howest
university of applied sciences

Lecture VI
A DevOps Journey : Container Basics

From Virtual Machines to Containers
Understanding Container Technology Fundamentals

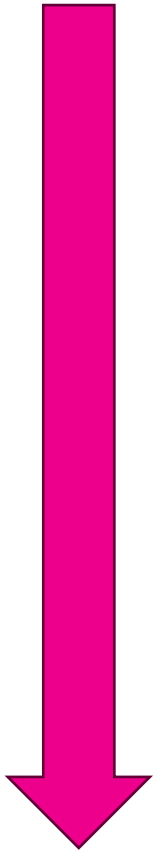
Guy Van Eeckhout

2025-2026

(inspired by Alex Desmedt)

What we'll talk about

- The historical evolution of container technology (1979-2013)
- How Docker emerged and revolutionized containerization
- Key milestones in Docker's evolution (2013-2025)
- How Docker positions itself against alternatives (Podman, Kubernetes)
- The concept of Docker registries and repositories



CONTAINERS VS VIRTUAL MACHINES (yay another recap)

Fun with VMWARE (until now)

- Multiple VMs on one physical machine
- Each VM: Full OS + Applications
- Hypervisor manages hardware access
- Heavy resource consumption

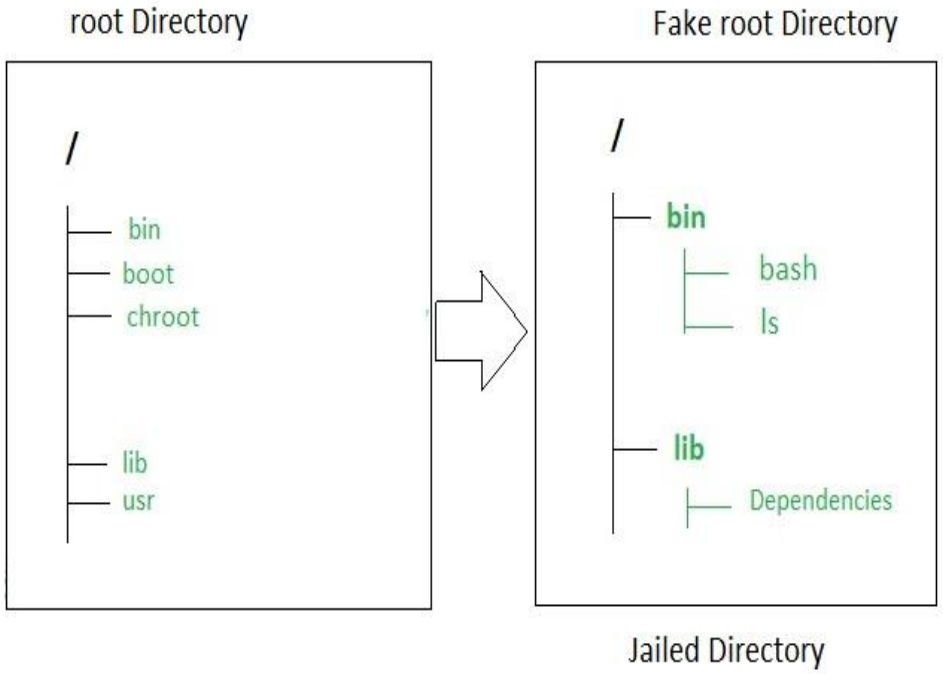
The container approach

- Multiple containers on one OS
- Shared kernel, isolated processes
- Container runtime manages isolation
- Lightweight and efficient

In short: Containers virtualize the OS, VM’s virtualize the hardware

Humble beginnings : 1979

- UNIX v7 introduces CHROOT
  - *Change root* command
  - Changes root directory of a process
  - First step toward process isolation
- What was the purpose of chroot ?
  - Originally used for testing new environments
  - Isolated filesystem access for processes
  - Created "chroot jail" concept
- Limitations
  - Only filesystem isolation
  - Processes could still communicate
  - Not suitable for security isolation alone



Still available on current linux systems !

21 years later : 2000

- FreeBSD Jails (2000) - Beyond Filesystem Isolation
- Building further on chroot:
- Developed by FreeBSD for hosting providers
- Addressed security and administration needs
- Partitioned FreeBSD system into independent "jails“

Enhanced Features:

✓ Own IP address per jail
✓ Own filesystem
✓ Own configuration
✓ Better process isolation
✓ Independent user spaces

Use Case:

- Hosting providers separating customer services
- Security enhancement over chroot
- True multi-tenant environment

Still available on current FreeBSD systems !

2001 : Vserver : Linux enters the fray

Parallel Development:

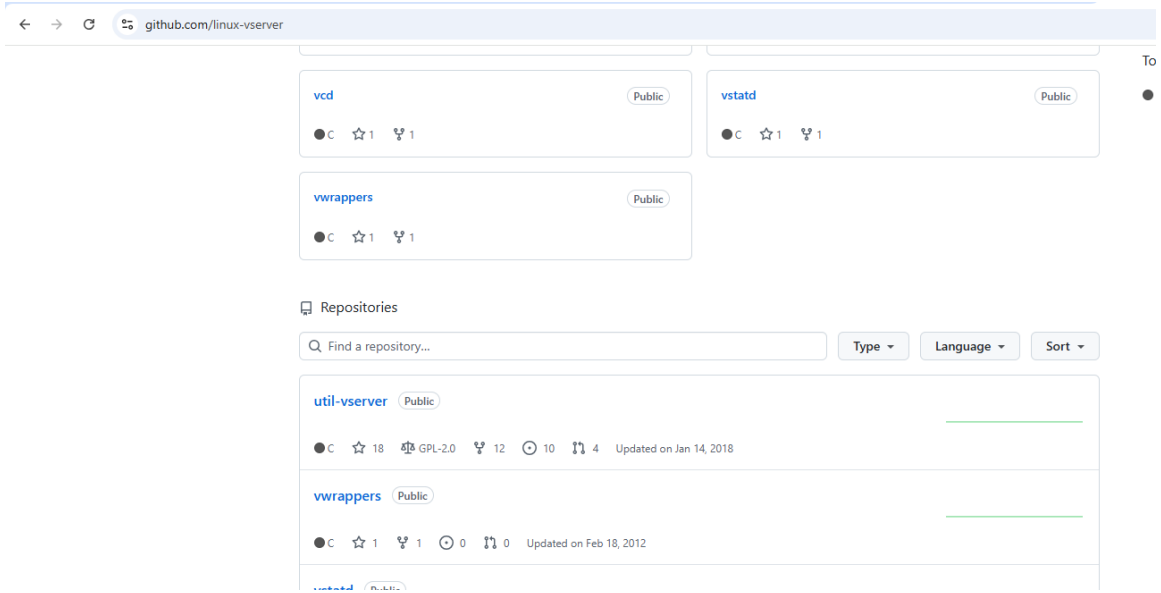
- Similar to FreeBSD Jails, but for Linux
- Operating system virtualization via kernel patches
- Created "Security Contexts" and VPS (virtual private server)

Vserver : Linux enters the fray

Isolated Resources:

- File systems
- Network addresses
- Memory allocation
- CPU time
- Process trees

Paved way for future Linux containers



Status: Last stable release: 2006

2004 – 2005 : Solaris Zones & OpenVZ

Solaris Zones (2004):

Introduced in Solaris 10 (Sun Microsystems/Oracle)

Lightweight virtualization for SPARC and x86

Combined resource controls + boundary separation

Two modes: Container mode & Whole system mode

OpenVZ (2005):

Developed by Swsoft

Patched Linux kernel approach

Similar features to Solaris Containers

Marked kernel-level virtualization was going mainstream

Impact: Enterprise adoption begins. Proven production-ready technology. Foundation for future container tech

2006 PROCESS CONTAINERS (CGROUPS)

Process Containers (2006)

- Later renamed "Control Groups" (cgroups)
- Merged into Linux kernel 2.6.24
- Limited, accounted, and isolated resource usage

cgroups control:

→ CPU allocation

→ Memory limits

→ Disk I/O

→ Network bandwidth

2008 : Linux Containers (LXC)

IBM development

BUT with mainstream Linux kernel integration

Combined cgroups + namespaces

First complete container solution

Used kernel features: namespaces, cgroups, chroots, capabilities

Foundation Complete: All pieces in place for modern containers!

howest
hogeschool

And now Docker

So turn off your lights

Reality Check from last week: No Docker vs Docker

No Docker

❌ "Works on my machine" syndrome

❌ Complex dependency management

❌ Different dev/test/production environments

❌ Difficult application deployment

❌ Lengthy setup times

❌ Environment configuration drift

Docker

✓ Package application + dependencies together

✓ Consistent behavior across environments

✓ Simple deployment process

✓ Fast startup times

✓ Easy sharing and distribution

✓ Developer-friendly tools

Solomon Hykes' Vision:"Shipping code to the server is hard" - make it easy!

2008-2010 : PLATFORM AS A SERVICE

Where Docker was born

Company Formation:

Founded 2008 in Paris

by Kamel Founadi, Solomon Hykes, Sebastien Pahl

Incorporated in US in 2010

Joined Y Combinator Summer 2010

The Vision:

- Platform-as-a-Service (PaaS) provider
- Enable easy cloud deployment
- Manage applications in the cloud

2008-2010 : PLATFORM AS A SERVICE

From an Internal Project:

- Docker started as internal tool (2010-2012)
- Solved dotCloud's own deployment challenges
- Used container technology to isolate processes
- Improved resource management for PaaSKey

💡 The tool they built was more valuable than the service!

PyCon March 13, 2013 : The day everything changed

The Big Reveal: Solomon Hykes demonstrates Docker publicly

5-minute demo that changed software development

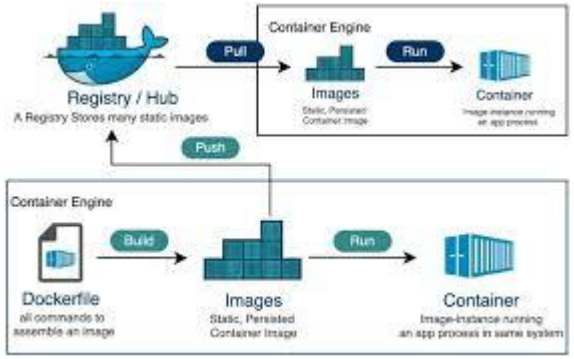
The Demo Message: "For developers, shipping to the server is hard"

Why Docker was revolutionary part 1

- Developer Experience:
  - Git-like semantics for containers
  - Simple CLI commands
  - Dockerfile for reproducible builds
- Standardized Format:
  - Universal container image format
  - "Build once, run anywhere“
  - Separates app from infrastructure

Why Docker was revolutionary part 2

- Ecosystem Integration:
  - Docker Hub for sharing images
  - Integration with existing tools
  - Rich community and marketplace
- Bridged Dev and Ops:
  - Developers focus on "inside container"
  - Operations focus on "outside container“
  - Reduced friction between teams



Docker Early Architecture : LXC Based

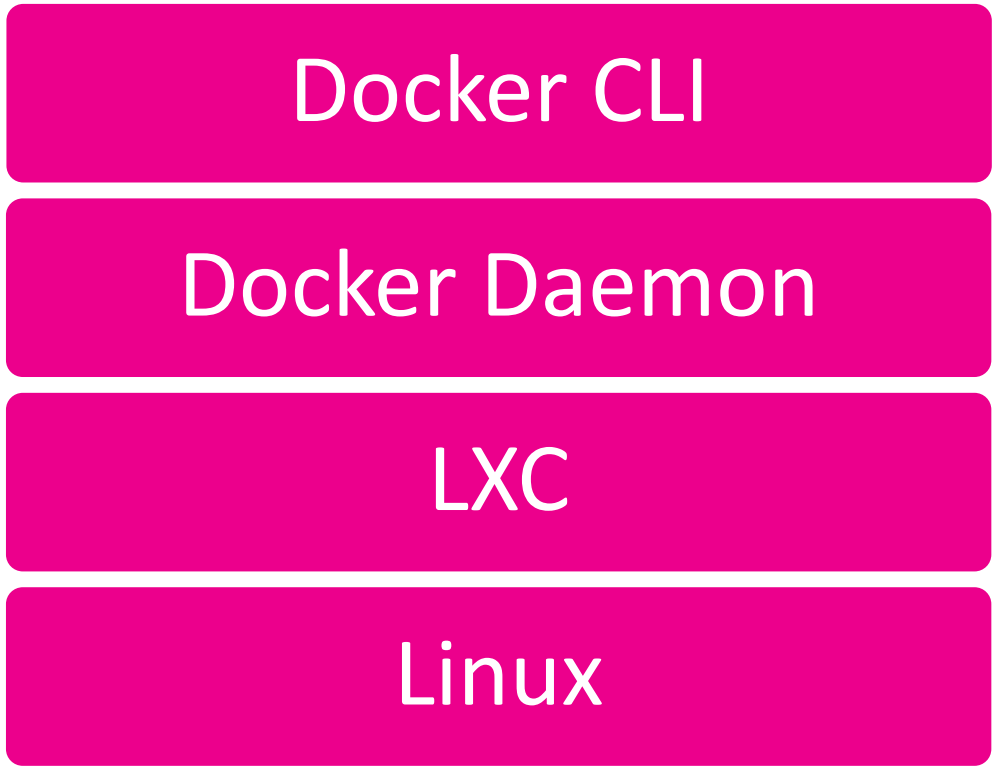
Initial Implementation:

- Built on top of LXC (Linux Containers)
- LXC as default execution environment
- Docker added user-friendly layer
- Simplified container management

Limitations of LXC Dependency:

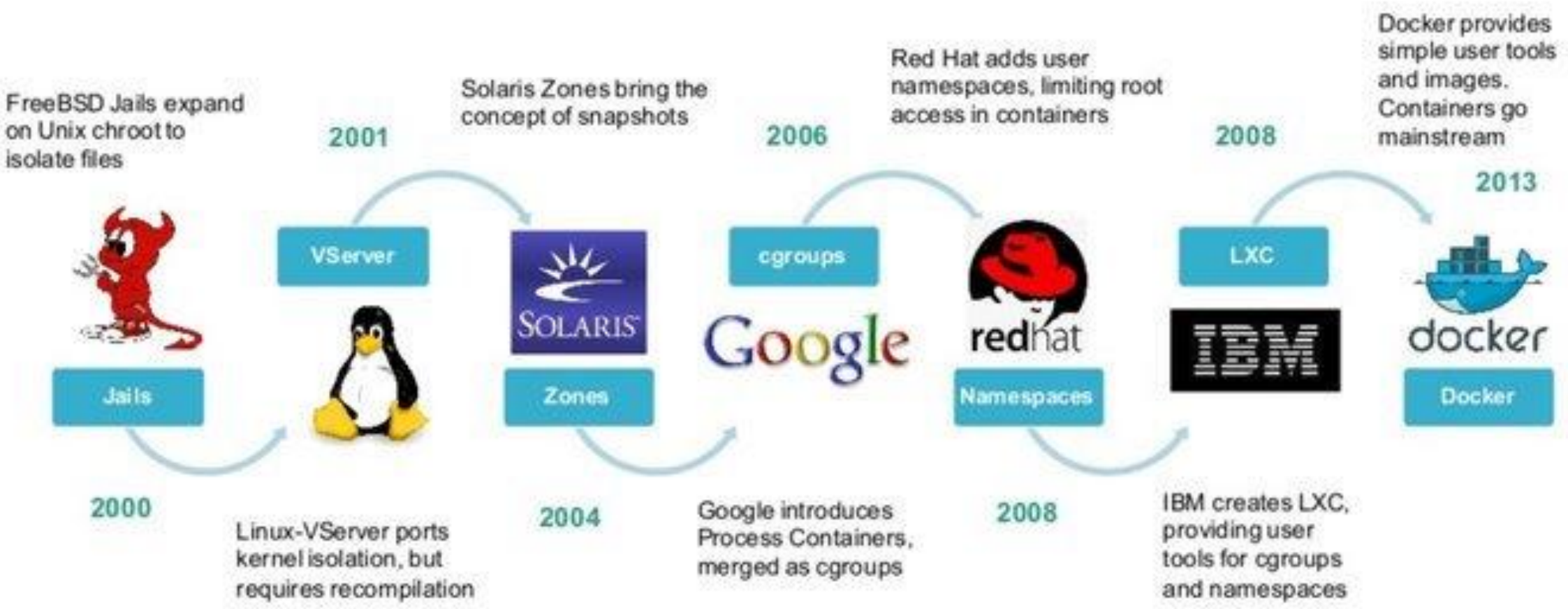
- Tied to Linux-specific implementation
- Limited portability
- Dependent on external project
- Needed more control

Architecture Components:



This led to Docker's biggest technical evolution...

We got to here : (recap)



DOCKER'S INNOVATION - SIMPLIFYING CONTAINERIZATION

Before Docker:

LXC existed but was complex

Required deep Linux knowledge

Manual configuration required

Steep learning curve

Docker's Abstraction:

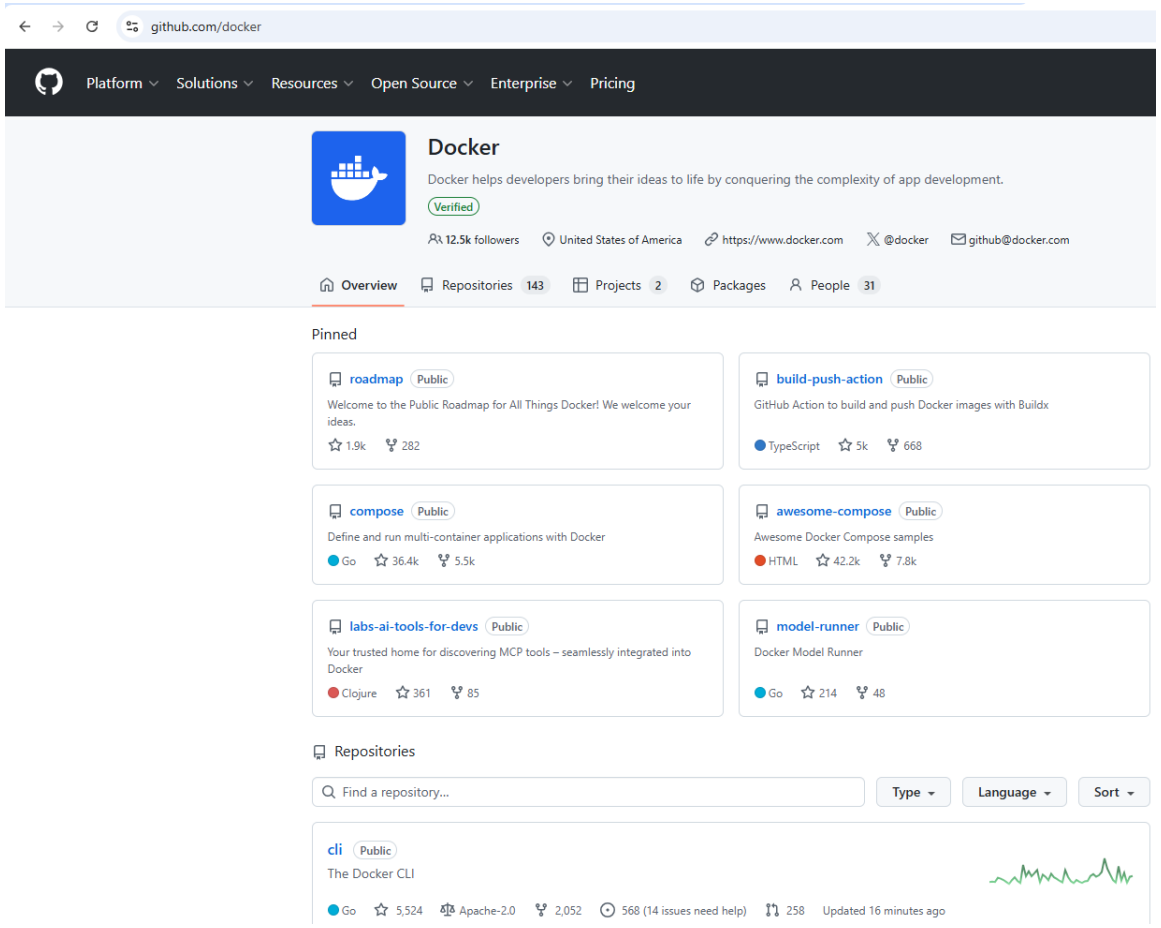
1. Dockerfile - Simple text file defines container
2. Docker Images - Layered, reusable, cacheable
3. Docker Hub - Central repository for sharing
4. Docker CLI - Intuitive commands
5. Docker Compose - Multi-container applications

docker run nginx

2013 : Docker goes open source

The Release:

- Code made publicly available March 2013
- Apache 2.0 License
- GitHub repository created
- Community development begins
- Impact of Open Source:
  - ✓ Rapid community adoption
  - ✓ Community contributions and feedback
  - ✓ Ecosystem development
  - ✓ Trust and transparency
  - ✓ Innovation acceleration

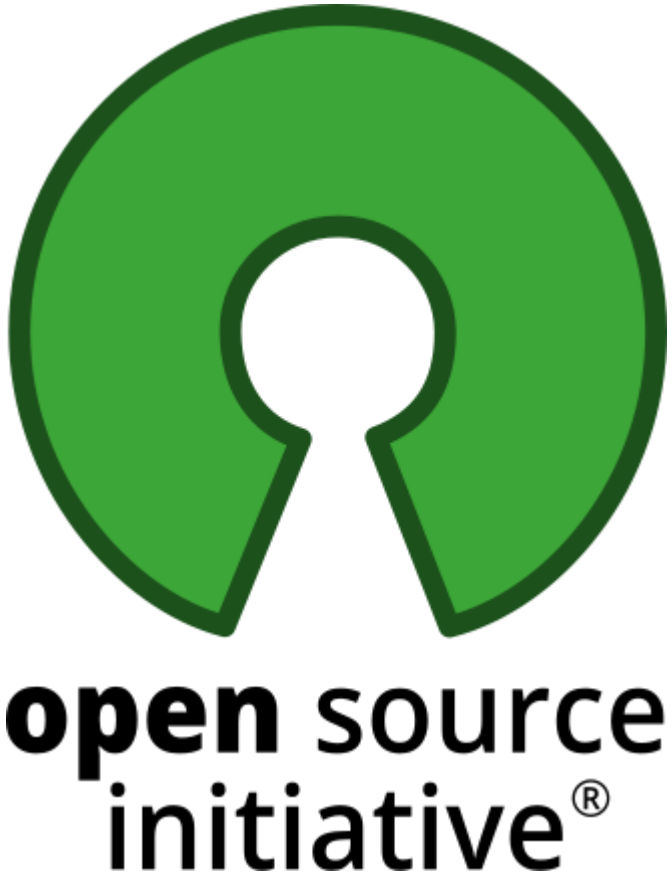


howest
university of applied sciences

2013 : Docker goes open source

By The Numbers (First Year):

- Thousands of contributors
- Rapid feature development
- Growing ecosystem
- Industry attention



Strategic Wisdom: Open sourcing Docker was key to its success

Enabled widespread adoption and innovation far beyond what a closed-source product could achieve.

2014 - FROM LXC TO LIBCONTAINER

The Transition:

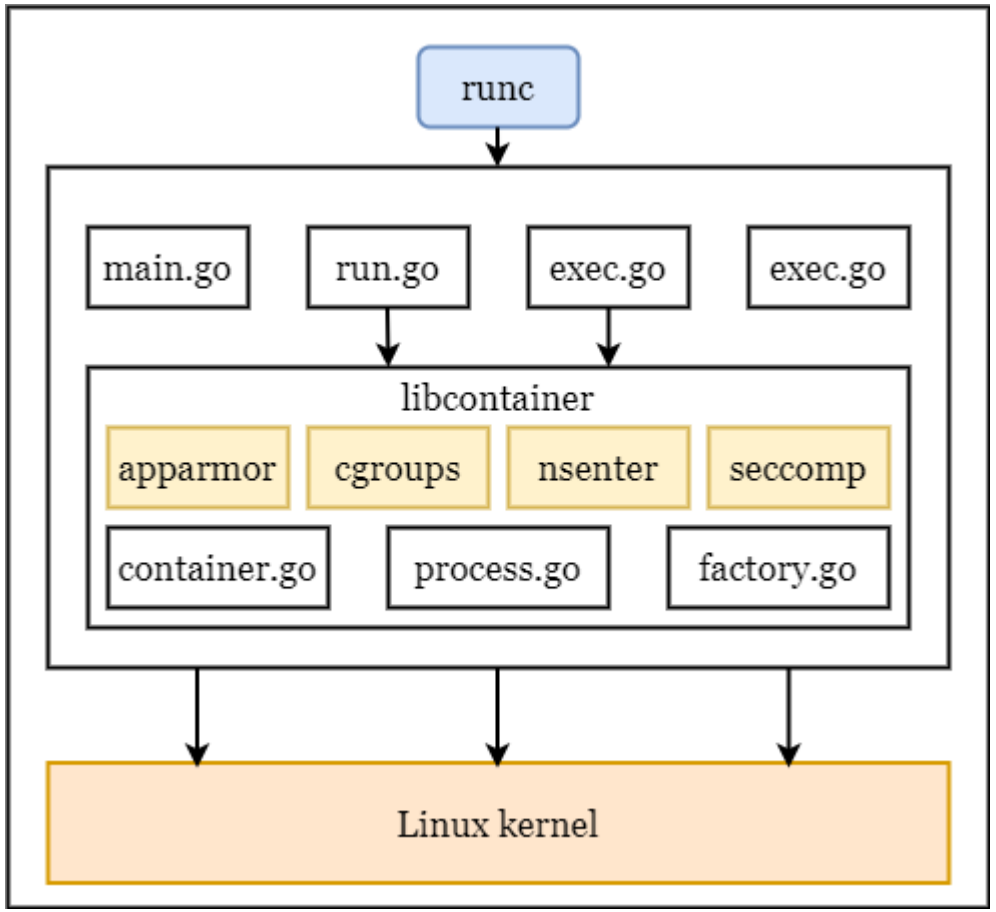
- Docker 0.9 release (March 2014)
- Introduced libcontainer (Written in Go programming language)
- Replaced LXC as default execution driver
- Why libcontainer?
    - ✓ Docker-owned and maintained
    - ✓ Cross-platform potential
    - ✓ Direct access to kernel features
    - ✓ Better performance
    - ✓ More control over functionality

2014 – LIBCONTAINER (evolved into => runc)

Technical Details:

- Direct interface to:
  - Linux namespaces
  - Control groups (cgroups)
  - Capabilities
  - AppArmor/SELinux profiles

Result: Docker became independent and portable!



2015 - THE OPEN CONTAINER INITIATIVE (OCI)

What is OCI?

- Founded June 2015
- Hosted by Linux Foundation
- Open governance structure
- Created by Docker, CoreOS, and others

Wat is its Mission ?

- Establish industry standards for containers:
    1. Runtime Specification - How containers execute
    2. Image Specification - Container image format

2015 - THE OPEN CONTAINER INITIATIVE (OCI)

Docker's Contribution:

- Donated runc (formerly libcontainer)
- Reference implementation of OCI runtime spec
- Later donated containerd

Why Standards Matter:

→ Prevents fragmentation

→ Ensures interoperability

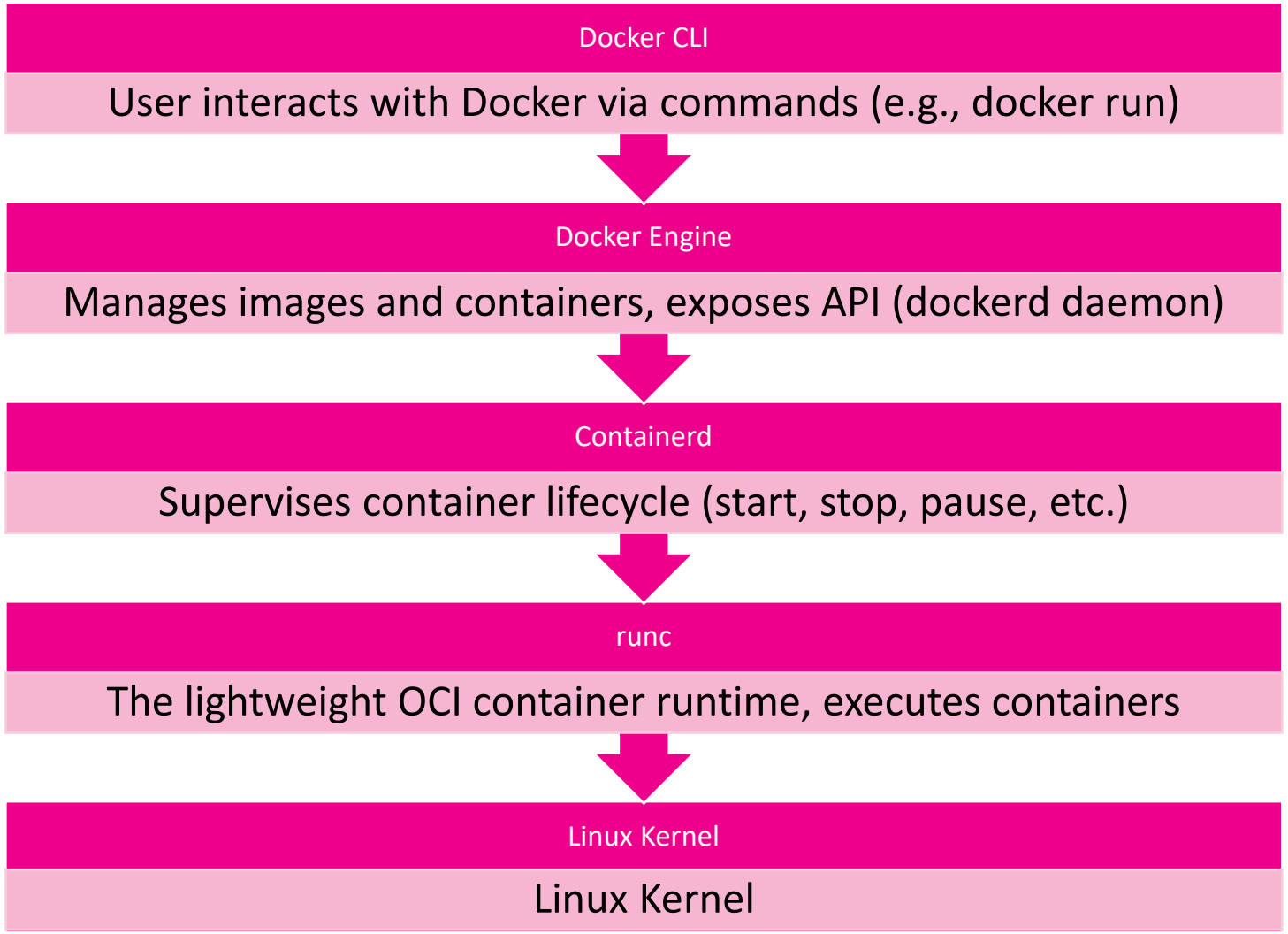
→ Enables innovation

→ Industry-wide adoption

→ Vendor neutrality

Impact: Docker containers work everywhere!

The current docker full stack



What about docker desktop ?

Layer	Role/Contribution
Docker Desktop	The orchestrator and facilitator for Docker on desktops
	- Provisioning VM: Runs a lightweight Linux VM using Hyper-V, WSL2, or Apple Hypervisor for kernel features on non-Linux OSes.
	- UI and Integration: Offers graphical dashboards, settings, extensions, and status monitors unavailable via plain Docker CLI.
	- Install/Run: Installs, launches, and updates Docker Engine and related binaries.
Docker CLI	User commands (text UI) get passed to the Docker Engine running inside the provisioned VM.
Docker Engine	Runs inside the VM, unchanged from how it would on a native Linux host.
containerd, runc, Linux Kernel	Operate inside the VM where Docker Engine runs, using Linux kernel features enabled by the virtual machine.

Docker desktop & Docker Hub

Docker Desktop:

Launched for Mac and Windows

2022: Linux support added

Local development environment

Integrated Kubernetes

Developer-focused features

Docker Hub:

Central registry for container images

Public and private repositories

Official images maintained by Docker

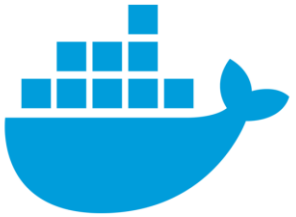
Automated builds

15+ million repositories

The Current Container Landscape

Container Runtimes:

- Docker - Most popular, developer-friendly
- Podman - Daemonless alternative (Red Hat)
- containerd - CNCF graduated project
- CRI-O - Kubernetes-native runtime



podman

cri-o

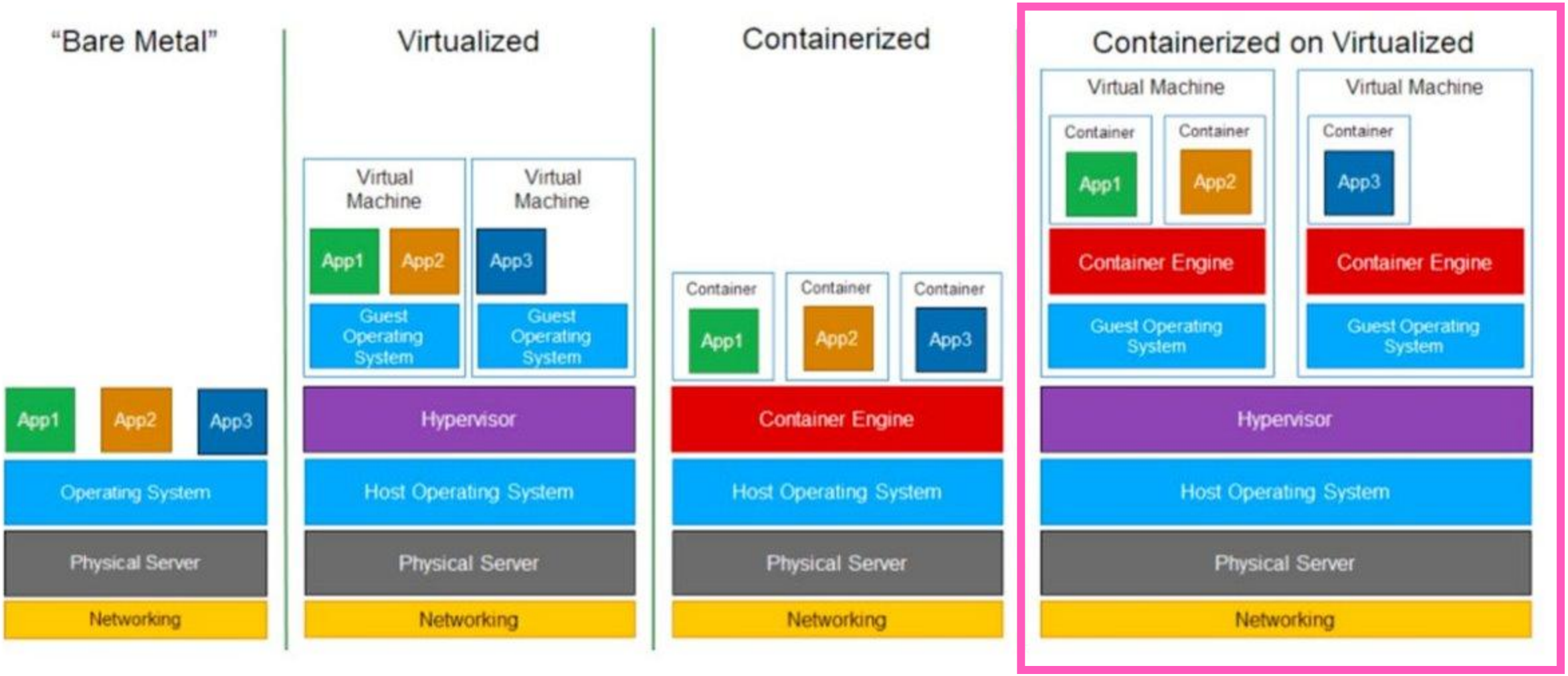
containerd

howest
university of applied sciences

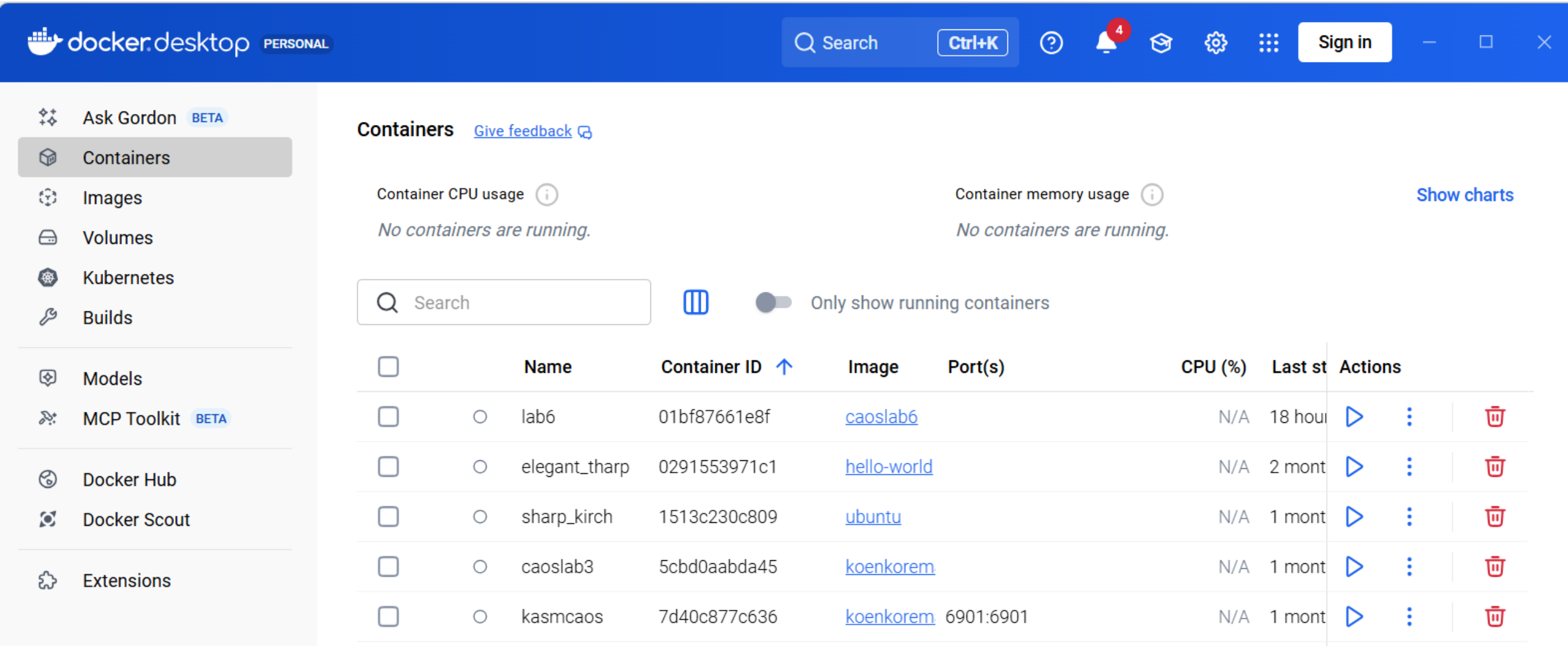
Getting your hands dirty (finally)

With Docker ! ☺

Which setup should you choose ?



So sorry, we won't be using



howest
university of applied sciences

Problem 1 : installing docker

On Debian :

apt install docker(.io) ?

Will (usually work) BUT !

Version Lag: Debian Stable prioritizes stability over recency. Docker versions in Debian repositories can be months or even years behind current releases

Limited Updates: Debian Stable only receives security updates and critical bug fixes, not feature updates.

Incomplete Package Set: The default Debian packages may not include all the modern Docker components

Solution 1 : installing docker the right way

Docker has its OWN package repository

But you need to tell Debian to trust that repository (it doesn’t do so by default)

Installing docker “the right way”

1. Install Prerequisites

sudo apt update

sudo apt install apt-transport-https ca-certificates curl gpg

2. Add Docker's GPG Key

curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker.gpg

3. Configure the Repository (all in ONE line !)

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker.gpg] https://download.docker.com/linux/debian trixie stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

Installing docker “the right way”

4. Install Docker

sudo apt update

sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

5. Check if Docker is started and enabled, if not Enable and Start Docker system service

systemctl status docker (is it running / enabled ? If not :)

sudo systemctl start docker

sudo systemctl enable docker

6. Configure User Permissions for your regular user(to allow non-root users to run docker)

sudo usermod -aG docker ${USER}

newgrp docker

7. Check your installation

docker --version

Docker commands you need to know blindly

Command	Purpose
docker run	Start a container image
docker pull	Download (pull) a container image, but do NOT create a container
docker build	Build your own image
docker exec	Execute a command inside of your running container.
docker start/stop	Start/stop a container
docker inspect	Retrieve all the metadata of a container.
docker ps	Get a list of all (running) containers
docker logs	Get the logs out of a container

Example commands

docker run hello-world

docker run -d httpd (detached mode) ; **docker run -d fedora sleep infinity (keeps alive)**

docker start/stop <container-id or name>

docker rm <container-id or name>

docker exec -ti <container-id or name> <command>

e.g. :

```
root@mis1:~# docker exec -ti my_web /bin/bash
root@38ed783b3b1e:/usr/local/apache2# cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 13 (trixie)"
NAME="Debian GNU/Linux"
VERSION_ID="13"
VERSION="13 (trixie)"
VERSION_CODENAME=trixie
DEBIAN_VERSION_FULL=13.1
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
root@38ed783b3b1e:/usr/local/apache2# whoami
root
```

shell in a container: docker exec -ti <container-id or name> /bin/bash

howest
university of applied sciences

Intermission : TAG rant

Please do it right

Container image tagging

Tag = metadata

Convenient for version control

Your choice (but please stick to common good practise)

Assigned when building a container image (e.g. **docker build –t <image_name>:<image_tag>**)

Or added later:

docker image tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]

Container image tagging

- **latest**: special tag, used as default
- Can be tekst, digit or any combination

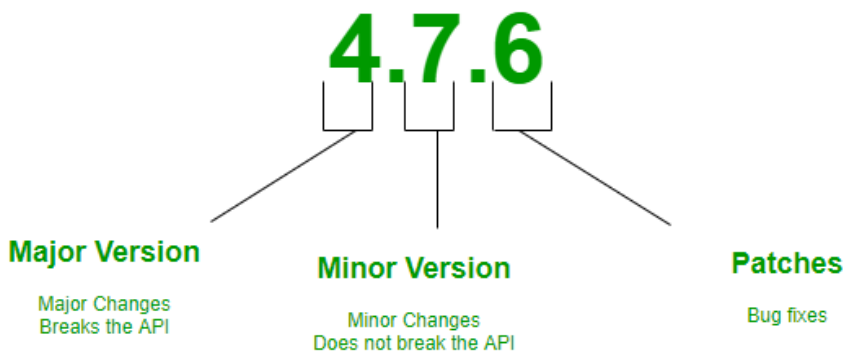
Tag	Image specifics
nginx:1.27.1	Exact version, no specification of OS.
nginx:mainline	No version tag, will update when tag is changed (not recommended)
nginx:1	Only specify major version. Every partial update will occur
nginx:1.27	Minor updates will occur.
nginx:1.27.1-trixie	Exact version pin + OS (Debian 13)
nginx:1.27.1-alpine	Exact version pin + OS (Alpine, no version)

Container image tagging – best practices

- **Always add a version pin** to your container images
- Never use latest, mainline or images without version info in production!
  - Automatic builds will update warning
  - This WILL break your pipeline. Trust me 😅
- Choose a standardized image tag acrosse services and STICK WITH IT
  - **v1.3.26** != **1.3.26**

Container image tagging – Semantic versioning

- Industry standard to tag container images
- MAJOR.MINOR.PATCH
- https://semver.org/



- Integrates great with git tagging !
  - git tag 4.7.6 # tag this commit
  - git tag -a 1.0.0 -m “Release version 1.0.0”
  - git push origin 1.0.1
  - git push --tags

howest
university of applied sciences

End of TAG rant

Thank you for your attention

Docker images

- Overview: **docker image ls**

```
osc-guy-van-eec@lnx-guy-van-eec:~$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
httpd         latest    6a4fe18d08d2   4 days ago      117MB
hello-world   latest    1b44b5a3e06a   3 months ago    10.1kB
osc-guy-van-eec@lnx-guy-van-eec:~$
```

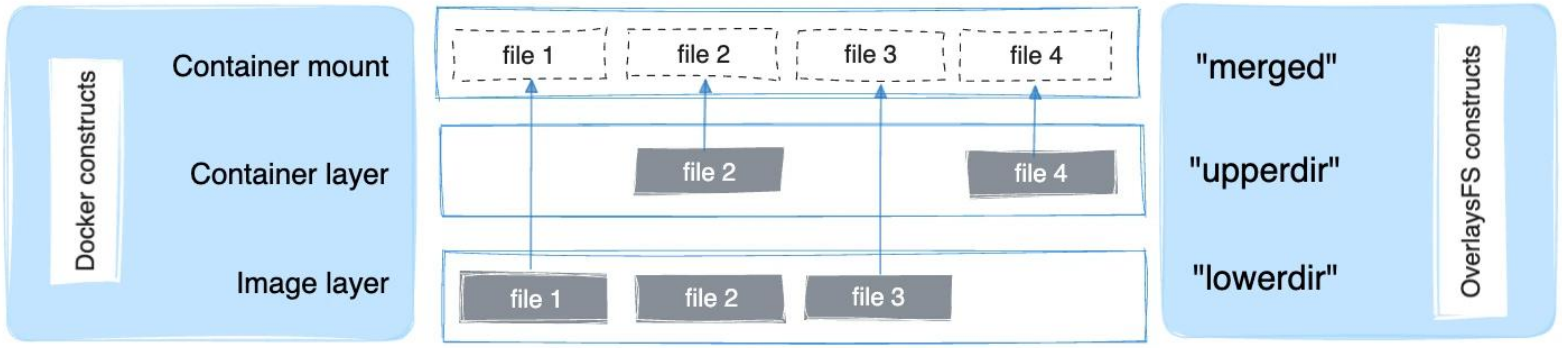
- (look at the SIZE)
- Images are stored at a *container registry*
  - hub.docker.com (Default Docker Registry)
  - quay.io (RedHat)
  - https://images.linuxcontainers.org/ (LXC containers)

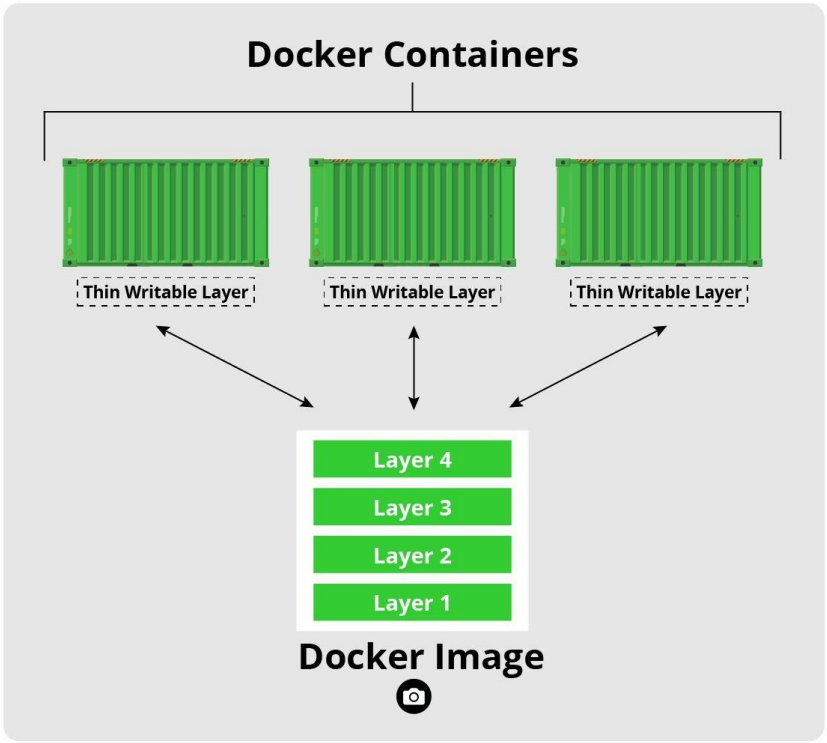
Some interesting Docker images

- Base OS containers:
  - Fedora: https://hub.docker.com/_/fedora
  - Debian: https://hub.docker.com/_/debian
  - Ubuntu: https://hub.docker.com/_/ubuntu
- Preinstalled Services:
  - Httpd (Apache): https://hub.docker.com/_/httpd
  - Nginx: https://hub.docker.com/_/nginx
  - Locust (load testing): https://hub.docker.com/r/locustio/locust
- Even full applications:
  - Wordpress: https://hub.docker.com/u/library
  - Netbox: https://hub.docker.com/r/netboxcommunity/netbox

Docker images

- Consists of layers. Only the top layer is writable !
- Overlay2 fs (/var/lib/docker/overlay2/)





- Layers that are shared between containers are only stored ONCE

Docker images

- Remove images: **docker image rm <image-naam>**

```
osc-guy-van-eec@lnx-guy-van-eec:~$ docker image rm hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:56433a6be3fda188089fb548eae3d91df3ed0d6589f7c2656121b911198df065
Deleted: sha256:1b44b5a3e06a9aae883e7bf25e45c100be0bb81a0e01b32de604f3ac44711634
Deleted: sha256:53d204b3dc5ddbc129df4ce71996b8168711e211274c785de5e0d4eb68ec3851
osc-guy-van-eec@lnx-guy-van-eec:~$
```

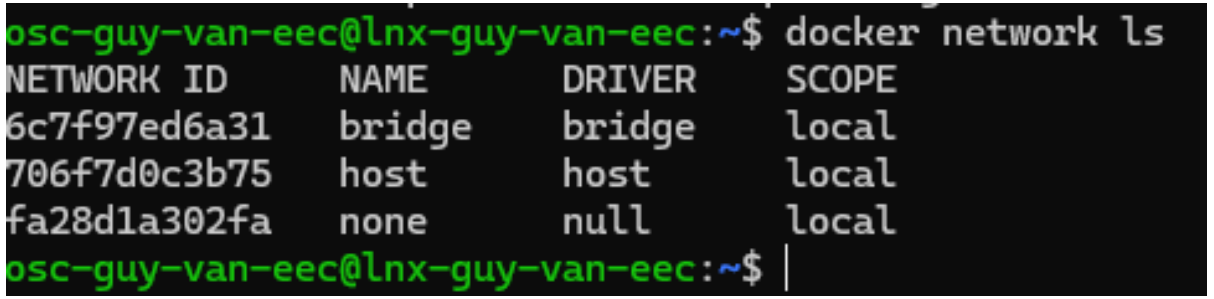
You cannot remove an image that is still being used by a container ! (without force)

```
osc-guy-van-eec@lnx-guy-van-eec:~$ docker image rm hello-world
Error response from daemon: conflict: unable to remove repository reference "hello-world" (must force) - container d77a0d37a812
 is using its referenced image 1b44b5a3e06a
osc-guy-van-eec@lnx-guy-van-eec:~$ docker image rm --force hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:56433a6be3fda188089fb548eae3d91df3ed0d6589f7c2656121b911198df065
Deleted: sha256:1b44b5a3e06a9aae883e7bf25e45c100be0bb81a0e01b32de604f3ac44711634
```

- Remove all unused images **docker image prune**
  - Only if images have not been used for a while (clean up / regain discspace)

Docker networking

- All containers have default network access
- 3 default networks available at install (**docker network ls)**:



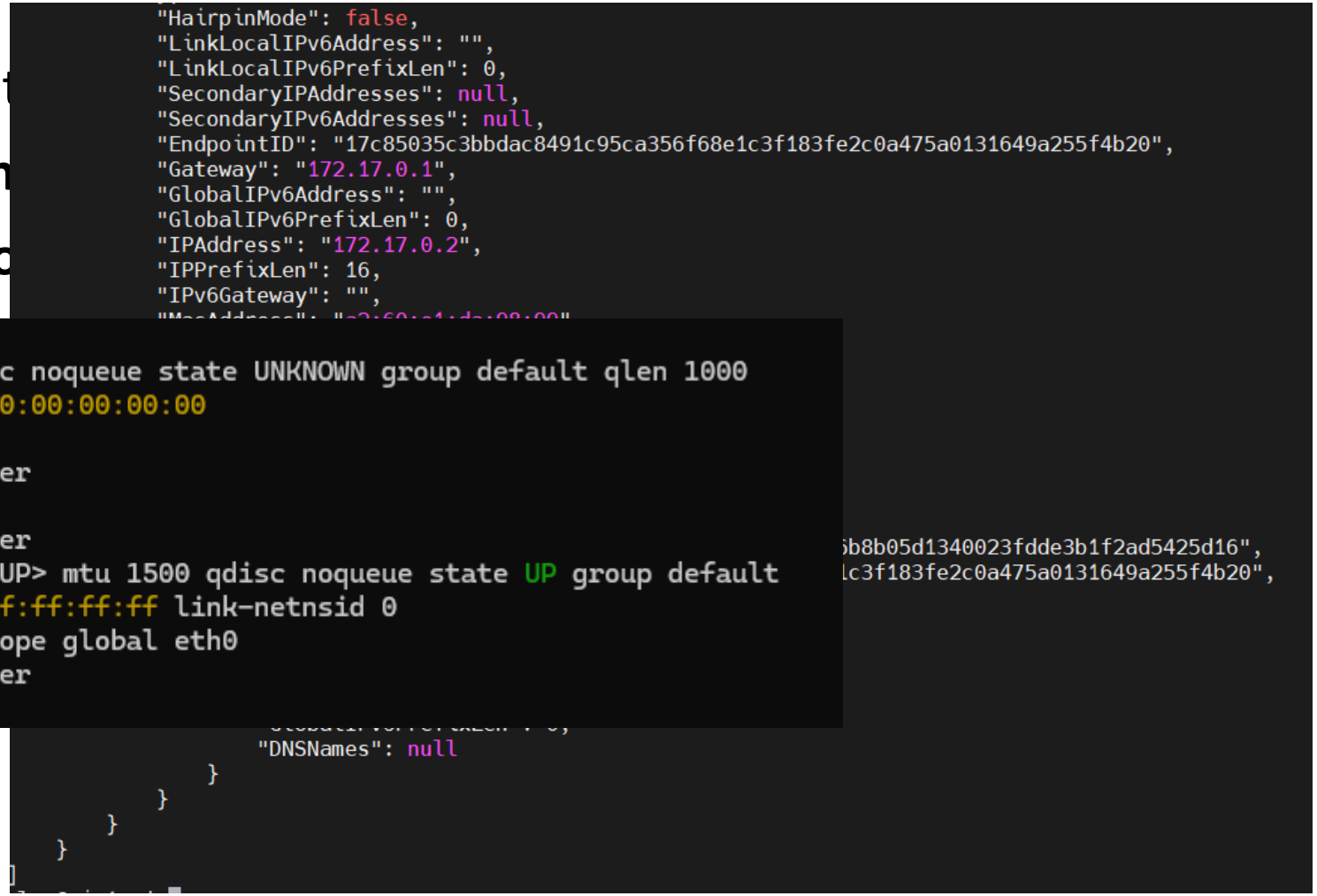
Docker networking

- Find out more info about these ne
  - **docker network inspect <netw**

```
osc-guy-van-eec@lnx-guy-van-eec:~$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "6c7f97ed6a31811c3e5cecc35a30123d26deba0bb79d4ec512da26935d02c420",
        "Created": "2025-11-08T10:07:05.10610615+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv4": true,
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```

Docker networking

- Request Netwerk info of a cont
- In container itself (**ip a** or **ifcon**
- **docker inspect <container-id c**



Docker networking

- Docker has its own NIC (docker0) the same subnet
- Used to connect to containers from the linux host

```
osc-guy-van-eec@lnx-guy-van-eec:~$ ip a s
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:3a:c7:22 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    altname enx000c293ac722
    inet 192.168.204.128/24 brd 192.168.204.255 scope global dynamic noprefixroute ens33
       valid_lft 1150sec preferred_lft 1150sec
    inet6 fe80::20c:29ff:fe3a:c722/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether ca:d9:ff:de:72:bc brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::c8d9:ffff:fede:72bc/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
```

Docker networking

- You can check your access to your containers from your linux vm

```
osc-guy-van-eec@lnx-guy-van-eec:~$ ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.272 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.107 ms
64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.119 ms
^C
--- 172.17.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2051ms
rtt min/avg/max/mdev = 0.107/0.166/0.272/0.075 ms
osc-guy-van-eec@lnx-guy-van-eec:~$
```

- E.g.: webserver on container (port 80)

```
osc-guy-van-eec@lnx-guy-van-eec:~$ curl http://172.17.0.2/
<html><body><h1>It works!</h1></body></html>
osc-guy-van-eec@lnx-guy-van-eec:~$
```

Docker networking

- Bridge netwerk is only available from the lnx host itself !
- Services should be made available outwards via port forwarding
  - **docker run -d -p 80:80 --name my_web <container image_name>**
  - E.g. Apache webserver: **docker run -d -p 8080:80 httpd**

Docker networking

- BEWARE ! You can only allocate a port on your lnx host ONCE
- So if you have multiple http containers / daemons => Allocate to other ‘external’ port:

```
osc-guy-van-eec@lnx-guy-van-eec:~$ docker run -d -p 80:80 httpd
08086d54407a2b424723a1fd456a207a5fa5b1dbd9c4b7e284ee7a9cd08dbed3
docker: Error response from daemon: failed to set up container networking: driver failed programm
ing external connectivity on endpoint distracted_wright (3a1fe1c6ed3ee1145a07f89034fa0118983da229
d479f0b97060947e404b69a2): failed to bind host port for 0.0.0.0:80:172.17.0.3:80/tcp: address alr
eady in use

Run 'docker run --help' for more information
osc-guy-van-eec@lnx-guy-van-eec:~$ docker run -d -p 8081:80 httpd
050b8e7b59d2a0bf2439c1a85369cdb7161dbb06b23021746353f48efab91e89
osc-guy-van-eec@lnx-guy-van-eec:~$
```

Docker networking

- Standard install uses iptables to control access
  - **sudo iptables –L**

```
Chain DOCKER (1 references)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             172.17.0.4           tcp dpt:http
ACCEPT     tcp  --  anywhere             172.17.0.3           tcp dpt:http
ACCEPT     tcp  --  anywhere             172.17.0.2           tcp dpt:http
DROP       all  --  anywhere             anywhere
```

Docker logs

- Starting a container in foreground will put all logs on the terminal

```
osc-guy-van-eec@lnx-guy-van-eec:~$ docker run --name my_nginx nginx:1.27.0
Unable to find image 'nginx:1.27.0' locally
1.27.0: Pulling from library/nginx
e4fff0779e6d: Pull complete
9c4e4e507b08: Pull complete
85664e7b25d6: Pull complete
e9d00ba15cc0: Pull complete
ca3474dd2ca6: Pull complete
472851929ad7: Pull complete
ed60582f2661: Pull complete
Digest: sha256:98f8ec75657d21b924fe4f69b6b9bff2f6550ea48838af479d8894a852000e40
Status: Downloaded newer image for nginx:1.27.0
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2025/11/08 10:15:19 [notice] 1#1: using the "epoll" event method
2025/11/08 10:15:19 [notice] 1#1: nginx/1.27.0
2025/11/08 10:15:19 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2025/11/08 10:15:19 [notice] 1#1: OS: Linux 6.12.48+deb13-amd64
2025/11/08 10:15:19 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1073741816:1073741816
2025/11/08 10:15:19 [notice] 1#1: start worker processes
2025/11/08 10:15:19 [notice] 1#1: start worker process 30
2025/11/08 10:15:19 [notice] 1#1: start worker process 31
```

Docker logs

- Starting a container in detached mode (-d) No longer shows logs !
- Get logs by using **docker logs <container-id of naam>**

```
osc-guy-van-eec@lnx-guy-van-eec:~$ docker logs my_nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/ngi
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/n
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-te
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-pr
/docker-entrypoint.sh: Configuration complete; ready for start up
2025/11/08 10:15:19 [notice] 1#1: using the "epoll" event method
2025/11/08 10:15:19 [notice] 1#1: nginx/1.27.0
2025/11/08 10:15:19 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14
2025/11/08 10:15:19 [notice] 1#1: OS: Linux 6.12.48+deb13-amd64
2025/11/08 10:15:19 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1073741816:
2025/11/08 10:15:19 [notice] 1#1: start worker processes
2025/11/08 10:15:19 [notice] 1#1: start worker process 30
2025/11/08 10:15:19 [notice] 1#1: start worker process 31
2025/11/08 10:16:27 [notice] 1#1: signal 2 (SIGINT) received, exiting
2025/11/08 10:16:27 [notice] 30#30: exiting
2025/11/08 10:16:27 [notice] 30#30: exit
2025/11/08 10:16:27 [notice] 31#31: exiting
2025/11/08 10:16:27 [notice] 31#31: exit
2025/11/08 10:16:27 [notice] 1#1: signal 17 (SIGCHLD) received from 30
2025/11/08 10:16:27 [notice] 1#1: worker process 30 exited with code 0
2025/11/08 10:16:27 [notice] 1#1: signal 29 (SIGIO) received
2025/11/08 10:16:27 [notice] 1#1: signal 17 (SIGCHLD) received from 31
2025/11/08 10:16:27 [notice] 1#1: worker process 31 exited with code 0
2025/11/08 10:16:27 [notice] 1#1: exit
osc-guy-van-eec@lnx-guy-van-eec:~$
```

Docker logs

- Stored in a seperate file per container
- Location: /var/lib/docker/containers/<container-id>/<container-id>-json.log

```
osc-guy-van-eec@lnx-guy-van-eec:~$ sudo cat /var/lib/docker/containers/050b8e7b59d2a0bf2439c1a85369cdb7161dbb06b23021746353f48efab91e89/050b8e7b59d2a0bf2439c1a85369cdb7161dbb06b23021746353f48efab91e89-json.log
{"log":"AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.3. Set the 'ServerName' directive globally to suppress this message\n","stream":"stderr","time":"2025-11-08T10:10:10.840198384Z"}
{"log":"AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.3. Set the 'ServerName' directive globally to suppress this message\n","stream":"stderr","time":"2025-11-08T10:10:10.846912295Z"}
{"log":"[Sat Nov 08 10:10:10.852052 2025] [mpm_event:notice] [pid 1:tid 1] AH00489: Apache/2.4.65 (Unix) configured -- resuming normal operations\n","stream":"stderr","time":"2025-11-08T10:10:10.852514562Z"}
{"log":"[Sat Nov 08 10:10:10.852161 2025] [core:notice] [pid 1:tid 1] AH00094: Command line: 'httpd -D FOREGROUND'\n","stream":"stderr","time":"2025-11-08T10:10:10.852531132Z"}
osc-guy-van-eec@lnx-guy-van-eec:~$
```

Today's LAB

1) Install docker on your lnx- VM THE RIGHT WAY
2) Docker hello-world (yay ! Hello world)
3) Observe docker container and container image behavior
4) Dabble with tags
5) Check out the docker logs
6) Explore Docker Networking