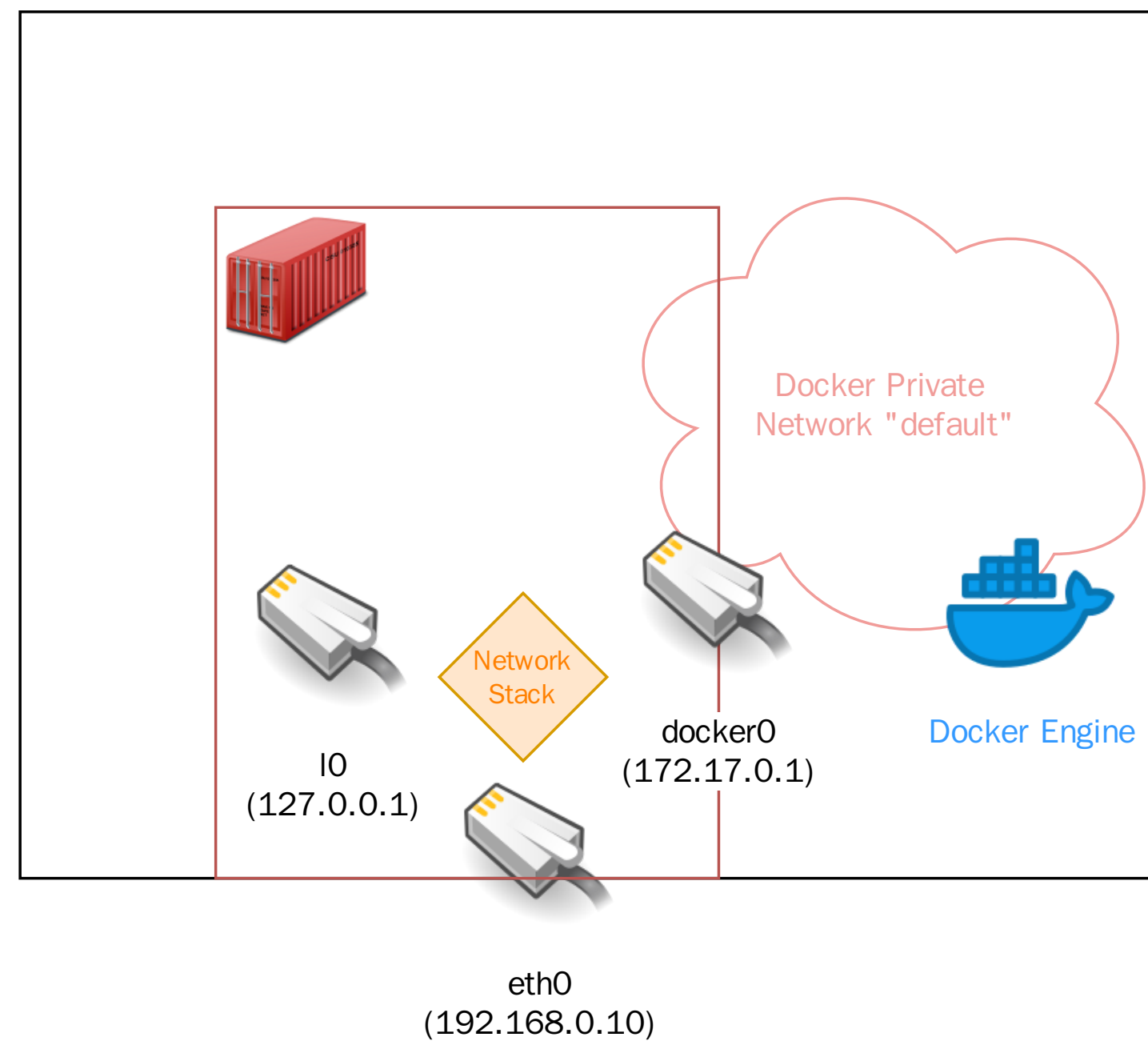
```
docker run --rm -ti alpine
```

```
apk add --no-cache curl
```

```
curl -v 172.17.0.x # Result ?
```

Host Network

A container can run on the host's network stack:



Exercise: Host Network

```
docker run --name=web2 --network=host --detach nginx:alpine
```

```
docker exec -ti web1 ip addr
```

Compare with

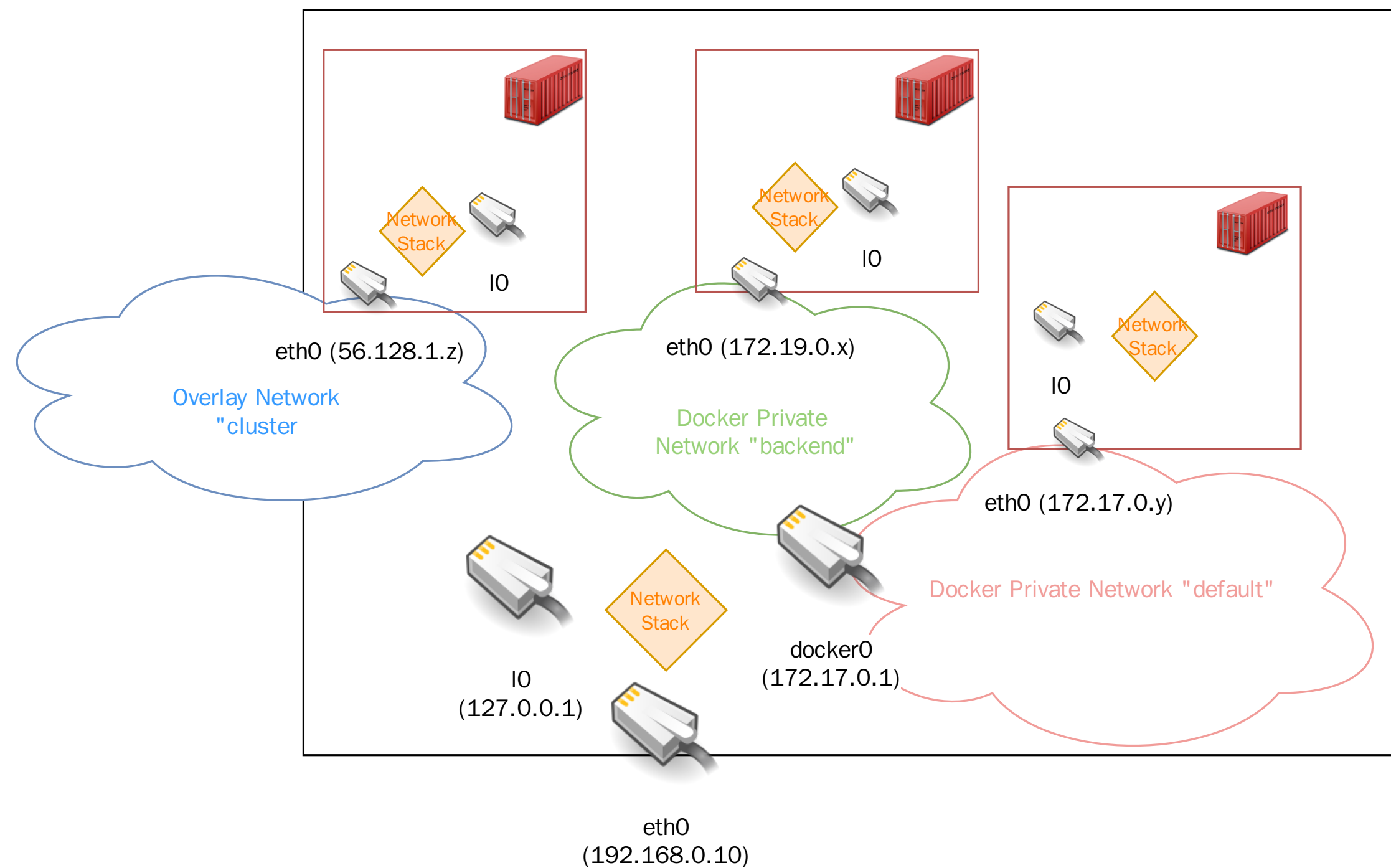
```
docker exec -ti web2 ip addr
```

Docker Networks

Docker allows to manage multiple isolated networks, each one with an implementation driver.

Default is a "bridge" driver, but overlay or custom networks can be used (clusters, VLANs, etc.)

Docker Networks Diagram



Exercise: Docker Networks

```
docker network ls
docker network create --attachable --subnet=192.168.0.1/24 backend # Run
docker network ls

docker run --rm -ti --network=backend alpine
  ip addr # eth0 ?
  apk add --no-cache curl
  curl 172.17.0.x # IP of web1. Result?

docker network connect backend web1
docker inspect web1 | grep IPA
docker exec -ti web1 ip addr
# Retry the "docker run --rm -ti --network=backend alpine" sequence again now
```

Publishing Container Ports

- How to access from the outside?
 - Publish ports with `-p` and `-P`

Exercise with Publishing Container Ports

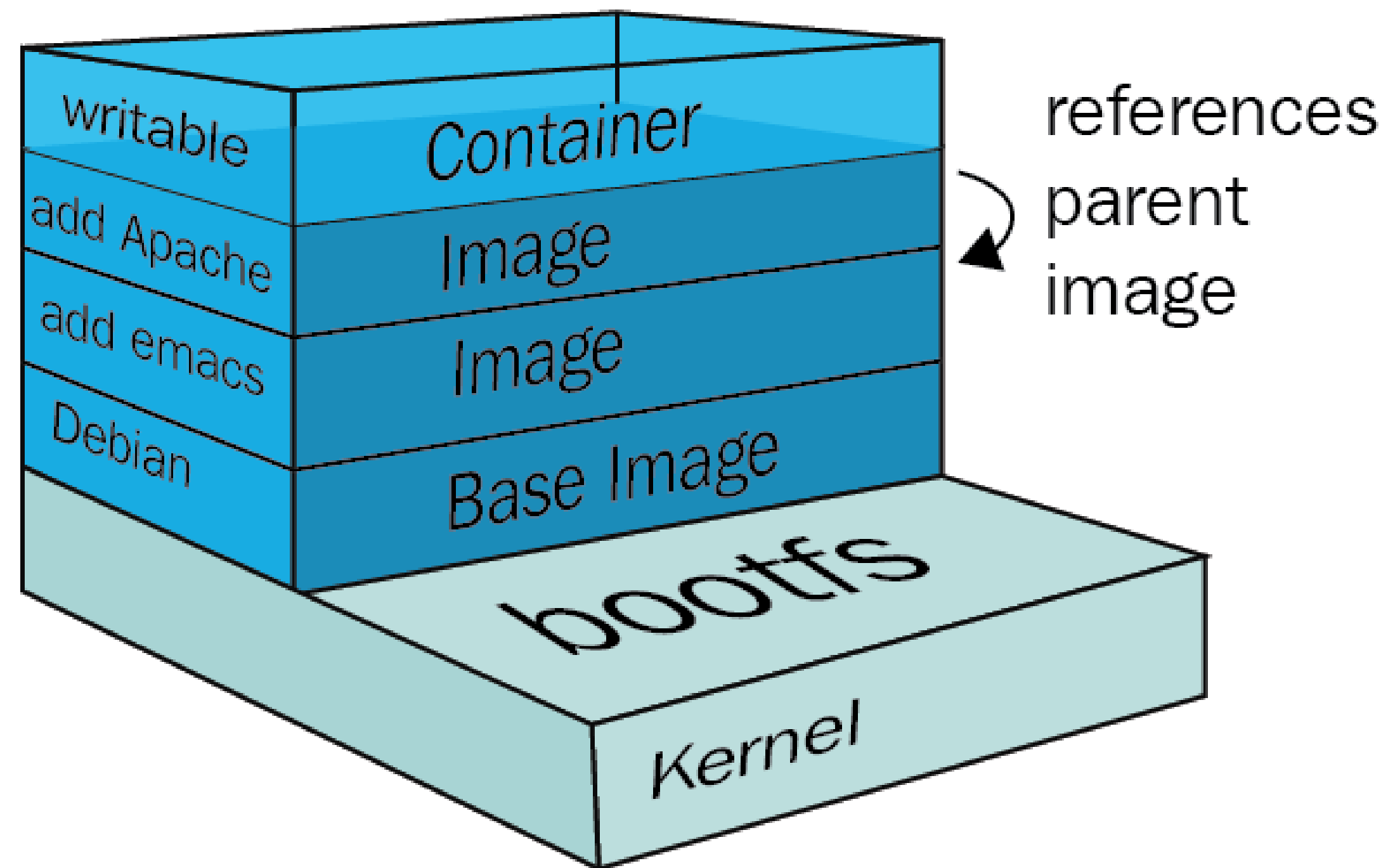
```
## For each: do not forget the "negative" testing:  
# - curl with the instance up should work  
# - Stop the instance with docker stop <instance name> and curl should NOT work  
  
docker run -d -p 80:80 --name=web3 nginx:alpine  
# Should be easy curl and docker ports/inspect  
  
docker run -d -P --name=web4 nginx:alpine  
# Search how to reach the published ports with curl and docker ports/inspect  
  
docker run -d -p 80:127.0.0.1:80 --name=web5 nginx:alpine  
# Search how to reach this instance with what we did
```

More about Networks to try...

- Container on the same network stack of another container
- Fixed IP addresses for a container
- IPv6 support
- Customisation of the default Docker bridge
- Docker compose and its networks

Docker Volumes

Images Layers



Exercise: Image with a Dockerfile

```
$ cat Dockerfile
FROM alpine:3.12
RUN apk add --no-cache sl
ENTRYPOINT ["sl"]

$ docker build -t sl:auto ./
$ docker run --rm -t sl:auto
```

Exercise: Image by Hand

Build the SAME image without a Dockerfile

```
$ docker run -ti alpine:3.12  
# ...  
$ docker commit ...  
$ docker run --rm -t sl:manual
```


Challenges with Layers

- Layer tracking is costly when doing a lot of I/O operations
- Concurrency concern: what if you commit while writing?
- By essence, containers are ephemeral and considered as immutables: what about the mutable persistent data?

Docker Volumes

- A volume is a set of data which lifecycle differs from a container. For instance: database data, cache, customer files...
- A volume is implemented through a driver:
 - Default is `local`, which is a "bind-mount" (host-native performances)
 - `tmpfs`, `NFS/SMB`,
https://docs.docker.com/engine/extend/legacy_plugins/#volume-plugins

Exercise with Volumes: Basics

```
docker run --rm -ti alpine
/ # mkdir -p /data
/ # echo "Hello" > /data/hello.txt
/ # exit
## hello.txt is lost because of --rm
```

```
docker volume ls
docker volume create customer-data
docker volume ls
```

```
docker run --rm -ti --volume=customer-data:/data alpine
/ # echo "Hello" > /data/hello.txt
/ # exit
## hello.txt still exists: find it on the Docker VM and/or with another container
```

Exercise with Volumes: Sharing between containers

```
docker volume create shared-data
docker run --rm -ti --volume=shared-data:/data alpine
/ # echo "SuperAdmin" > /data/root.txt
docker run --rm -t --volume=shared-data:/data jenkins/jenkins:2.235.5-alpine bash
bash-5.0$ cat /data/root.txt
bash-5.0$ rm -f /data/root.txt
bash-5.0$ ls -l /data/root.txt
```

Exercise with Volumes: Anonymous Volumes

```
$ cat Dockerfile
FROM alpine:3.12
WORKDIR /data
VOLUME ["/data"]
ENTRYPOINT ["echo", "bonjour", ">", "/data/bonjour.txt"]
$ docker build -t ano ./

$ docker volume ls
$ docker run ano
$ docker volume ls
```

Exercise with Volumes: Read-Only volume

```
docker volume create shared-data
docker run --rm -ti --volume=shared-data:/data:rw alpine
/ # echo "GuttenTag" > /data/de.txt
docker run --rm -ti --volume=shared-data:/data:ro alpine
/ # cat /data/de.txt
/ # ls -l /data/de.txt
/ # rm -f /data/root.txt
```

Exercise with Volumes: tmpfs

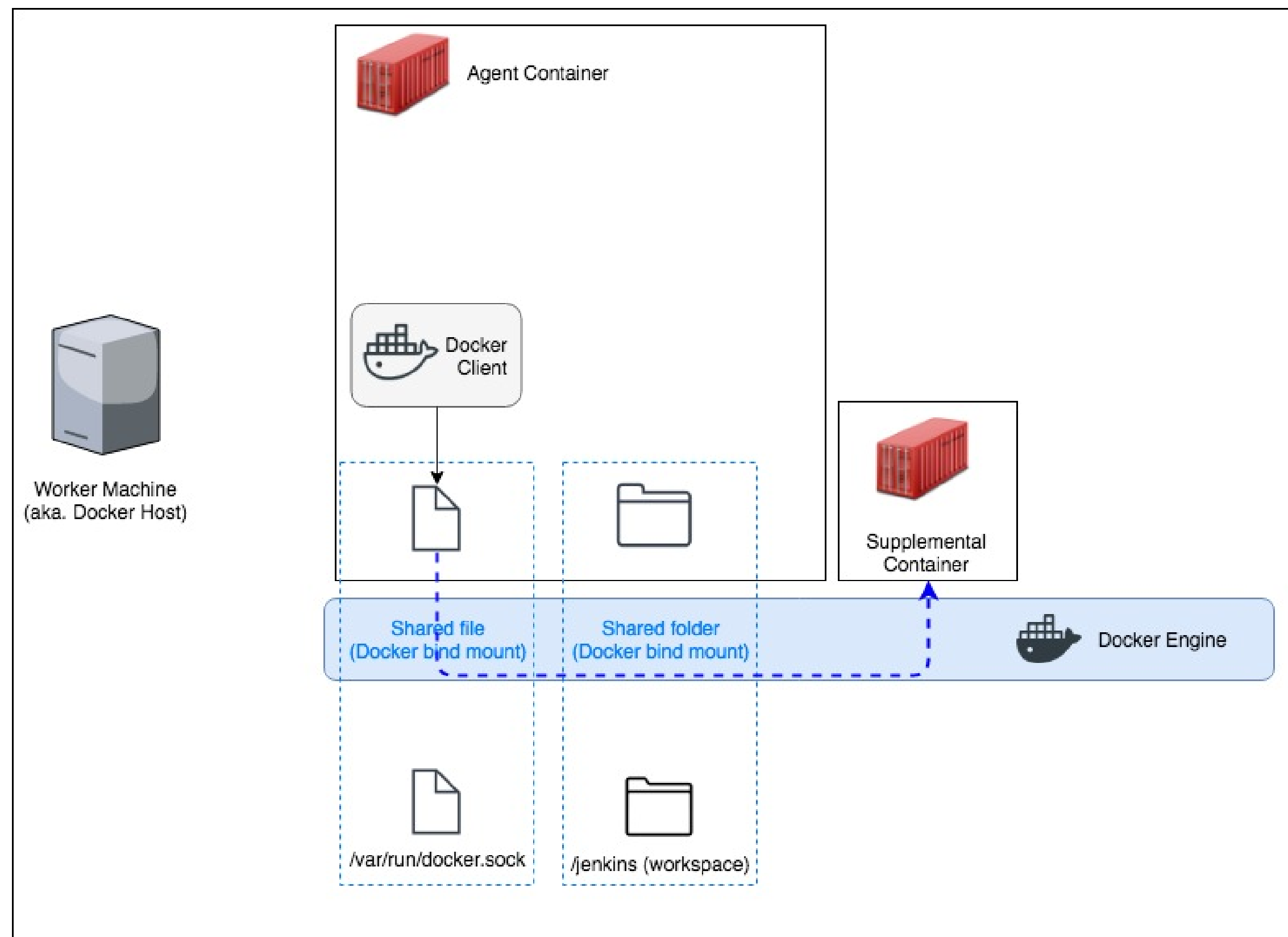
```
docker run --rm -ti --tmpfs=/temp_data alpine  
/ # echo "SuperAdmin" > /temp_data/root.txt  
/ # df -h
```

Docker CLI

Client / Server

```
which docker  
docker --version  
docker info  
curl --unix-socket /var/run/docker.sock http://containers/containers/json
```

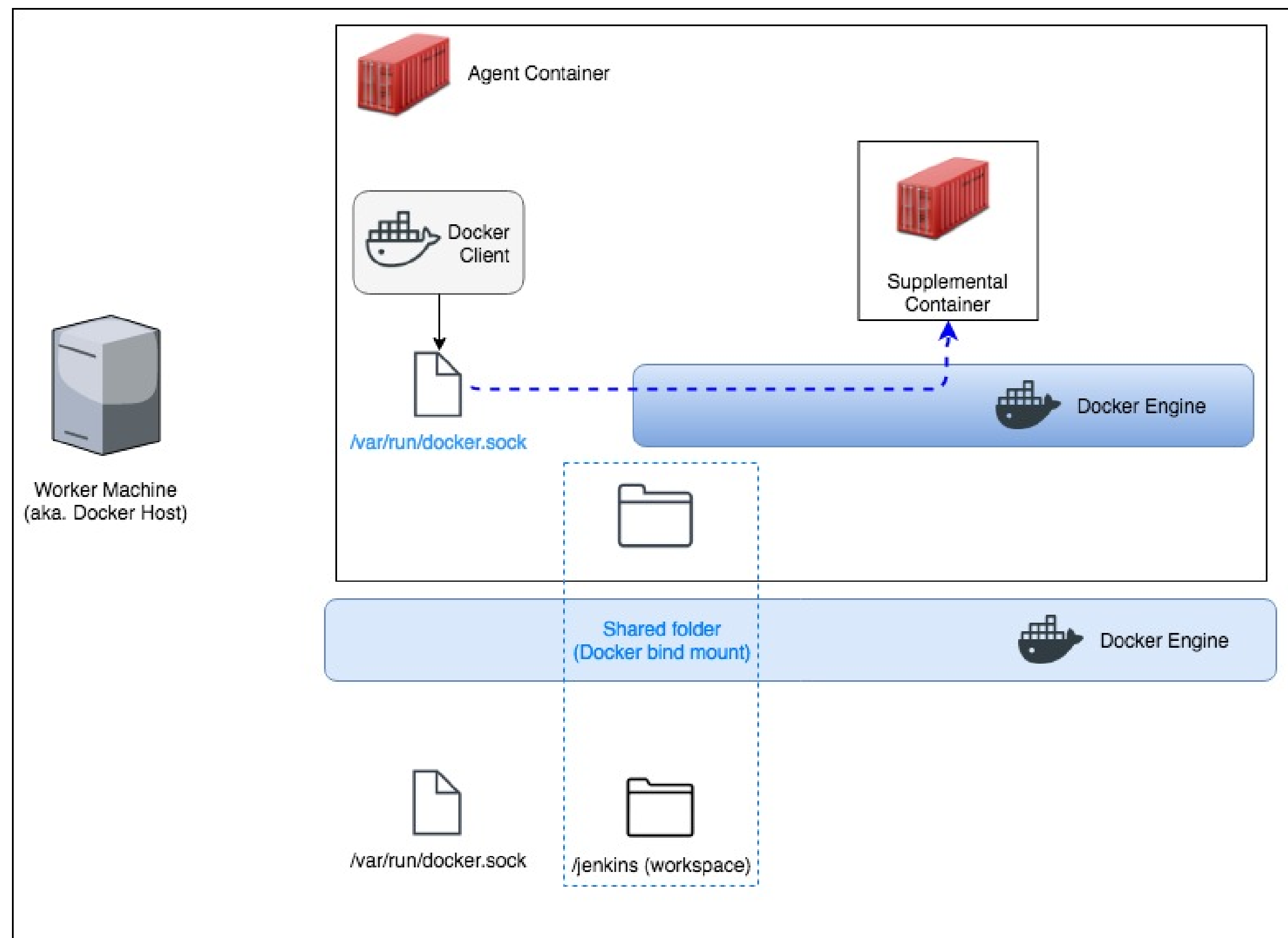
Docker on Docker



Exercise: Docker on Docker

```
docker run -v /var/run/docker.sock:/var/run/docker.sock:rw -ti alpine  
/ # apk add --no-cache curl docker  
/ # curl --unix-socket /var/run/docker.sock http://containers/containers/json
```

Docker in Docker



Exercise: Docker in Docker

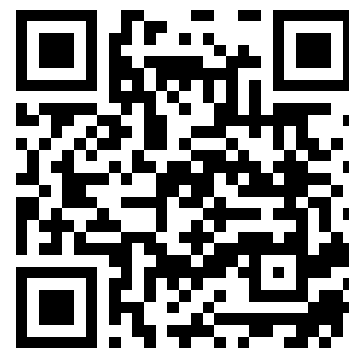
```
docker run --rm --name=dind --privileged docker:dind  
docker exec -ti dind bash  
/ # docker info  
/ # docker ps
```

Merci !

✉ damien.duportal+pro <chez> gmail.com

🐦 @DamienDuportal

Slides: <https://dduportal.github.io/slides/2020-arexo-docker-level2>



Source on : <https://github.com/dduportal/slides/tree/2020-arexo-docker-level2>