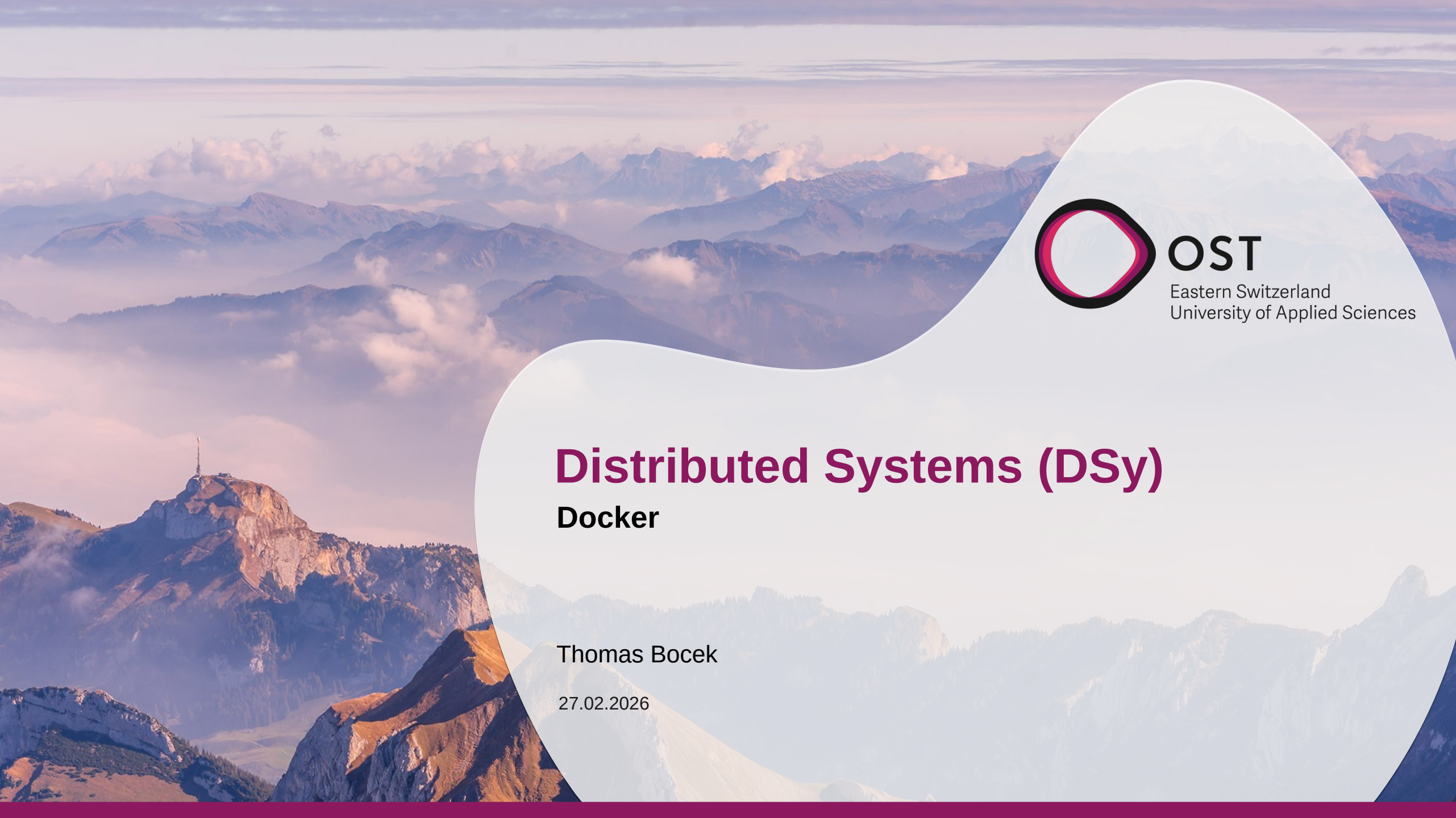


OST
Eastern Switzerland
University of Applied Sciences

# Distributed Systems (DSy)

## Docker

Thomas Bocek

27.02.2026

# Learning Goals

- Lecture 3 (Docker)
  - How does docker work (container implementation)?
    - Best practices
  - What is docker-compose, and how to run multiple services

# Introduction

- Containers – docker?
  - LXC (Linux Containers) - Lower abstraction level and direct use of Linux kernel features
  - systemd-nspawn - Part of the systemd project, minimalist container manager
  - Solaris Zones - Oracle/Sun-specific container technology
  - Linux-VServer - Kernel patch for Linux, older virtualization technology
  - OpenVz - Operating system-level virtualization, popular hosting tool
  - Singularity - Scientifically oriented container solution, HPC-friendly
  - Podman – Docker alternative / replacment [example issue]
  - Application kernel to sandbox applications
    - syd, Bubblewrap, Firejail, GVisor, and minijail
- Docker / docker-compose for software development

# Introduction - Docker

- Docker is a containerization platform
  - Packages software into containers
    - Existing images on Docker Hub
  - Containers are isolated from each other
    - Communicate over well-defined channels
  - Docker, Inc is the company behind its tooling
  - Alternatives: Podman
    - Different architecture, docker runs a daemon and you connect via CLI, podman does not [source], quadlet, podlet
    - Podman supports docker-compose , sort of
- Linux-based Isolation Mechanisms
  - Network namespaces
  - Cgroups
  - OverlayFS
  - Security
    - seccomp profiles
    - Linux capabilities
- Docker on Windows, macOS?
  - macOS, WSL2 runs a lightweight Linux inside VM → even better Isolation than with Linux, but I/O is slower

# Docker Examples

- Install docker [ubuntu, Mac, Windows]
  - `docker run hello-world`
  - Fetches the hello world example from docker hub
  - No version provided – latest
  - Docker Hub: container image repository
    - Community / official
    - Alpine
  - `docker save hello-world -o test.tar`
  - `tar xf test.tar`
  - `tar xf cdccdf50922d90e847e097347de49119be0f17c18b4a2d98da9919fa5884479d/layer.tar`
  - `./hello`

- See your installed images
  - `docker images / docker images -a`
  - `docker rmi hello-world / docker rmi fce289e99eb9`
  - `docker ps -a`
  - `docker rm 913edc5c90c4`
- GUI: e.g., Docker Desktop, Podman Desktop



OST

# Details

- FS Virtualization
  - OverlayFS: union filesystem, “combines multiple different underlying mount points into one”
- Dockerfile:
  - docker build . –t test
  - docker run test
  - docker save test:latest > test.tar

```
Dockerfile:

FROM alpine
ADD hello.sh .
ENTRYPOINT ["sh", "hello.sh"]

hello.sh:

#!/bin/sh
echo "Hallo"
```

- 2+ Layers
  - Alpine, with BusyBox, 1MB, libc (musl), crypto, ssl, etc.
  - hello.sh
- Add a new layer
  - If input does not change, docker layer is kept - cached



OST

# OverlayFS

- Example
  - The lower directory can be read-only or could be an overlay itself
  - The upper directory is normally writable
  - The workdir is used to prepare files as they are switched between the layers.

```
cd /tmp
mkdir lower upper workdir overlay

sudo mount -t overlay -o \
lowerdir=/tmp/lower,\
upperdir=/tmp/upper,\
workdir=/tmp/workdir \
none /tmp/overlay
```

- Read only
- How to remove data in read-only lowerdir
  - Mark as deleted in upperdir

```
cd /tmp
mkdir lower upper workdir overlay

sudo mount -t overlay -o
lowerdir=/tmp/lower1:/tmp/lower2 /tmp/overlay
```

```
cd /tmp
mkdir lower upper workdir overlay

sudo mount -t overlay -o \
lowerdir=/tmp/lower1:/tmp/lower2,\
upperdir=/tmp/upper,\
workdir=/tmp/workdir \
none /tmp/overlay
```



OST

# Cgroups

- control groups: limits, isolates, prioritization of CPU, memory, disk I/O, network

```
ls /sys/fs/cgroup

sudo apt install cgroup-tools / yay -S libcgroup

cgcreate -g cpu:red
cgcreate -g cpu:blue

echo -n "20" > /sys/fs/cgroup/blue/cpu.weight
echo -n "80" > /sys/fs/cgroup/red/cpu.weight

cgexec -g cpu:blue bash
cgexec -g cpu:red bash

sha256sum /dev/urandom #does not work?
taskset -c 0 sha256sum /dev/urandom
```

- Install tools
- Create two groups
  - Assign 20% of CPU and 80% of CPU
- Execute bash → test CPU
- Resource control with docker

```
docker run \
--name=low_prio \
--cpuset-cpus=0 \
--cpu-shares=20 \
alpine sha256sum /dev/urandom

docker run \
--name=high_prio \
--cpuset-cpus=0 \
--cpu-shares=80 \
alpine sha256sum /dev/urandom
```



OST

# Separate Networks

- Linux Network Namespaces
  - provide isolation of the system resources associated with networking [source]

```
ip netns add testnet
ip netns list
```

  - Create virtual ethernet connection

```
ip link add veth0 type veth peer name veth1 netns testnet
ip link list
ip netns exec testnet <cmd>
```

  - Configure network

```
ip addr add 10.1.1.1/24 dev veth0
ip netns exec testnet ip addr add 10.1.1.2/24 dev veth1
ip link set dev veth0 up
ip netns exec testnet ip link set dev veth1 up
```

- Run server

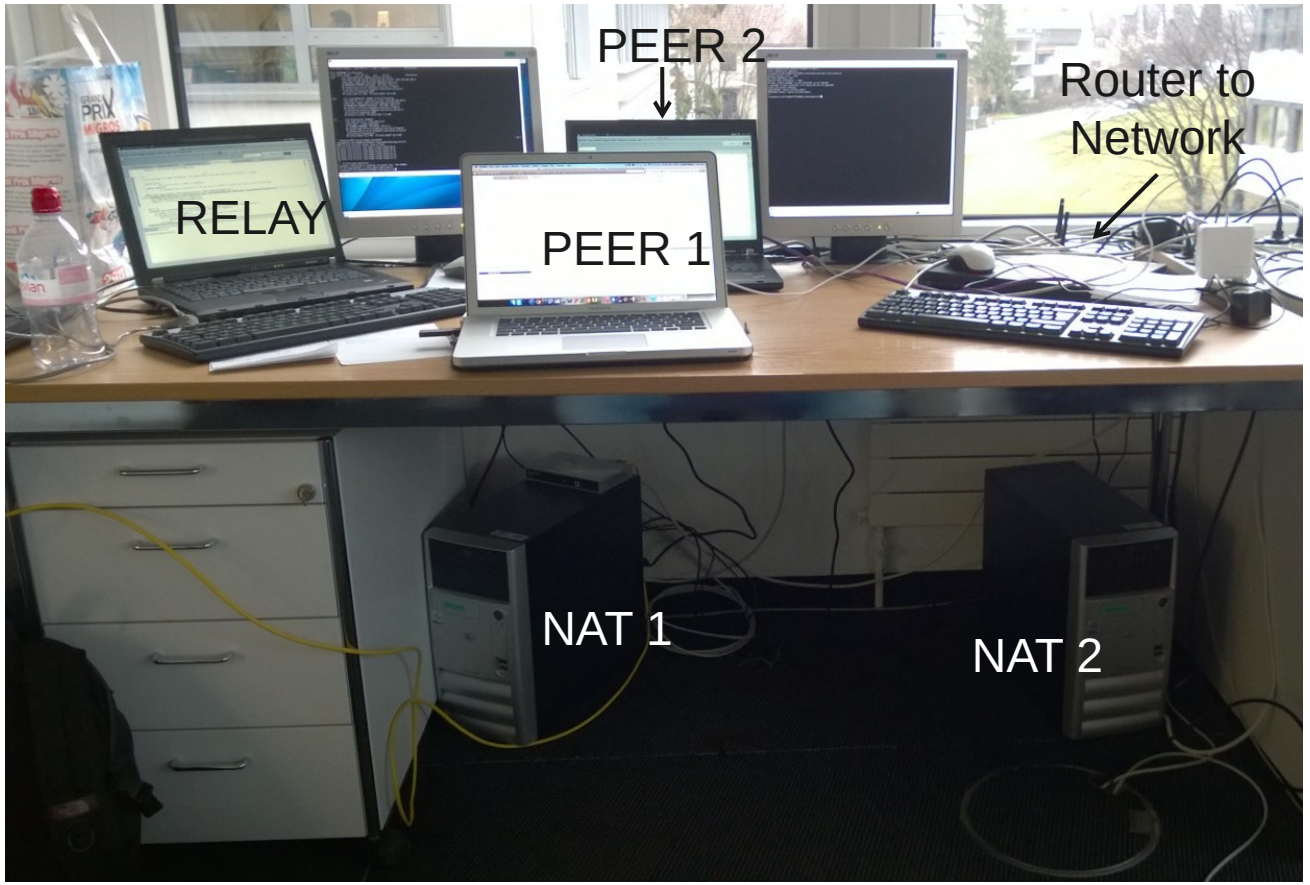
```
ip netns exec testnet nc -l -p 8000
```

- Server can be contacted
- How to connect to outside?
  - E.g. layer 3

```
echo 1 > /proc/sys/net/ipv4/ip_forward
ip netns exec testnet ip route add default via 10.1.1.1 dev veth1
iptables -t nat -A POSTROUTING -s 10.1.1.0/24 -o enp6s0 -j MASQUERADE
iptables -A FORWARD -j ACCEPT #open up wide…
```

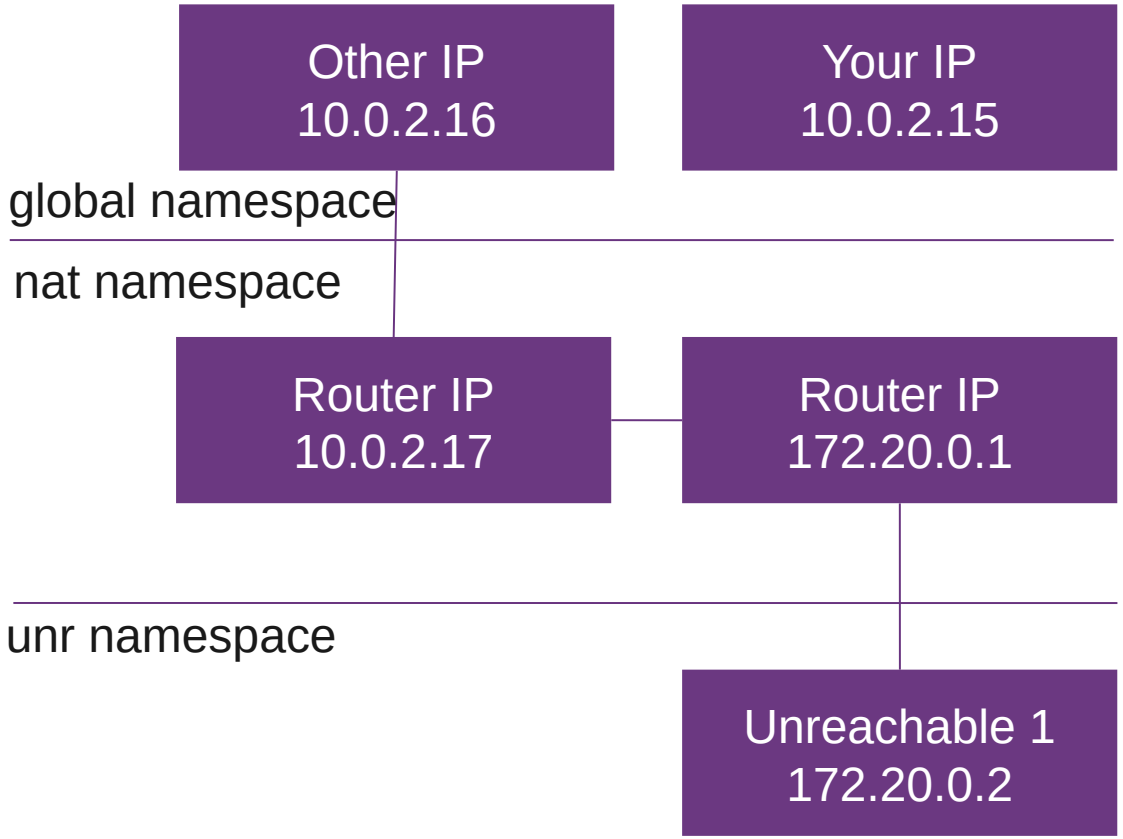
# Connectivity, Security, and Robustness

- NAT (Network Address Translation): Multiple devices share one public IP
  - Devices are not directly reachable
- Hole Punching: Two peers behind NAT send coordinated packets to create NAT mappings, enabling a direct connection
  - P2P / Hole Punching Development (in the old days)
  - Currently: network namespaces (since Linux 2.6.24)



# Make your own Testbed for P2P System

- veth - Virtual Ethernet Device
  - Tunnels between network namespaces
  - `ip netns add unr / ip netns list`
  - `ip link add nat_lan type veth peer name nat_wan`
  - `ip link set nat_lan netns unr`
  - `ip address add 10.0.2.16/24 dev nat_wan`
  - `ip link set nat_wan up`
    - ifconfig / ping
  - `ip netns exec unr ip address add 172.20.0.1/24 dev nat_lan`
  - `ip netns exec unr ip link set nat_lan up`

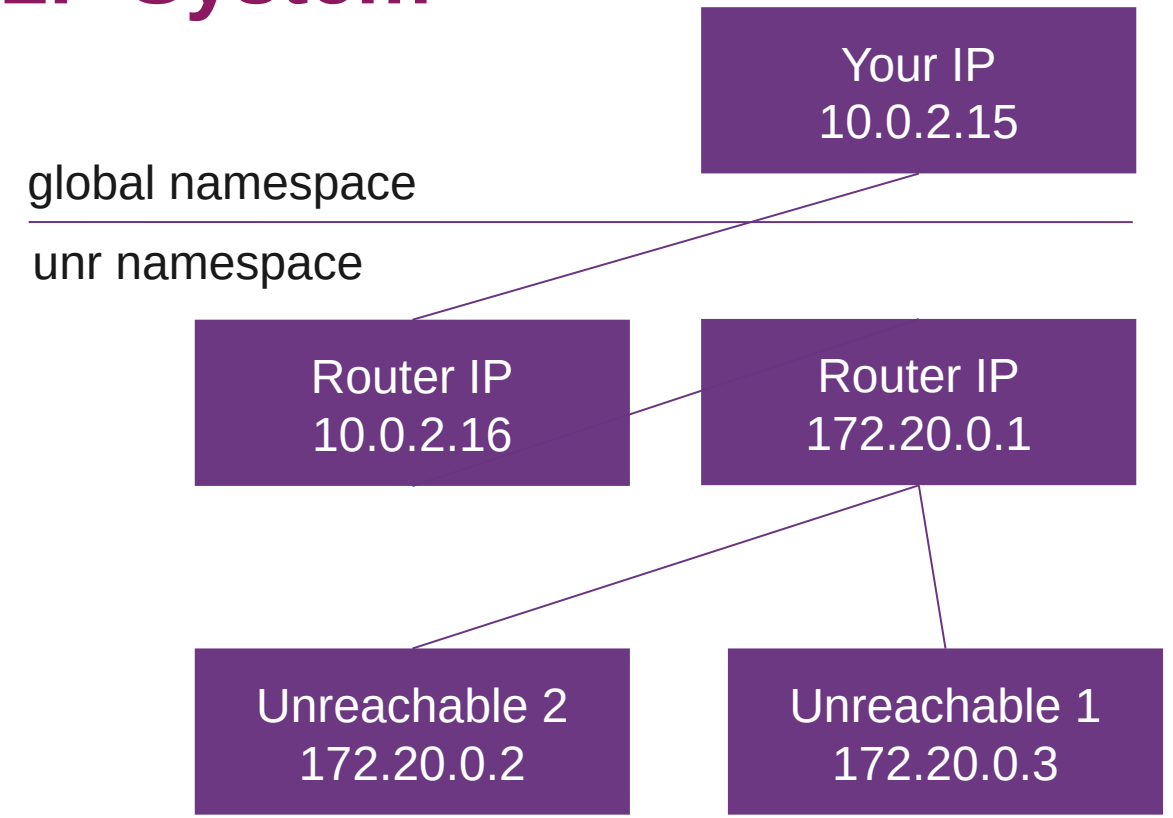





OST

# Make your own Testbed for P2P System

- Setup 2 unreachable peers
  - `ip netns exec unr ip link add unr1 type dummy`
  - `ip netns exec unr ip address add 172.20.0.2/24 dev unr1`
  - `ip netns exec unr ip link set unr1 up`
    - `ip netns exec unr ifconfig`
  - `ip netns exec unr ip link set lo up`
  - `ip netns exec unr route add default gw 172.20.0.1`
  - `ip route add 172.20.0.1 dev nat_wan`






OST

# Docker Compose

- Dockerfile
  - Build your binary with a Dockerfile - example
  - Create our own image with a Dockerfile
    - keep images small
  - Multi-stage builds, copy required files
  - Squash images: `COPY --from=initial / /`
  - Remove cache, as docker is caching aggressively

```
#Dockerfile
FROM golang:alpine AS builder
WORKDIR /build
COPY server.go .
RUN go build server.go

FROM alpine
WORKDIR /app
COPY --from=builder /build/server .
ENTRYPOINT ["./server"]
```

- Docker Compose to deploy multiple containers
  - E.g, load balancer, services, DB
  - Configure your services
  - Lightweight orchestration
    - Start service depending on others (e.g., PostgreSQL)

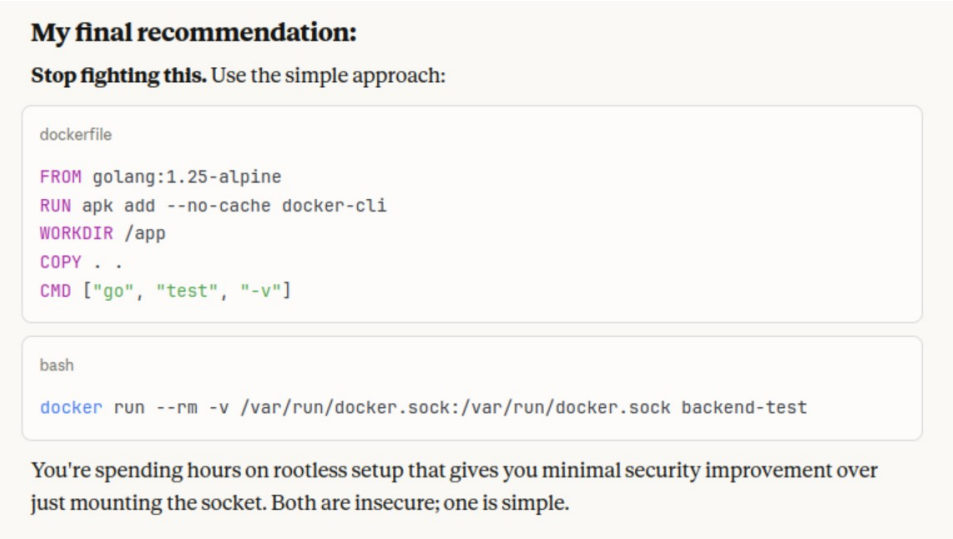
```
#docker-compose.yml
services:
 server1:
  build: .
 client:
  image: alpine
  entrypoint: >
   sh -c "sleep 3 && echo hallo | nc server1 8081"
```



OST

# Docker Security

- Best practices [link]
  - Keep images small, up to date / attack surfac
    - alpine / distroless
    - My preference alpine (convenience)
  - Check your image for vulnerabilities, snyk, trivy
  - Do not expose the Docker daemon socket (even to the containers) – keep fighting!



**My final recommendation:**

**Stop fighting this.** Use the simple approach:

dockerfile

```
FROM golang:1.25-alpine
RUN apk add --no-cache docker-cli
WORKDIR /app
COPY . .
CMD ["go", "test", "-v"]
```

bash

```
docker run --rm -v /var/run/docker.sock:/var/run/docker.sock backend-test
```

You're spending hours on rootless setup that gives you minimal security improvement over just mounting the socket. Both are insecure; one is simple.

- Set a user – do not run as root
  - Needs a bit more configuration, but good advice
- Limit capabilities (Grant only specific capabilities, needed by a container)
  - Seems overkill
- Limit resources (memory, CPU, file descriptors, processes, restarts)
  - Good for production → docker does not have a hard memory limit. Docker kills process that goes over limit, no pressure for GC if not docker aware
- Set filesystem and volumes to read-only



OST

# How to Debug? / Pitfalls

- What is your base image
  - Ubuntu-based, Debian-based, Alpine-based, distroless
  - GNU libc (e.g. distroless) vs. musl (e.g., alpine)
    - Glibc: wide adoption, many distros use it, larger binary
    - Musl: less used, smaller binary
    - Benchmarks
  - Need glibc? → busybox docker, or gcompat
  - → building outside docker, then copy binary to docker may fail
- Alpine has many tool
  - Busybox: nc, ping, sha256sum, wget, netstat
  - Reach other container: ping cointainername
  - Service bound to localhost? Cannot run outside docker, use 0.0.0.0
- docker system prune -a (attention)
- Check logfiles!
- Peformance: mulit-stage, .dockerignore
- Docker: aggressive caching – wget is chached! E.g., use version numbers

```
#Dockerfile
FROM golang:alpine
RUN wget https://app/package-latest.tar.gz
```