OST
Eastern Switzerland
University of Applied Sciences

Distributed Systems & Blockchain (DS1)

Virtualization

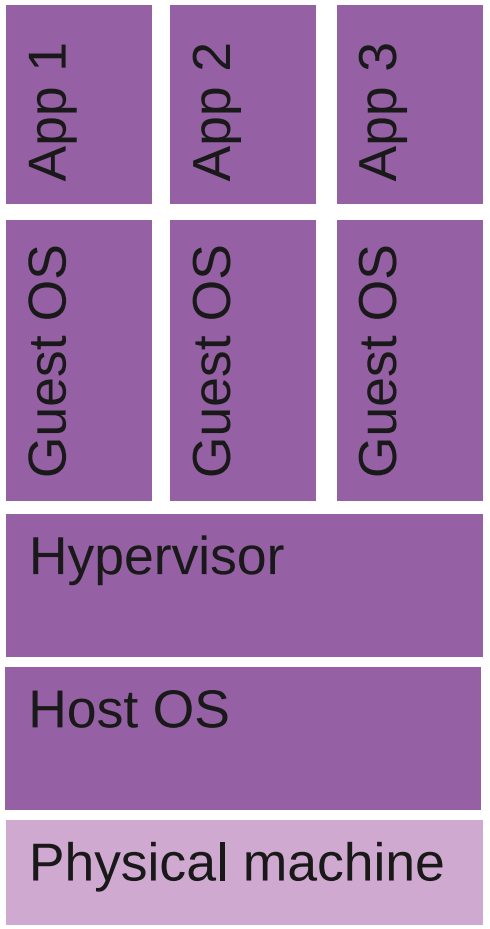
Thomas Bocek

7 March 2021

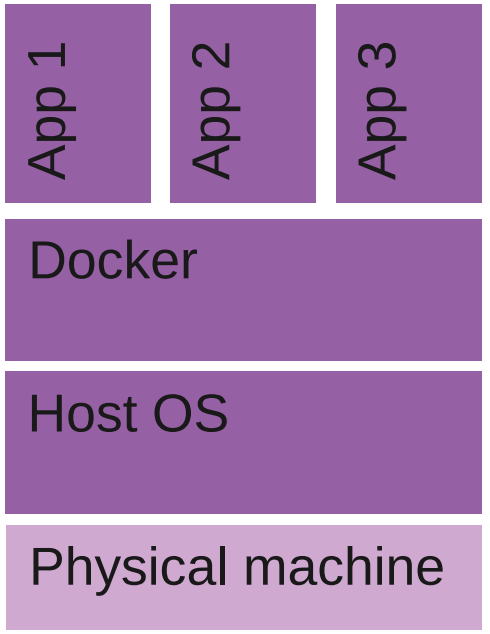
Virtualization

- “creation of a virtual machine that acts like a real computer with an operating system” [source]
  - Host machine: machine where the virtualization software runs
  - Guest machine: virtual machine
- Hypervisor runs virtual machines
  - Type 1: bare-metal – e.g., Xen
    - “We built Amazon EC2 using a virtual machine monitor by the name of Xen” [source]
  - Type 2: hosted – e.g., VirtualBox
- Run unmodified OS with Intel VT-x and AMD-V, or paravirtualized if not present
  - E.g., VM should not access memory directly
- Needs to be the same architecture
  - Otherwise use emulation, e.g., QEMU
    - Ubuntu on a RISC-V processor
    - Qemu, opensbi, u-boot
  - Console emulators: Snes9x, Mupen64Plus, Switch
- Virtual desktop infrastructure (VDI)
  - Interact with a virtual machine over a network
- Containers
  - Isolated user-space instances
  - Share the OS

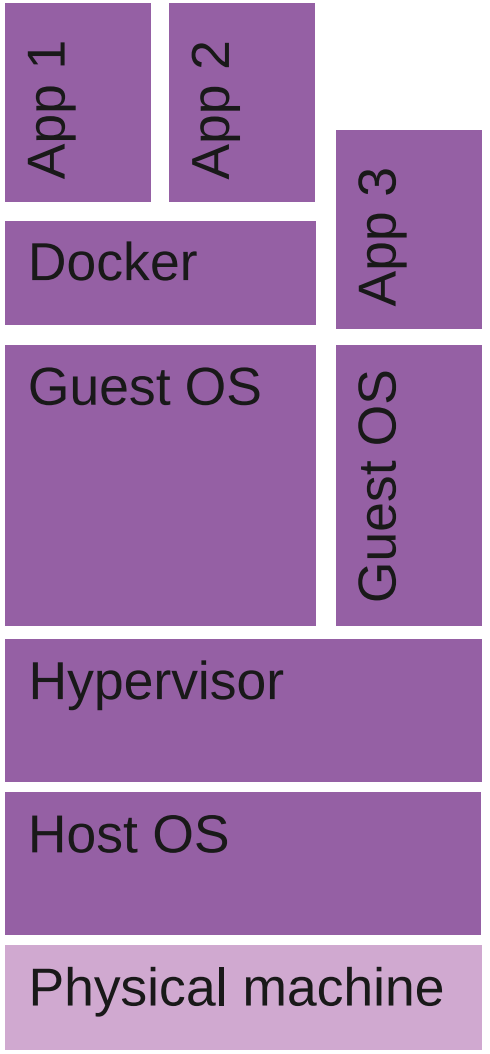
Introduction



- Virtual machines



- Container



- Both

OST

Introduction

- Docker is a containerization platform
- Docker as a software delivery framework
  - Packages software into containers
    - Existing images on Docker Hub
  - Provides OS-level virtualization
  - 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]
    - Podman supports docker-compose
- OS virtualization (Containers, e.g., Docker) vs virtual machine (VirtualBox)
  - Containers Are More Agile than VMs
    - “works on my machine”
  - Containers Enable Hybrid and Multi-Cloud Adoption
  - Integrate Containers with Your Existing IT Processes
  - Containers Save on VM Licensing
    - Kernel-based Virtual Machine (KVM)

OST

Comparison

Container

+ Reduced IT management resources

+ Reduced size of snapshots 2MB vs 45MB

+ Quicker spinning up apps

+ / - Available memory is shared

+ / - Process-based isolation (share same kernel)

Use case: complex application setup, with container less complex configuration

Providers: ECS, Kubernetes Engine, Docker on Azure (or Kubernetes)

Virtual Machine

+ App can access all OS resources

+ Live migrations

+ / - Pre allocates memory

+ / - Full isolation

Use case: better hardware utilization / resource sharing

EC2, Virtual Machines, Compute Engine, Droplets

Market shares, market hares, other views

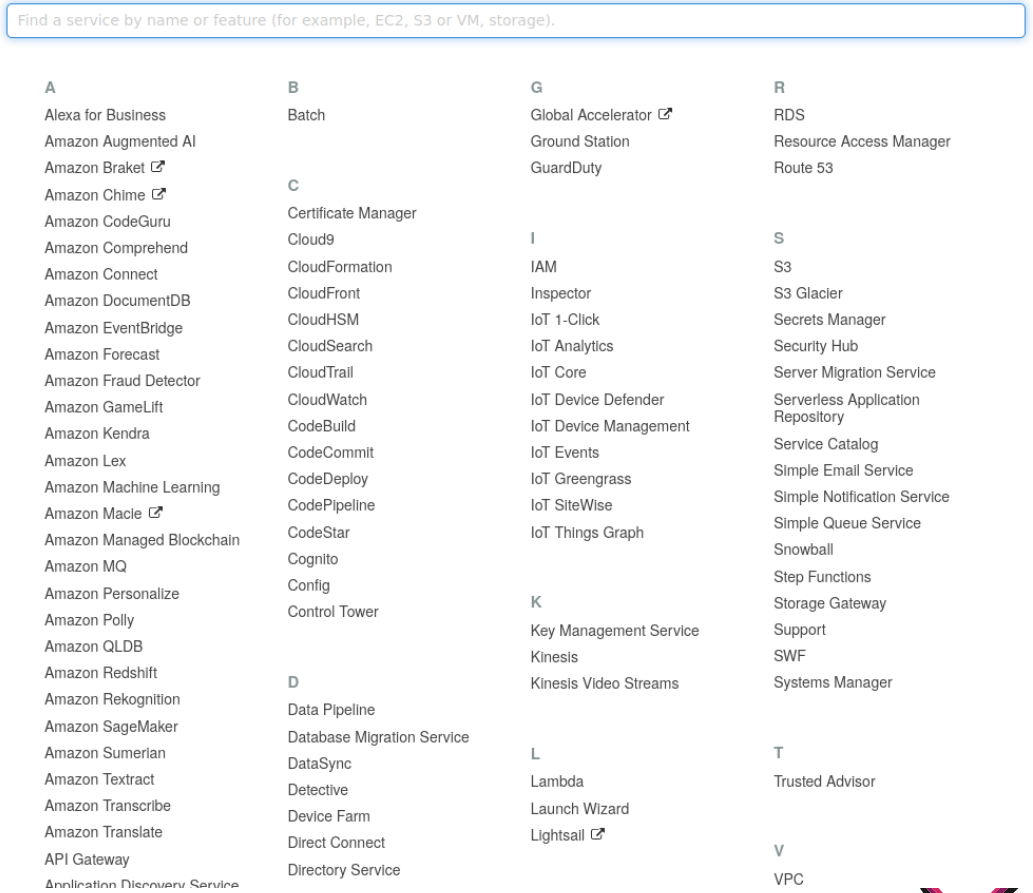
Prices / VM on e.g., AWS

Virtual Machines

- On-Demand
  - Machine
  - Data transfer
  - IP address
- Spot instances (discount when not needed)
- Reserved Instances
- Comparison, comparison, comparison
  - Not easy to compare
  - Optimize for cost → provider changes cost structure, you need to adapt again for optimizing

Example EC2

- Instead VirtualBox as in the exercises, spin up VM on EC2



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., DockStation, other

OST

Details

- Bocker: Docker implemented in around 100 lines of bash
  - Requirements: btrfs-progs, curl, iproute2, iptables, libcgroup-tools, util-linux, coreutils
- 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 .
CMD ["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

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
```

Cgroups

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

```
ls /sys/fs/cgroup

sudo apt install cgroup-tools

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

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

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 sah256sum /dev/urandom
```

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 netns exec testnet ip link set dev veth1 up
```

  - Run server

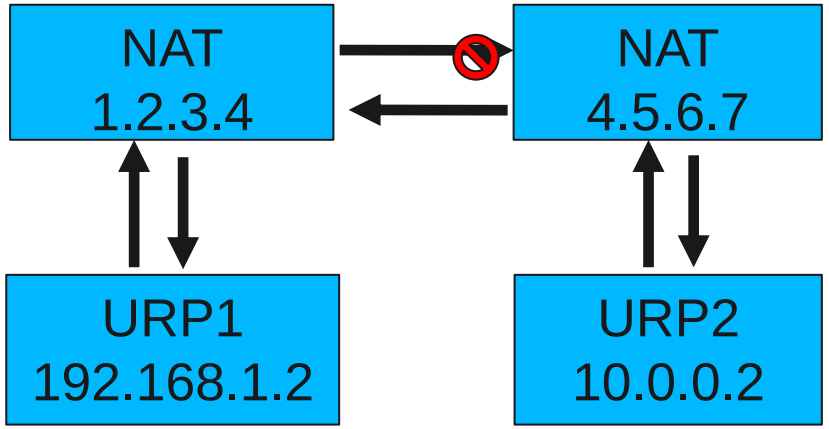
```
ip netns exec blue nc -l 8000
```

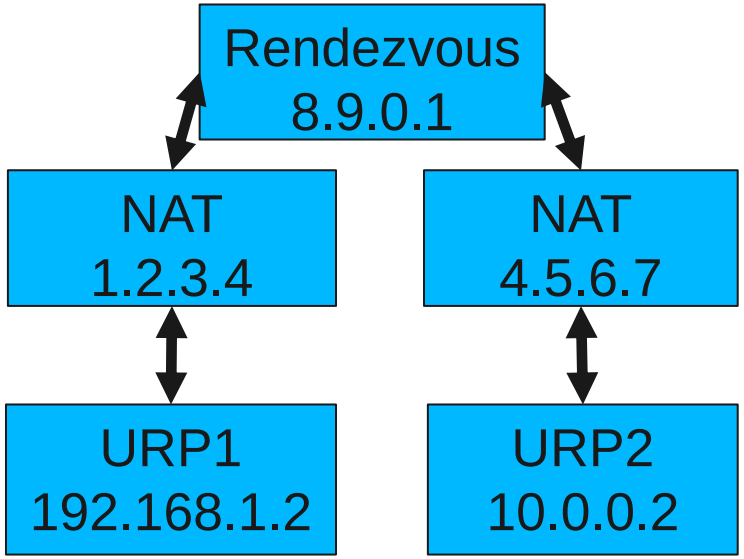
  - Server can be contacted
  - How to connect to outside?
    - E.g. layer 3

```
iptables -t nat -A POSTROUTING -s 10.1.1.0/24 -o enp9s0 -j MASQUERADE
iptables -A FORWARD -j ACCEPT #open up wide…
```

Connectivity, Security, and Robustness

- Hole punching
  - URP1 got 4.5.6.7:5000, URP2 got 1.2.3.4:4000
  - Unreachable peer 1 request to NAT 4.5.6.7, will fail – no mapping, however, unreachable peer 1 creates mapping with that request
  - Unreachable peer 2 sends request to unreachable peer 1 (1.2.3.4:4000)
    success!





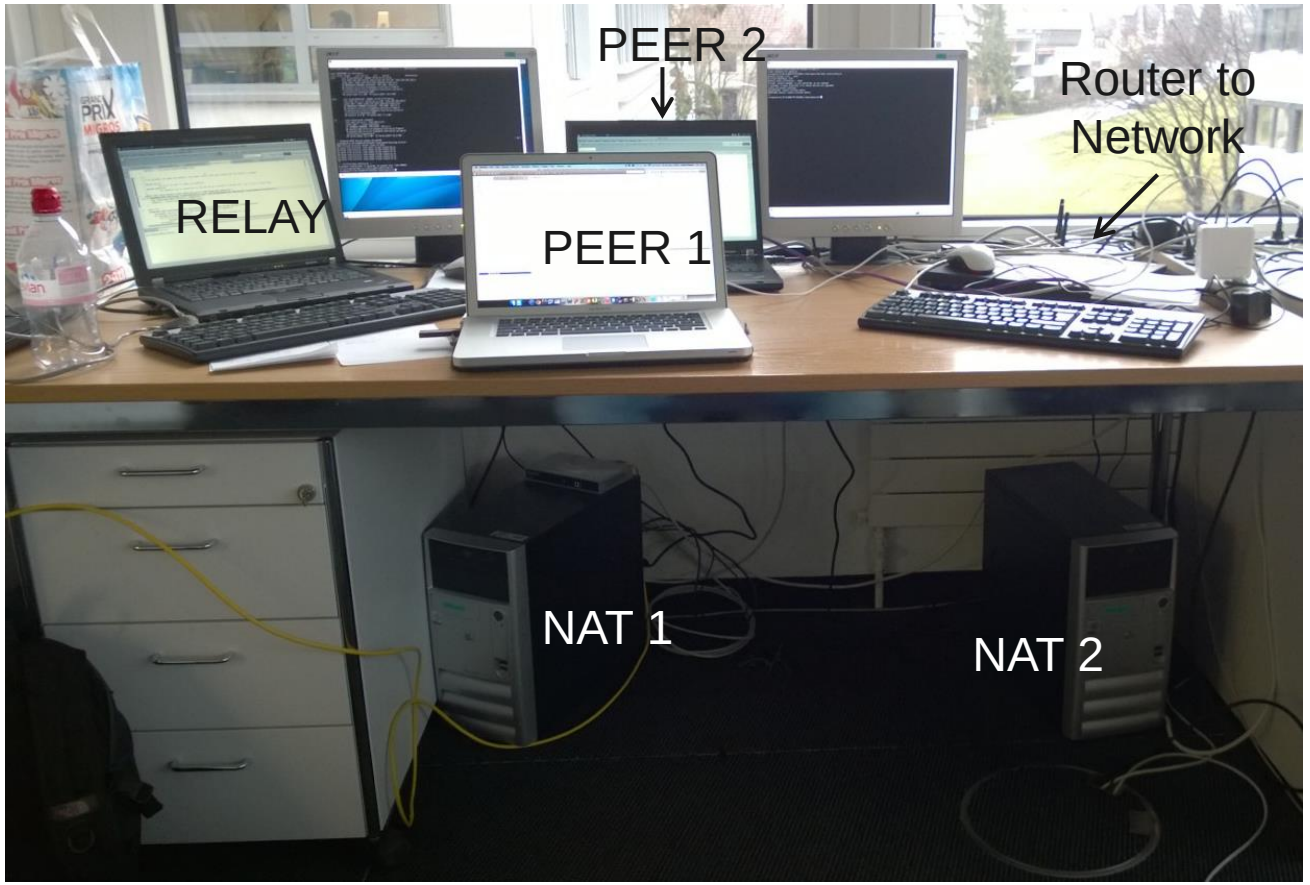
Mapping for NAT 1.2.3.4 (Unreachable peer 1)			
192.168.1.2:4000	4.5.6.7:5000	4.5.6.7:5000	1.2.3.4:4000

Mapping for NAT 4.5.6.7 (Unreachable peer 2)			
10.0.0.2:5000	1.2.3.4:4000	1.2.3.4:4000	4.5.6.7:5000

OST

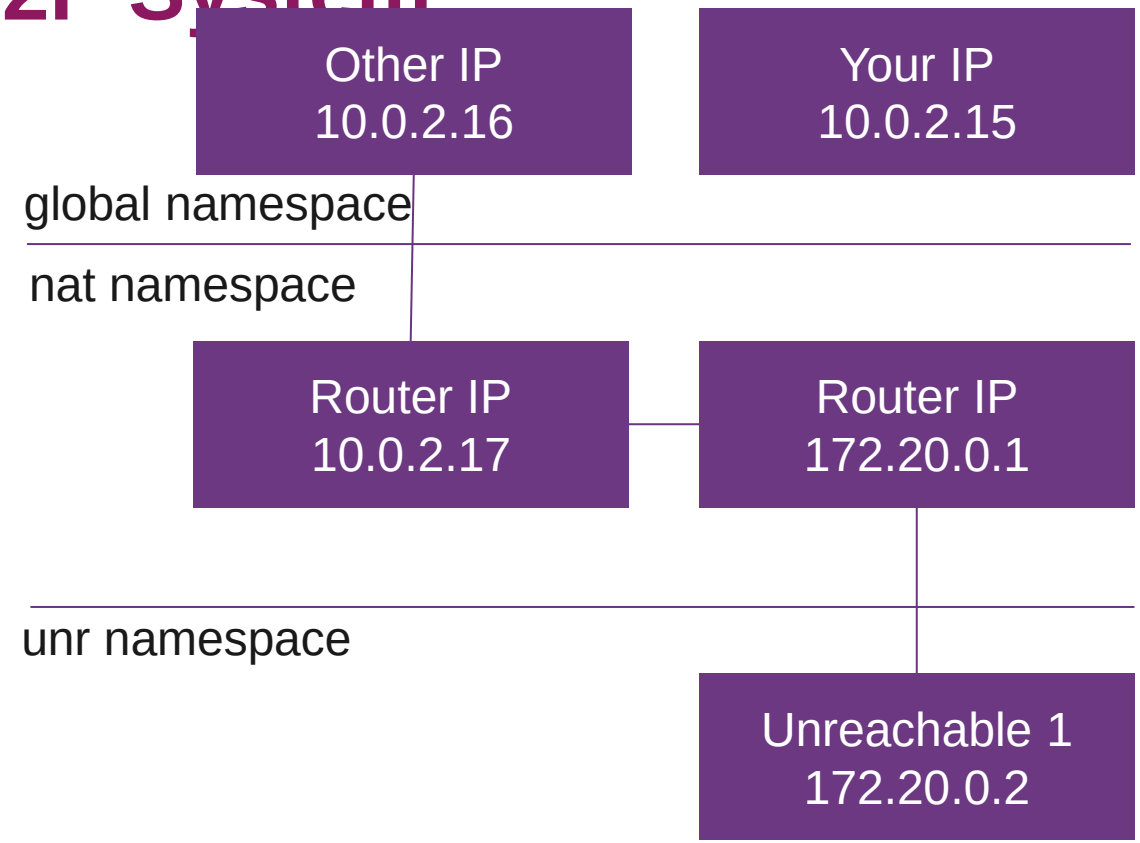
Connectivity, Security, and Robustness

- Hole Punching Development (in the old days)
- Currently: network namespaces (since Linux 2.6.24)



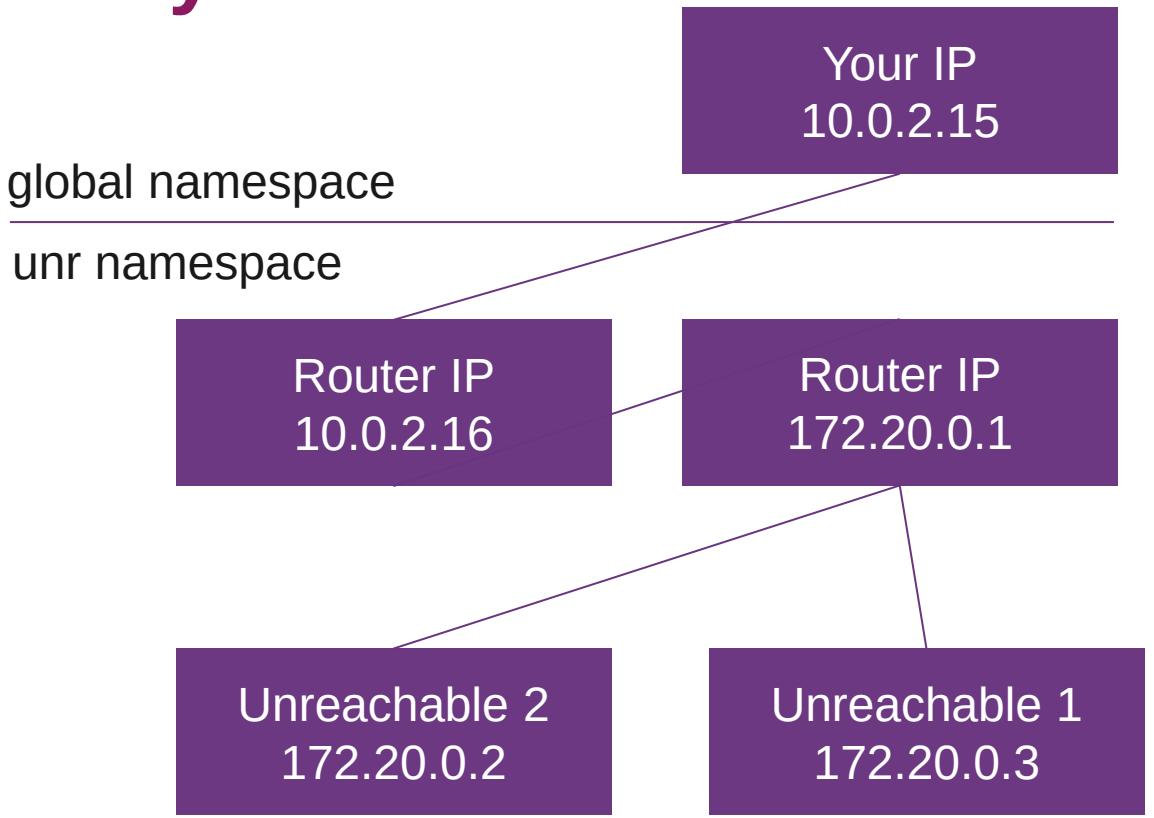
Make your own Testbed for P2P System

- 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



Make your own Testbed for P2P System

- Seutp 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



Docker Compose

- Docker Compose to deploy multiple containers
  - E.g, load balancer, services, DB
  - Configure your services

```
#docker-compose.yml
version: '3'
services:
  service:
    build: .
    ports:
      - "8080:8081"
  db:
    image: "postgres:13-alpine"
    ports:
      - “5432:5432"
    volumes:
      - ./.db:/var/lib/postgresql/data
    environment: …
```

- Dockerfile
  - Build your binary with a Dockerfile
  - Up to now, we used existing images, now we create our own image
  - Multi-stage builds

```
#Dockerfile
FROM golang:alpine AS builder
WORKDIR /build
COPY server-stateful.go .
RUN apk --no-cache add git
RUN go get -d -v
RUN go build server-stateful.go

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /build/server-stateful .
CMD ["./server-stateful", "-p", "8081"]
```


OST