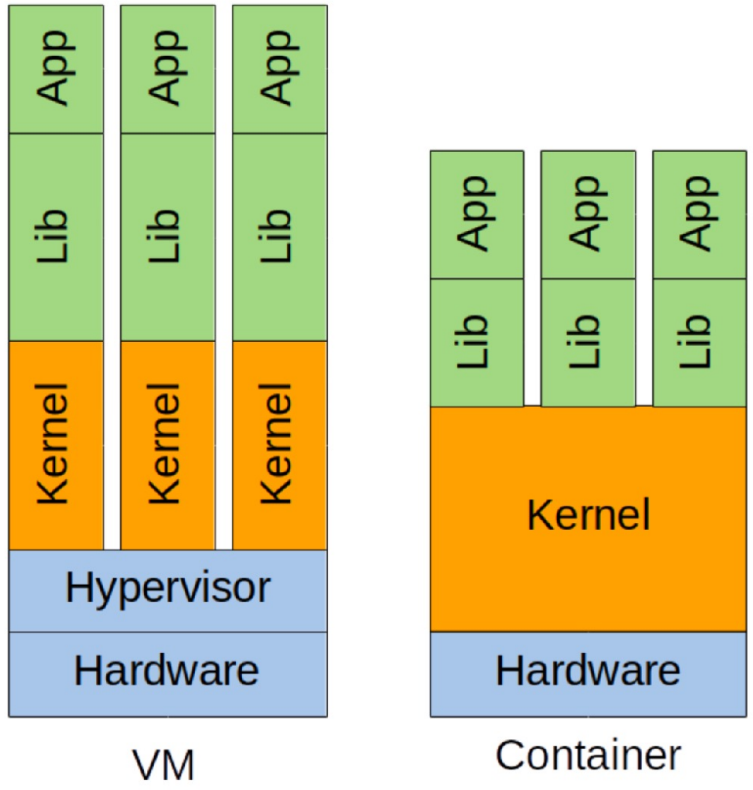
Light Virtualization

VMs vs Containers



Source: *Tom Goethals, Merlijn Sebrechts, Ankita Atrey, Bruno Volckaert, Filip De Turck: Unikernels vs Containers: An In-Depth Benchmarking Study in the Context of Microservice Applications. SC^2 2018: 1-8*

VMs vs Containers

- Containers
  - Lightweight
  - Almost no loss of perf
- But
  - Inherently less secure (shared kernel)
  - Harder to migrate (but backup is ok)

Containers key components (Linux)

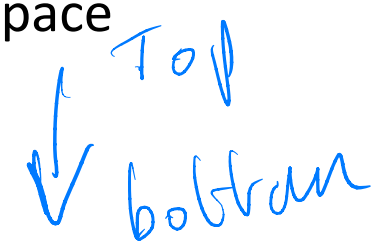
- Namespaces : logical separation of processes
- Cgroups : physical separation of processes
  - Assign ressources (CPU, Memory, Network) to processes
- Features of the Linux Kernel
- Container solutions (Docker, LXC, OpenVZ….) are built on these features

Namespaces

Overview

Namespaces

- Introduced in Linux Kernel 2.4.19 in 2002
- Limits what a process sees/uses in terms of other processes, net. Interfaces, mounted volumes…
- 8 kinds of namespace
- A process belongs to a set of namespaces
  - Default namespace
- Namespaces are organized in a hierarchy
- A process can see/use resources
  - In the associated namespace
  - In the descendant ones

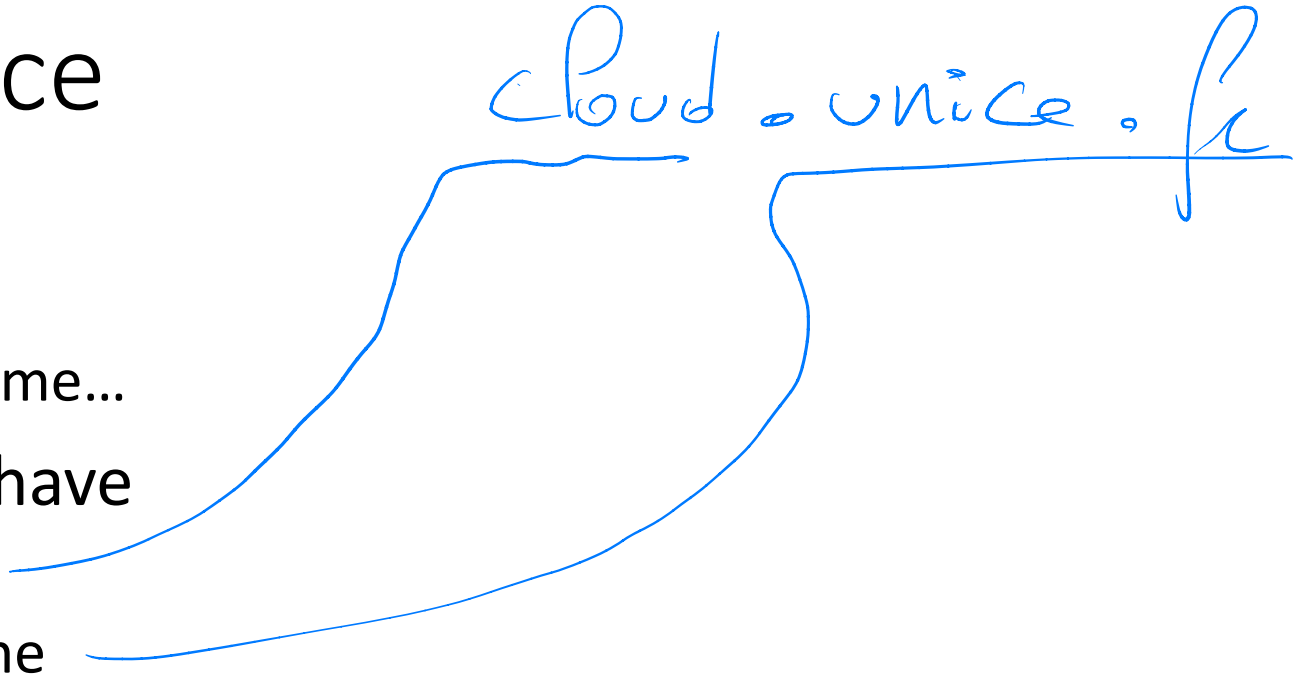


Net Namespace

- A process sees only the networking stack of the NET namespace it belongs to
  - Its own interfaces (loopback)
  - Its own routing tables
  - Its own iptables (Firewall/NAT) rules
  - Its own sockets

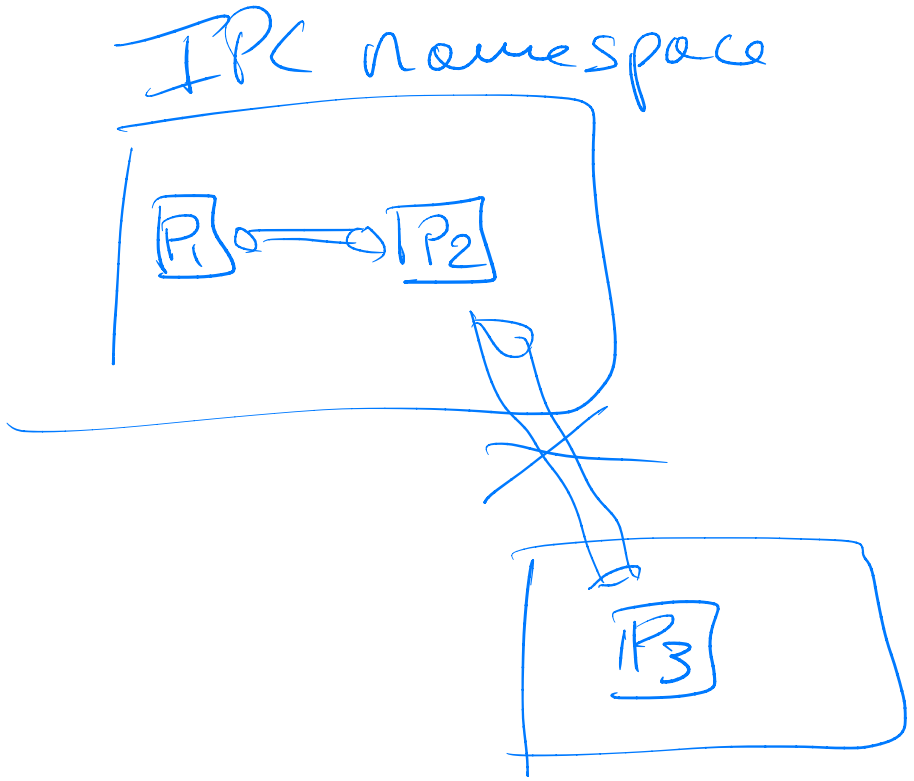
UTS Namespace

- Unix Time Sharing
  - But not related to time…
- Allows a process to have
  - Specific hostname
  - Specific domainname



IPC Namespace

- Inter Process Communication
- Enable a set of processes to have their
  - Semaphores
  - Messages queues
  - Shared Memory
- Allows 2 processes in different IPC namespaces to
  - Open Shared Memory with same parameters (like name)
  - Not share anything



User Namespace

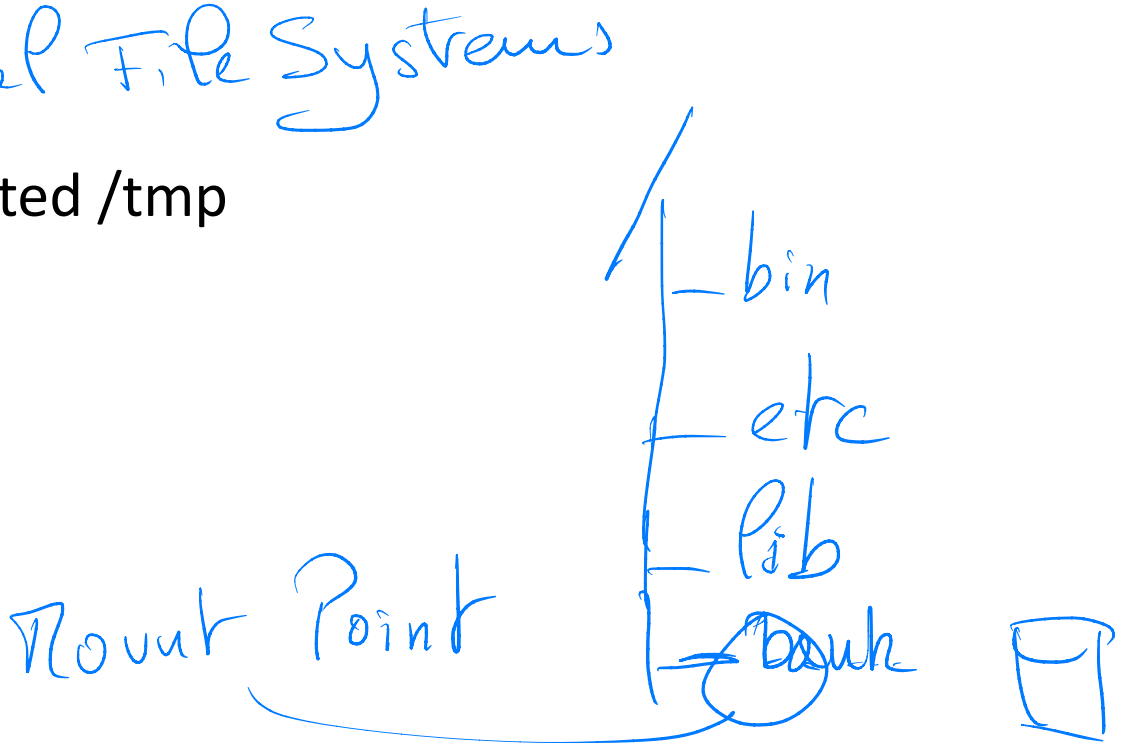
user / group

- App needs specific UID
- UID not available on host

- Specific UID/GID for each user namespace
- Mapping of IDs of user namespace to IDs of Host
- Example :
  - In one namespace, UIDs from 0 to 9999 can be mapped to 10 000 -> 19 999 on host
  - In another namespace, UIDs from 0 to 9999 can be mapped to 20 000->29 999

Mount Namespace

- Control visibility of mount points
- At creation, gets mounts from current mount namespace
- Useful for
  - Masking /proc, /sys
  - Having private mount and isolated /tmp



PID Namespace

- Restrict the view to processes in the same PID namespace
- Processes IDs start at 1 in each namespace
  - First process created gets PID 1 and is considered special
- If PID 1 disappears (exit, killed…)
  - All process in namespace and its descendant are killed
  - The namespace disappears

Time Namespace

- Allows a process to see different system times

Cgroup Namespace

- Virtualize the view of a process's cgroups
- More on that later

Namespaces

In practice

Namespaces

- The namespaces of a process are visible in */proc*

```
fhuet@gpu ~> ps
    PID TTY          TIME CMD
 970323 pts/0    00:00:00 fish
 976126 pts/0    00:00:00 ps
fhuet@gpu ~> ls -al  /proc/970323/ns
total 0
dr-x--x--x 2 fhuet fhuet 0 sept. 25 14:46 ./
dr-xr-xr-x 9 fhuet fhuet 0 sept. 25 14:25 ../
lrwxrwxrwx 1 fhuet fhuet 0 sept. 25 14:46 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 fhuet fhuet 0 sept. 25 14:46 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 fhuet fhuet 0 sept. 25 14:46 mnt -> 'mnt:[4026531841]'
lrwxrwxrwx 1 fhuet fhuet 0 sept. 25 14:46 net -> 'net:[4026531840]'
lrwxrwxrwx 1 fhuet fhuet 0 sept. 25 14:46 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 fhuet fhuet 0 sept. 25 14:46 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 fhuet fhuet 0 sept. 25 14:46 time -> 'time:[4026531834]'
lrwxrwxrwx 1 fhuet fhuet 0 sept. 25 14:46 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 fhuet fhuet 0 sept. 25 14:46 user -> 'user:[4026531837]'
lrwxrwxrwx 1 fhuet fhuet 0 sept. 25 14:46 uts -> 'uts:[4026531838]'
```

Unshare

- The *unshare* command

```
fhuet@gpu ~ [127]> sudo unshare -h

Usage:
 unshare [options] [<program> [<argument>...]]

Run a program with some namespaces unshared from the parent.

Options:
 -m, --mount[=<file>]      unshare mounts namespace
 -u, --uts[=<file>]        unshare UTS namespace (hostname etc)
 -i, --ipc[=<file>]        unshare System V IPC namespace
 -n, --net[=<file>]        unshare network namespace
 -p, --pid[=<file>]        unshare pid namespace
 -U, --user[=<file>]       unshare user namespace
 -C, --cgroup[=<file>]     unshare cgroup namespace
```

Unshare - example

- Create a shell with namespace PID
  - Shell will get PID 1
- Need some options to have it work like intended

```
fhuet@gpu ~> sudo unshare --fork --pid --mount-proc  /bin/bash
root@gpu:/home/fhuet# ps
    PID TTY          TIME CMD
      1 pts/0    00:00:00 bash
      8 pts/0    00:00:00 ps
root@gpu:/home/fhuet#
```

Control Groups

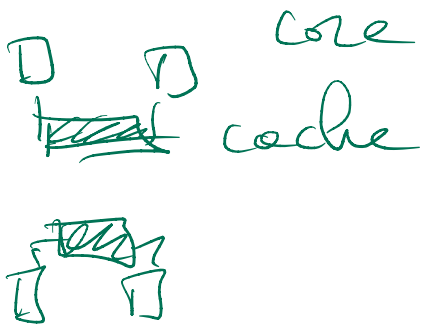
Introduction

- Introduced in 2008
- Enables control and monitoring of resources
  - Per process
- Resources
  - CPU, Memory, Network, Block IO, Devices
- Subsystem :
  - A module that acts as a resource controler
  - Deals with a single resource (e.g. CPU time, memory…)

Introduction - 2

- Tasks
  - Processes in cgroup terminology
- Cgroup
  - Associate a set of tasks with a set of parameters for one or more subsystems
  - Cgroups are hierarchical
  - Child cgroups inherit certain attributes from their parent cgroup
  - Many hierarchies can coexist (not a single root)
- Cgroupfs, aka cgroup filesystem
  - Located in /sys/fs/cgroup (Read only)
  - Can be manipulated with standard fs commands
- Cgroups – v1 (*default)*
  - Per subsystem hierarchy
- Cgroups – v2
  - Single control group hierarchy

Available subsystems



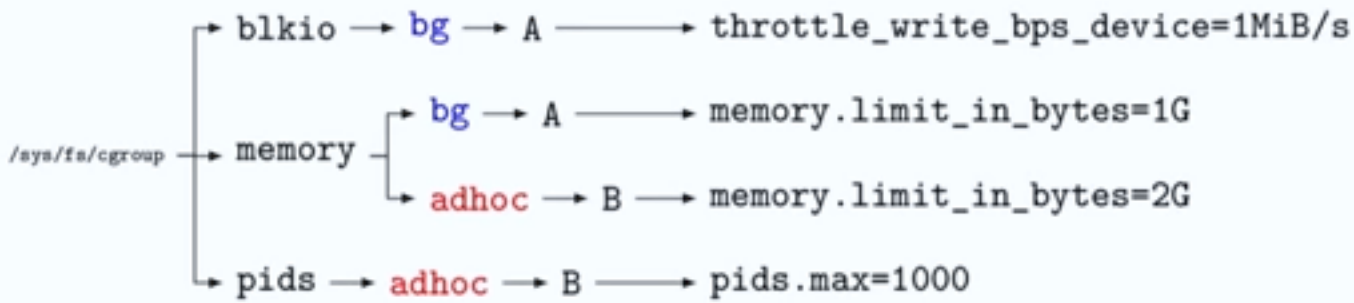
- 13 in latest kernel
- blkio
  - Limits on I/O access to/from block devices
- cpu
  - Provide cgroup tasks access to the CPU
- cpuacct
  - Automatic reports on CPU resources used by tasks in a cgroup
- cpuset
  - Assigns individual CPUs (on a multicore system) and memory nodes to tasks in a cgroup
- devices
  - Allows or denies access to devices by tasks in a cgroup.

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/ch01

Available subsystems

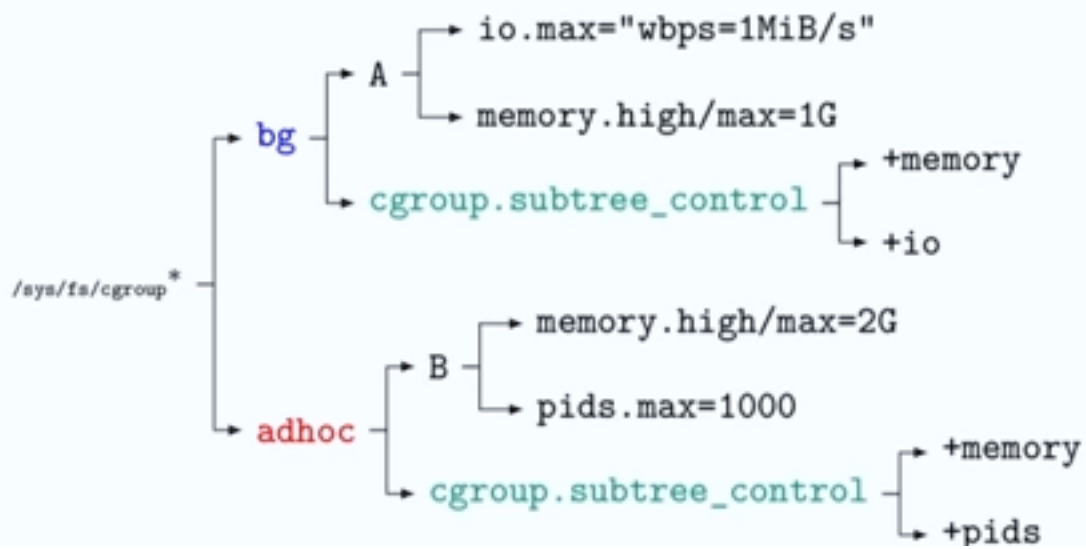
- freezer
  - Suspends or resumes tasks in a cgroup
- memory
  - Sets limits on memory use by tasks in a cgroup and generates automatic reports on memory resources
- net_cls
  - Tags network packets with a class identifier (classid) that allows the Linux traffic controller (tc) to identify packets originating from a particular cgroup task.
- net_prio
  - Provides a way to dynamically set the priority of network traffic per network interface.
- ns
  - Namespace subsystem
- perf_event
  - Identifies cgroup membership of tasks and can be used for performance analysis

Hierarchy in cgroupv1

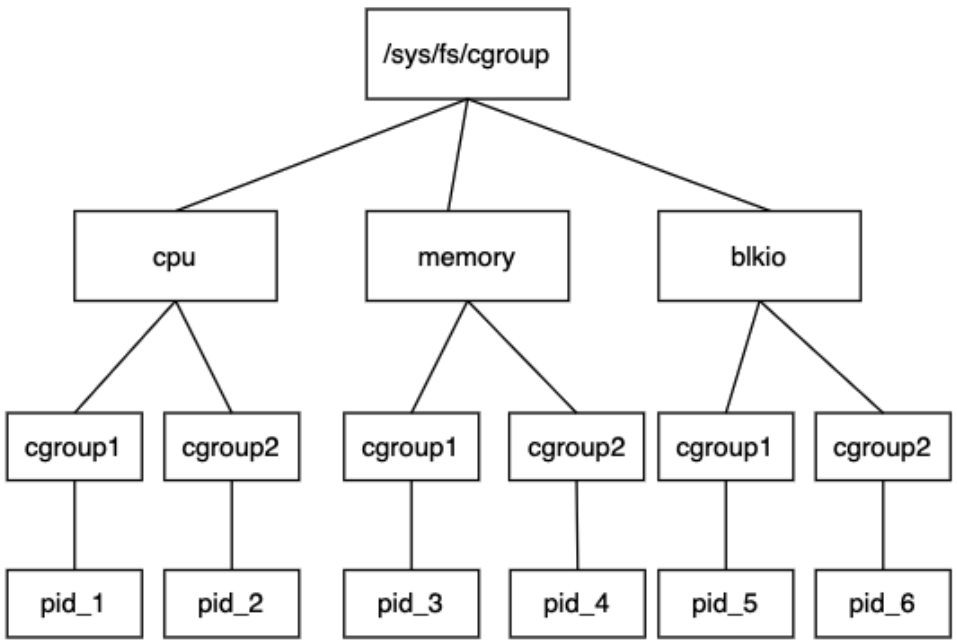


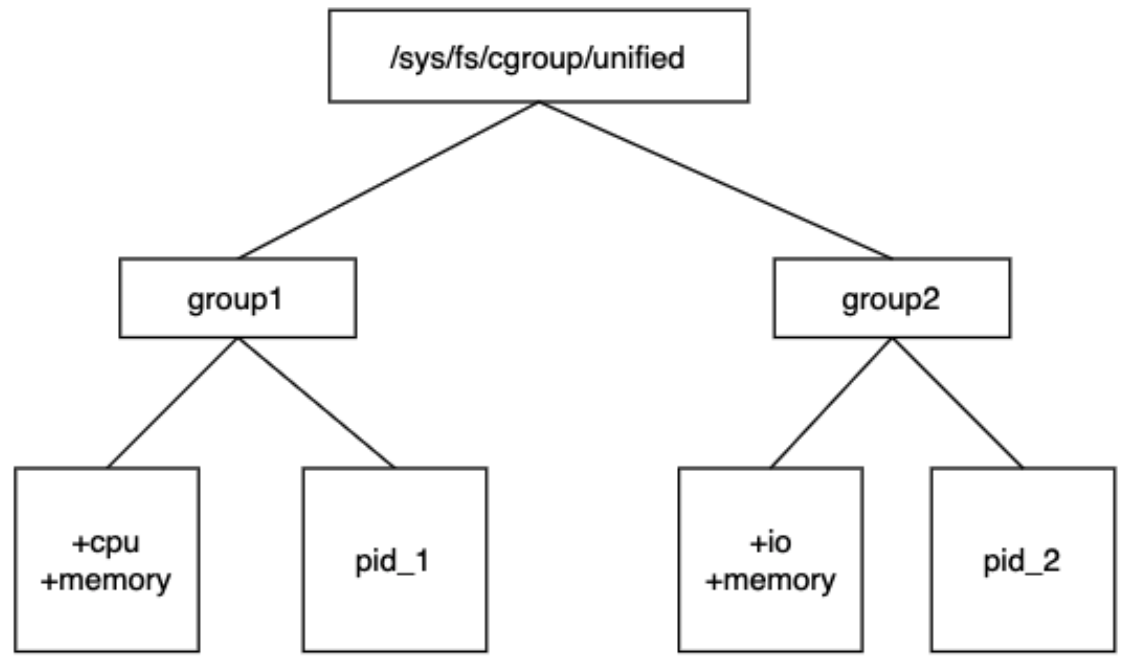
VS

Hierarchy in cgroupv2



https://medium.com/some-tldrs/tldr-understanding-the-new-control-groups-api-by-rami-rosen-980df476f633





https://www.sobyte.net/post/2022-07/blkio-cgroup/

Control Groups

In practice

Creating a new cgroups (v1)

- Use of cgroupfs
  - Standard filesystems commands
- A cgroup is a subdirectory in subsystems of cgroupfs
  - /sys/fs/cgroup/cpuset
  - /sys/fs/cgroup/memory …
- *mkdir*
  - Create a directory
  - Will also create all necessary files for cgroups
    - Content will depend on the subsystem
- Can also use the cgroup-tools package

```
root@gpu /s/f/c/cpuset# mkdir cg1
root@gpu /s/f/c/cpuset# ls cg1
cgroup.clone_children  cpuset.cpus            cpuset.mem_exclusive   cp
cgroup.procs           cpuset.effective_cpus  cpuset.mem_hardwall    cp
cpuset.cpu_exclusive   cpuset.effective_mems  cpuset.memory_migrate  cp
root@gpu /s/f/c/cpuset#
```

Managing limits and tasks

- Limits depends on the subsystem
- For cpuset

```
root@gpu /s/f/c/cpuset# more cpuset.cpus
0-19
root@gpu /s/f/c/cpuset#
```

```
root@gpu /s/f/c/cpuset# more cpuset.cpu_exclusive
1
```

- Processes (tasks) are specified in the *tasks* file
  - Text file with PID
  - By default contains inherited tasks
- Add the PID to the *task* file will move the process to the cgroup
  - Its future subprocess will be part of this cgroup

Adding a process to a cgroup

```
root@gpu /s/f/c/c/cg1# fish
Welcome to fish, the friendly interactive shell
Type `help` for instructions on how to use fish
root@gpu /s/f/c/c/cg1# ps
    PID TTY          TIME CMD
 643136 pts/0    00:00:00 sudo
 643137 pts/0    00:00:00 fish
 647505 pts/0    00:00:00 fish
 647560 pts/0    00:00:00 ps
1355435 pts/0    00:00:38 tini
1355499 pts/0    00:00:00 sudo
2661386 pts/0    00:00:00 sh
2661522 pts/0    06:32:21 python3
root@gpu /s/f/c/c/cg1# cat > tasks
647505
```

Getting a process's cgroup

- Use the *proc* virtual filesystem
- Look into /proc/<pid>/cgroup

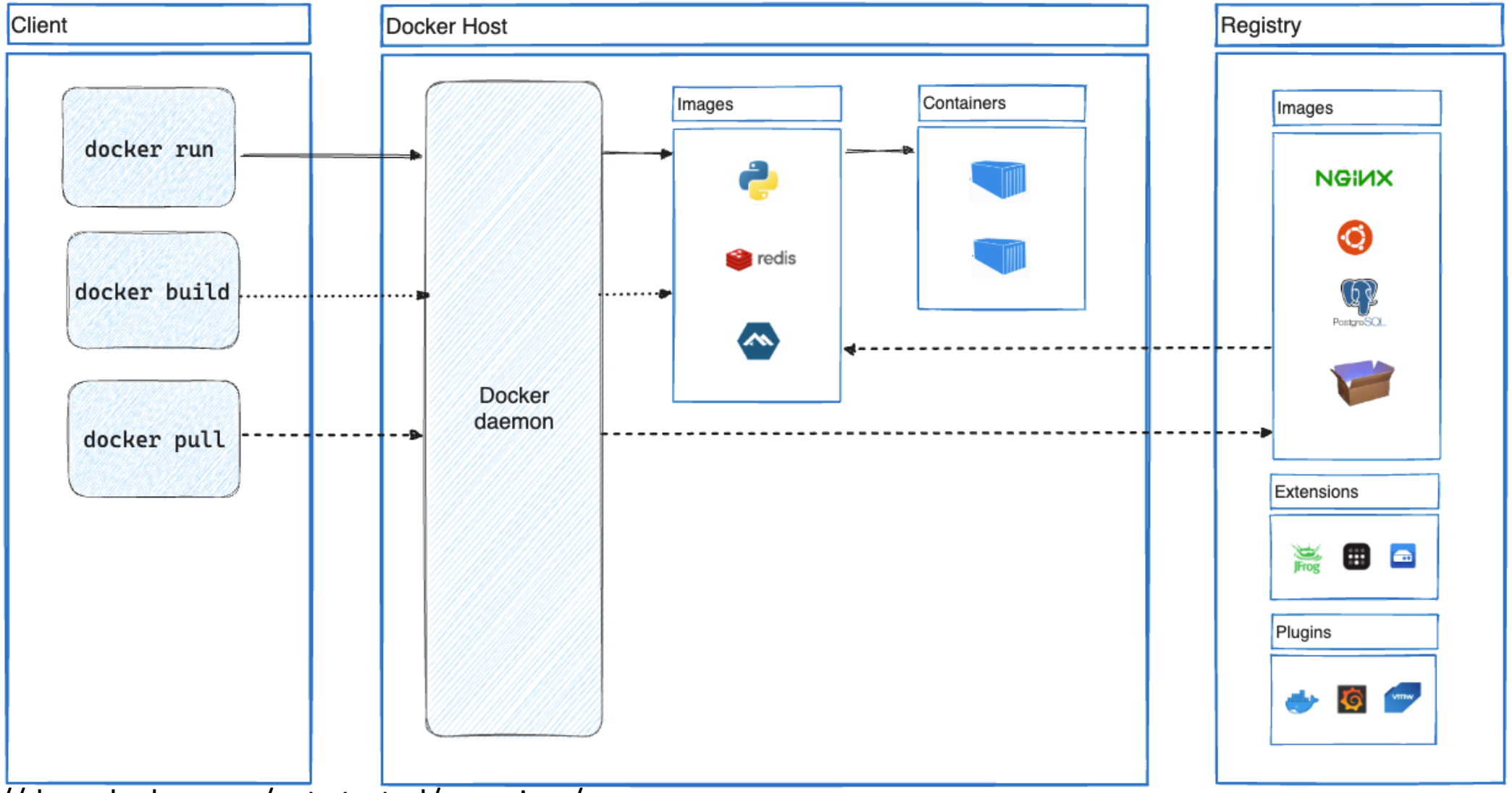
```
root@gpu /s/f/c/c/cg1# more /proc/647505/cgroup
13:perf_event:/
12:blkio:/user.slice
11:devices:/user.slice
10:pids:/user.slice/user-1001.slice/session-14221.scope
9:cpuset:/cg1
8:memory:/user/root/0
7:net_cls,net_prio:/
6:rdma:/
5:freezer:/user/root/0
4:cpu,cpuacct:/user.slice
3:misc:/
2:hugetlb:/
1:name=systemd:/user/root/0
0::/user.slice/user-1001.slice/session-14221.scope
```

Docker

Introduction

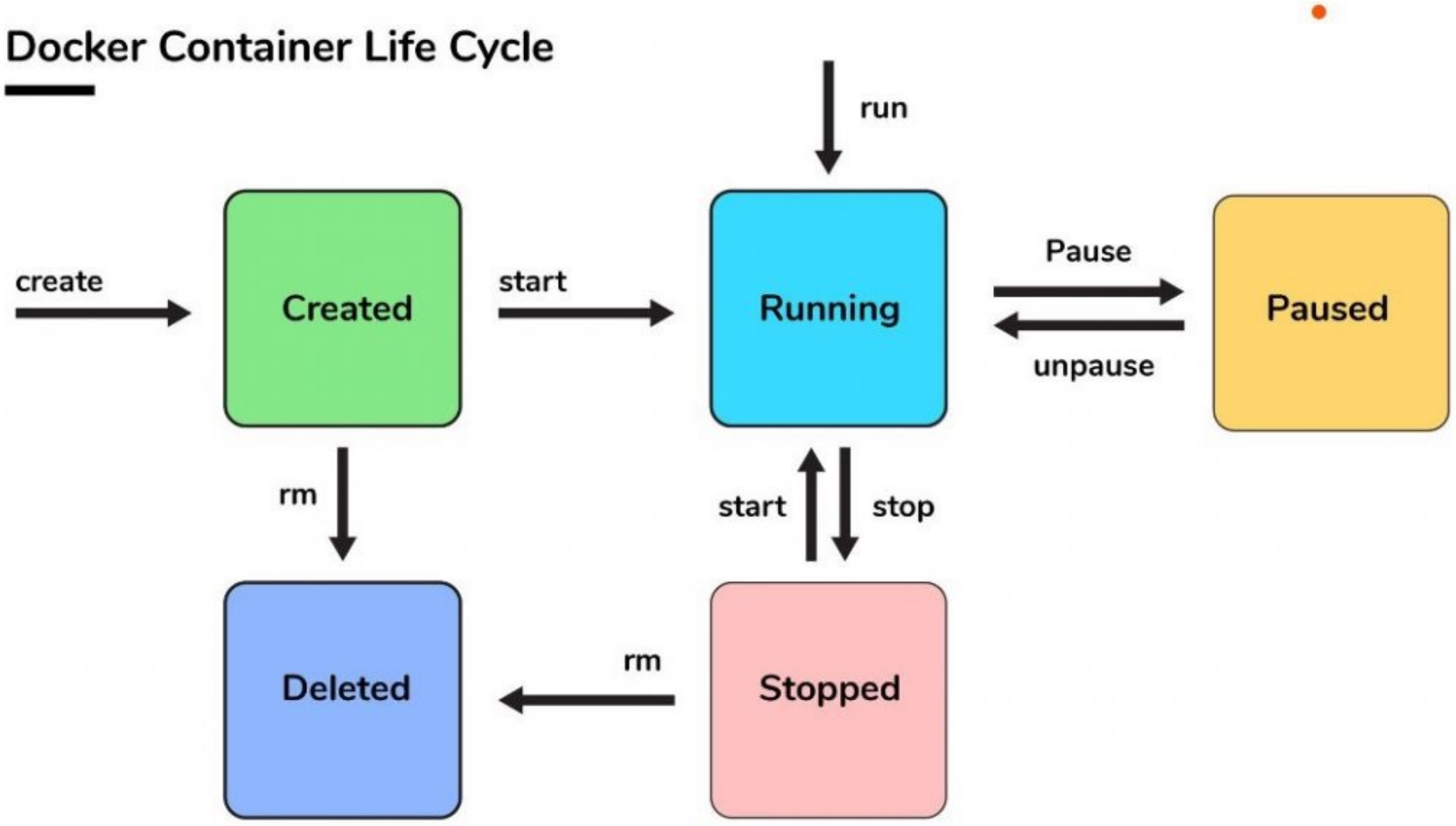
- Docker is a container engine
  - Package, deploy and run applications
- It’s a service running on a machine
  - Relies on namespaces and cgroups to isolate processes into containers
- Native to Linux
- Relies on images
  - A read-only template with instructions for creating a Docker container
  - Repository : https://hub.docker.com
  - Custom images can be created, based on existing ones

Architecture



https://docs.docker.com/get-started/overview/

Lifecycle of a container



https://www.devopsschool.com/blog/lifecycle-of-docker-containers/

Data

- All files created in a container are stored in a special container layer
- Consequences
  - This layer is tight to the host (format) and hard to move
  - Performance is reduced
  - Data are lost when removing container
- For persistent data
  - Volumes
    - Stored on the host machine (*/var/lib/docker/volumes*)
    - Managed by Docker, not accessed by processes on the host machine
  - Bind mount
    - Files/Directory of the host system
    - Can be accessed by any process

Volumes vs Bind mounts

- Volumes are
  - Created the first time you need them
  - Persistent, need to be explicitly removed
  - Can be shared among multiple containers
- Bind mounts
  - Refer to existing files on the host
  - Useful for sharing between host and container

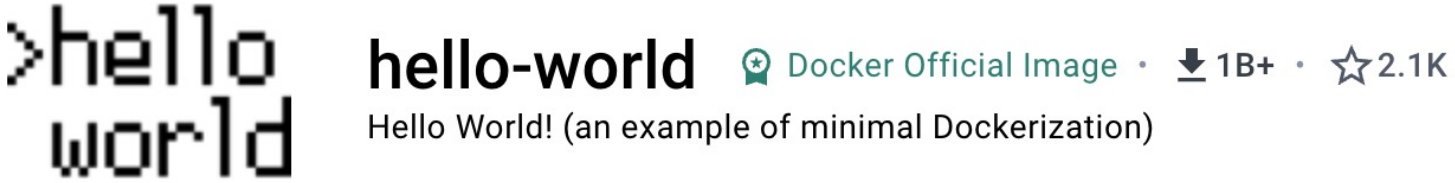
Docker

Usage

Commands

- Docker is controlled with the *docker* command
- Direct commands
  - *docker pull | start | stop | restart | ps | images*
- Indirect commands
  - *docker volume create | ls | prune | rm*
  - *docker network connect | create | ls | prune | rm*

Pulling an image



https://hub.docker.com/_/hello-world

```
fhuet@gpu ~ [1]> docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:4f53e2564790c8e7856ec08e384732aa38dc43c52f02952483e3f003afbf23db
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
fhuet@gpu ~>
```

Docker create and run

```
fhuet@gpu ~ [1]> docker create hello-world
f687a79f3c8c33b5e17f21a14386aa0afef1b5c7739d4b83138db712b5115a50
fhuet@gpu ~>
```

```
fhuet@gpu ~> docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

```
fhuet@gpu ~> docker ps
CONTAINER ID   IMAGE                                                                                  CO
                                                                                                      NAMES
e17b985ff207   cschranz/gpu-jupyter:v1.5_cuda-11.6_ubuntu-20.04_python-only                            "t
                                                                                                      jupyter-admin
048ff2315166   cschranz/gpu-jupyter:v1.5_cuda-11.6_ubuntu-20.04_python-only                            "t
                                                                                                      jupyter-fhuet
7f0d532a5dd9   gpu-jupyter                                                                                "t
                                                                                                      gpu-jupyter_1
7b86cee7afcf   jupyterhub                                                                                 "j
                                                                                                      jupyterhub
8245cc47f071   text-generation-webui-text-generation-webui                                                "/
p, :::5005->5005/tcp, 0.0.0.0:7860->7860/tcp, :::7860->7860/tcp                                       text-generatio
14c38a4077c0   lscr.io/linuxserver/code-server:latest                                                     "/
                                                                                                      code-server
```

Docker

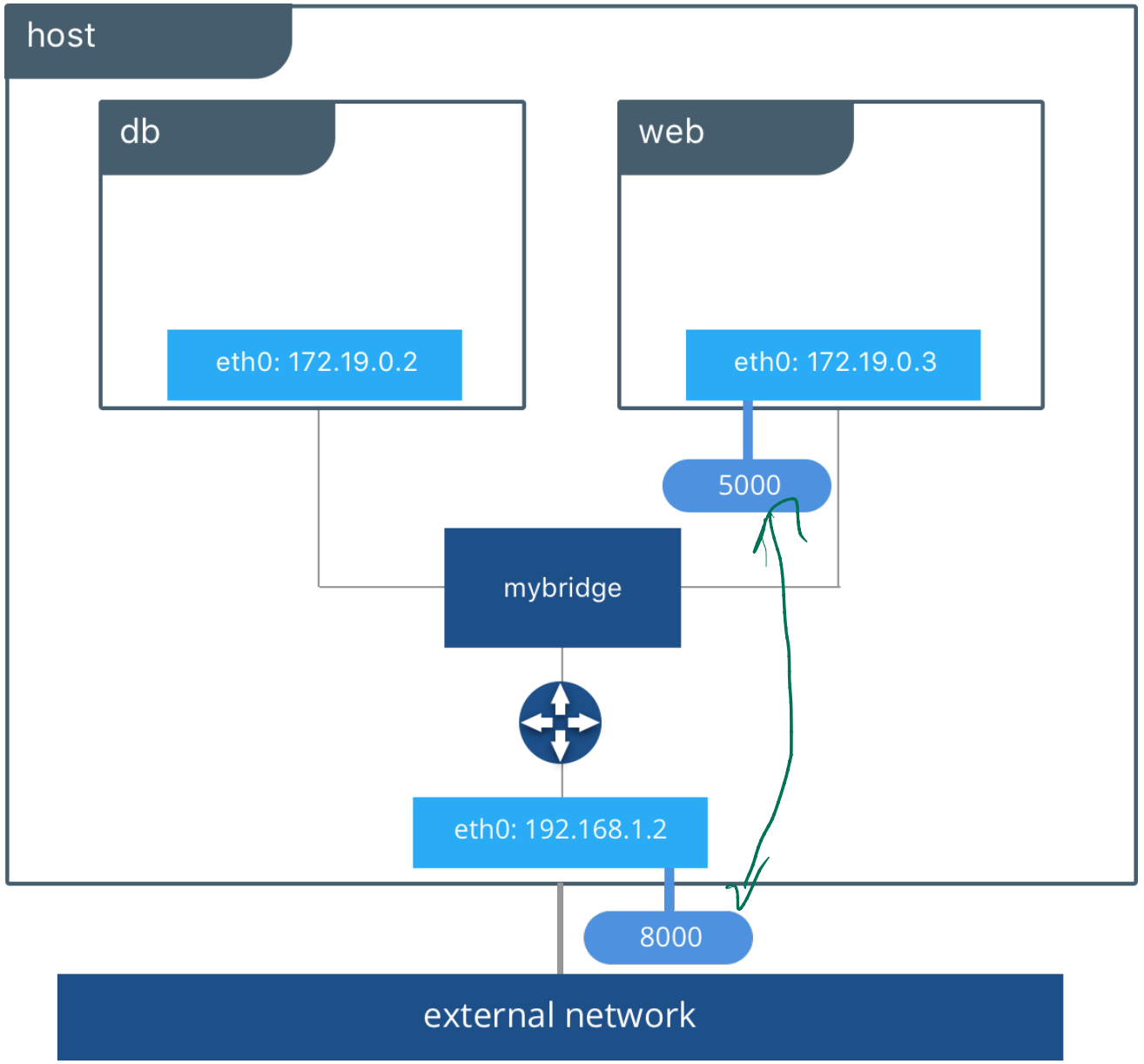
Network

Overview

- A container is connected to a network
  - IP address, dns, routing table…
- Container is connected to a Docker network
  - IP from this subnet
  - Can join multiple networks
- Network functionalities are provided by drivers
  - *none, bridge, host, overlay, ipvlan, macvlan*
  - *none* isolates the container completely

Bridge

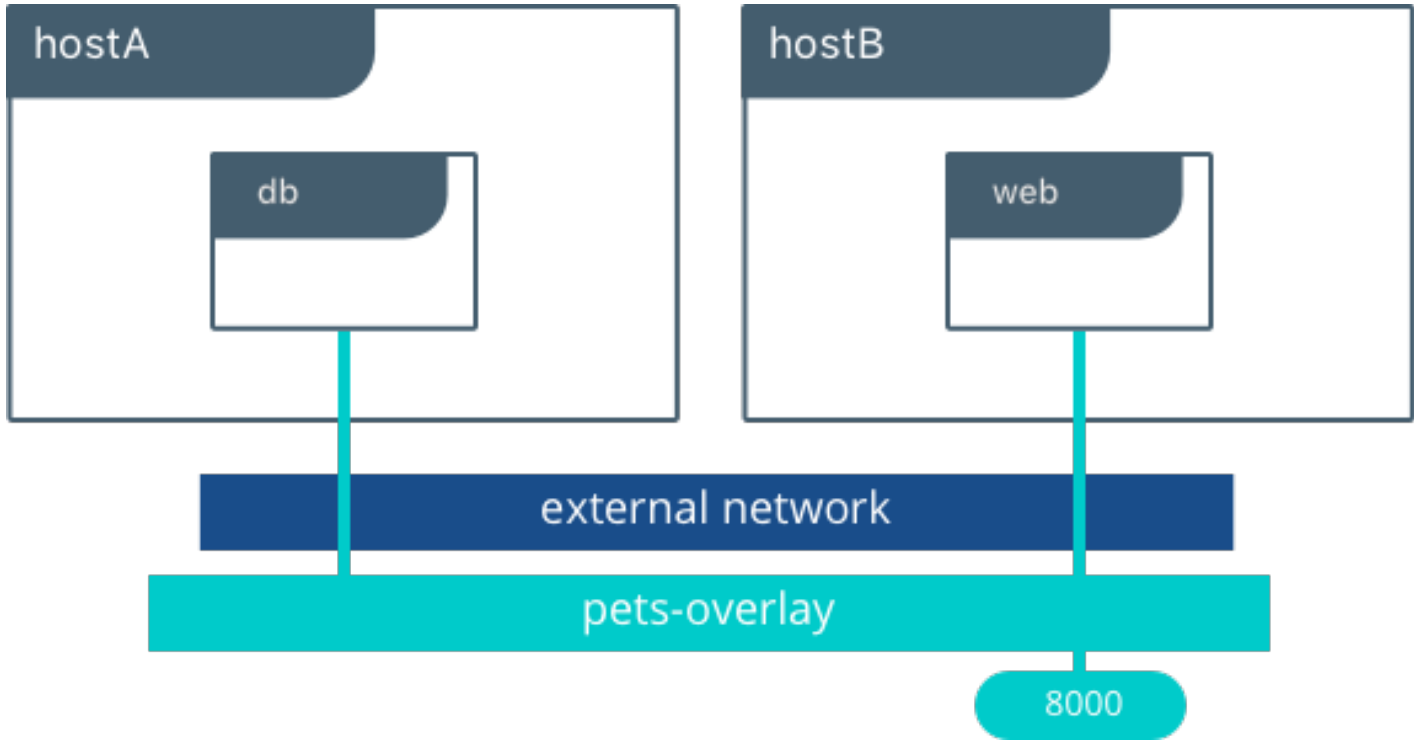
- Creates a private network internal to the host
  - Containers on this internal network can communicate
  - Containers can expose ports to the host
- Service discovery on a single host
- Default network



https://www.docker.com/blog/understanding-docker-networking-drivers-use-cases/

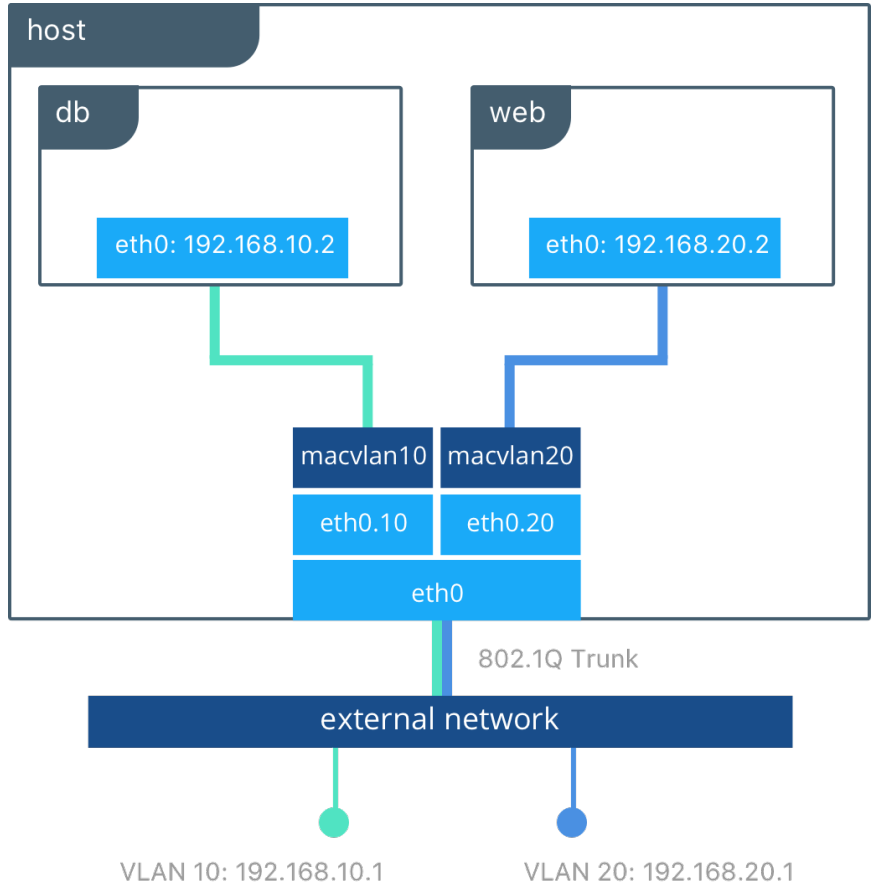
Overlay

- Designed for multi-host
- Creates a distributed network
  - Routing handled by docker daemons
- Useful for docker Swarm and Kubernetes



Others

- Host
  - Container uses the host's network stack
  - No namespace separation, all interfaces of the host are visible
- MACVLAN
  - Allows containers to have IP address on the physical network

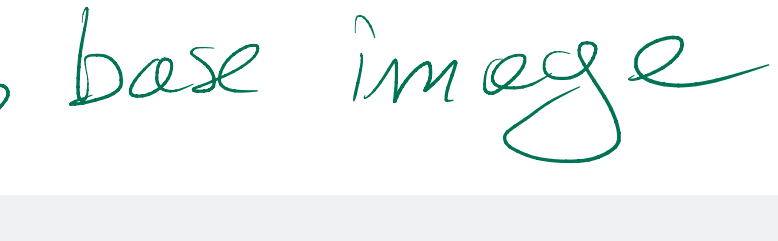


Docker

Custom images

Overview

- Why create custom images ?
  - Cannot find one in official repository
  - Adapt an existing one with specific application or version
- Simple process
  - Use of a *Dockerfile*
  - Text file with instructions




```
FROM alpine
CMD ["echo", "Hello StackOverflow!"]
```

Some Dockerfile instructions

FROM	Name of the base image to use
RUN	Execute a command during the build process
COPY	Copy files from host folders to image
CMD	Default command and arguments to execute when container is started. Can be overridden from the command line
ENTRYPOINT	Specify the main command that should be executed when a container is started. Default entrypoint is */bin/sh –c*

Entrypoint vs cmd

- Cmd is used to pass parameters
  - Overridden from the command line
- Entrypoint specify the command to start
  - Not easily overridden
- Can be combined
  - Start Entrypoint + Cmd
  - Entrypoint for base command
  - Cmd for parameters, can be overridden

Docker

Other OS

Native docker

- Docker is tight to Linux
  - Cgroups and namespaces
- Non native docker
  - Run a VM with Linux
  - Run docker inside VM
  - No isolation of host processes
- Native docker
  - On windows (https://github.com/docker/labs/blob/master/windows/windows-containers/README.md)
  - On MacOS (https://macoscontainers.org/)