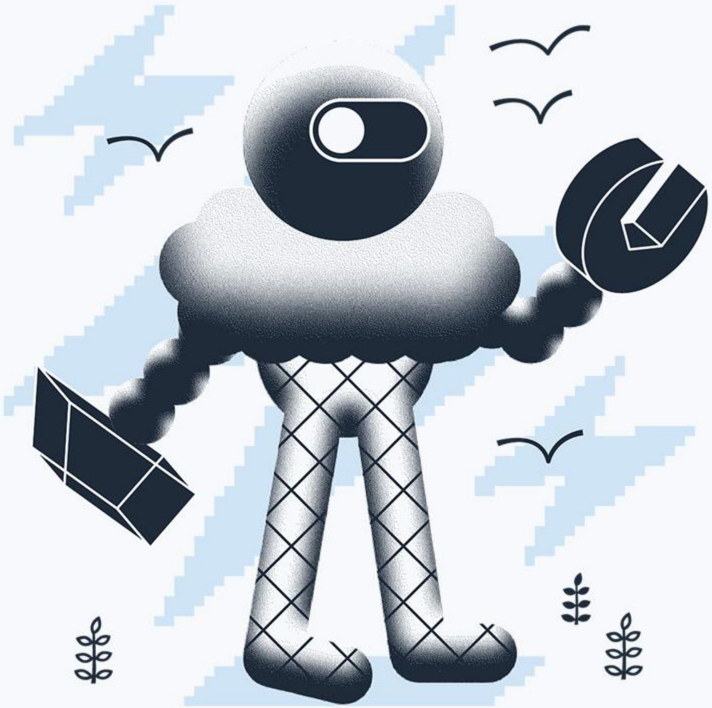
# Containers: A Peek Under the Hood



Aleš Brelih

Backend software engineer

13 APRIL  2024

# A little bit about myself

Before

R&D engineer

@ https://www.arctur.si/

Now

Backend software engineer

@ https://www.3fs.cloud

# How did I get here?

- I've been actively working with containers for over 7 years.
- For much of that time, I didn't really understand what was happening under the hood. Tools like Docker and Kubernetes really do a good job at hiding the complexity.
- My interest in security made me dig deeper, leading to a realization: you can't protect what you're unaware of.

# What is a container?

# Let's google it

## Docker

"A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another."

## Google Cloud

"Containers are lightweight packages of software that contain all of the necessary elements to run in any environment."

## Red Hat

"Containers are technologies that allow the packaging and isolation of applications with their entire runtime environment—all of the files necessary to run."

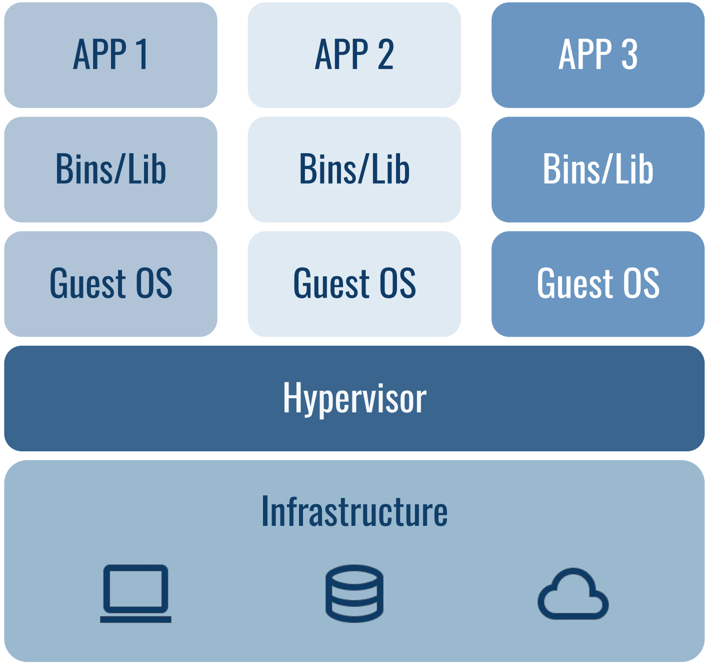
# Probably most heard answer?

“ Oh you mean docker? It is something like a Virtual Machine but lightweight.”

# Truth?

- They are **just processes**, just like any other application that runs directly on the host
- The major difference between containers and VMs is that containers **share host kernel** while VMs virtualize hardware and run multiple guest operating systems on a single physical machine.

## Virtual Machines



APP 1 | APP 2 | APP 3

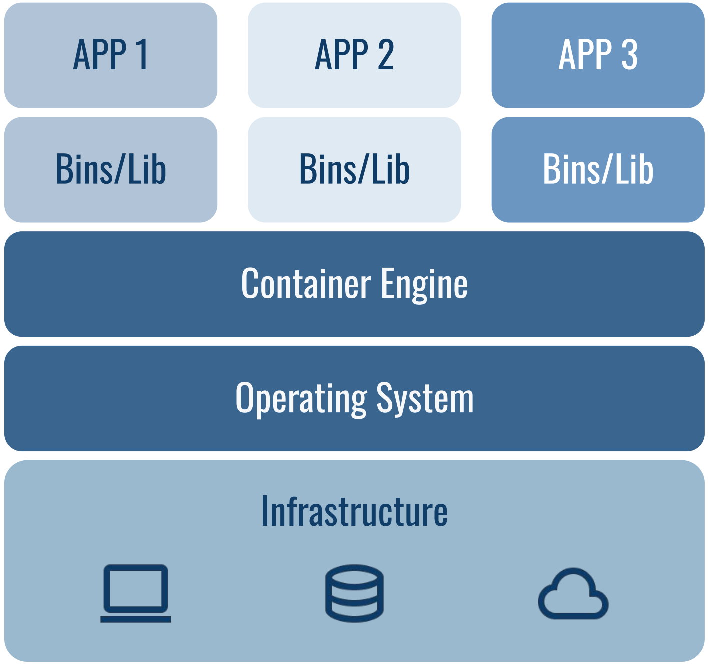
Bins/Lib | Bins/Lib | Bins/Lib

Guest OS | Guest OS | Guest OS

Hypervisor

Infrastructure

## Containers



APP 1 | APP 2 | APP 3

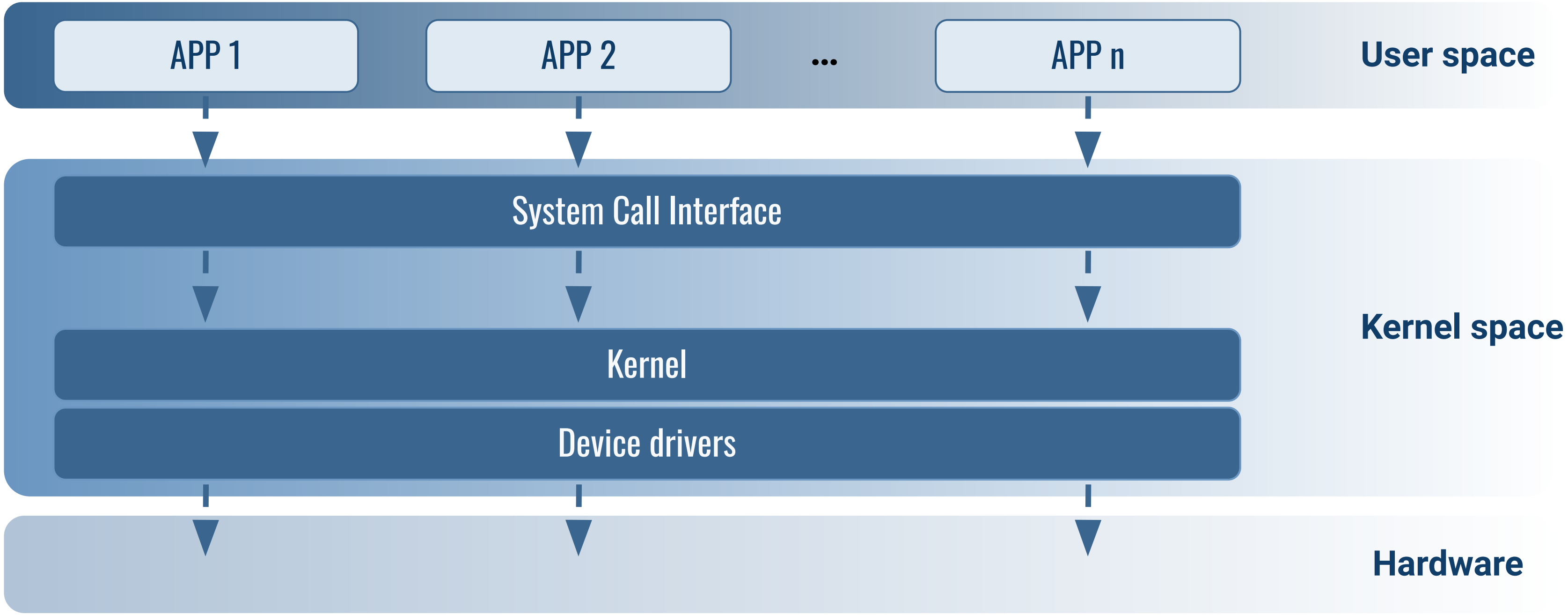
Bins/Lib | Bins/Lib | Bins/Lib

Container Engine

Operating System

Infrastructure

# Why does the shared kernel matter?



# DEMO

It is just a process.

# What makes a process a container?

### Isolation

Processes should run independently, unaware of other processes or the host system.

### Encapsulation

Wraps everything an application needs to run (including dependencies and libraries) into a single package.

### Resource restriction

Controls and limits the amount of CPU, memory, and disk resources that each container can use.

# Namespaces

Isolation

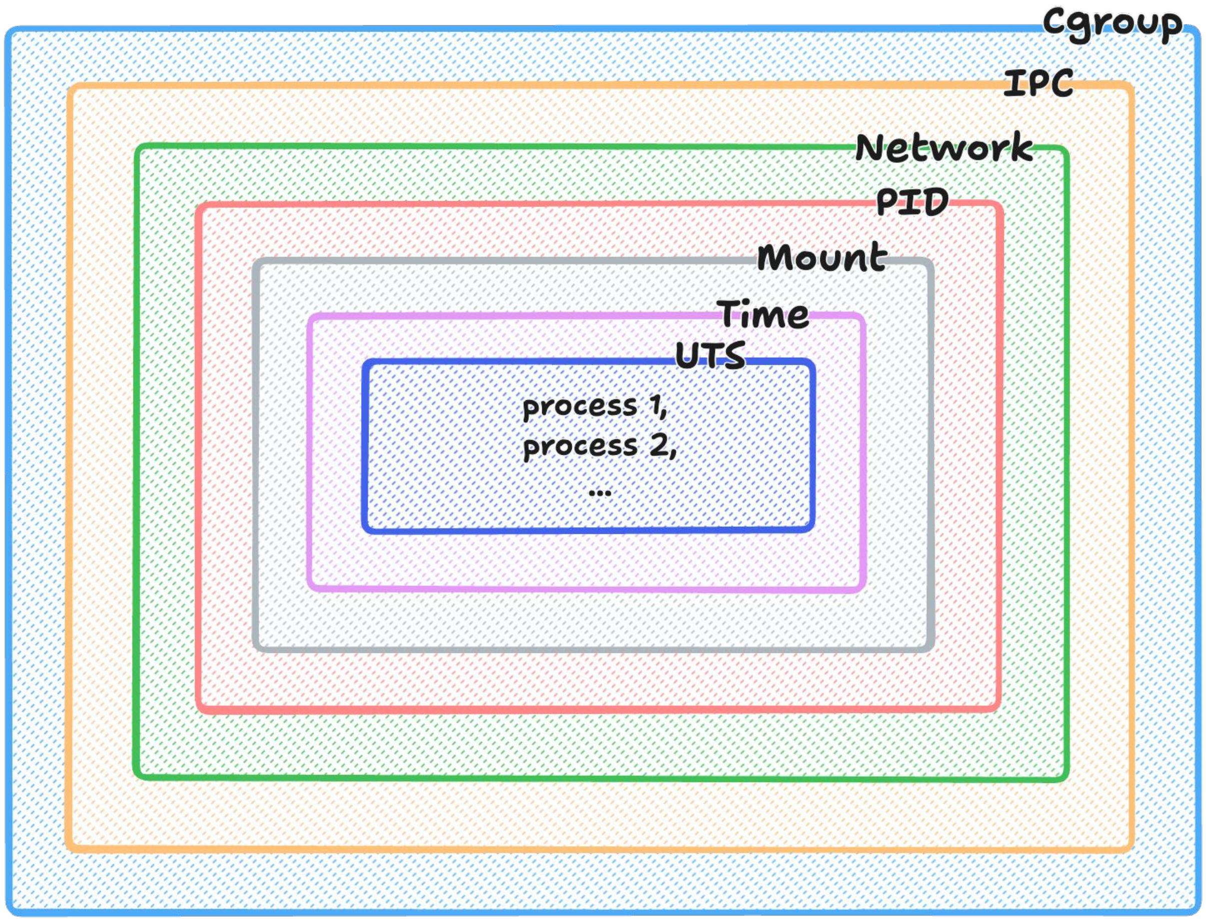
# Namespaces

- Allow a process to have its own isolated instance of global resources (e.g., process IDs, network interfaces).
- They limit the potential impact of malicious processes.
- Changes to the system resource are visible to other processes that are members of the namespace, but are invisible to other processes.

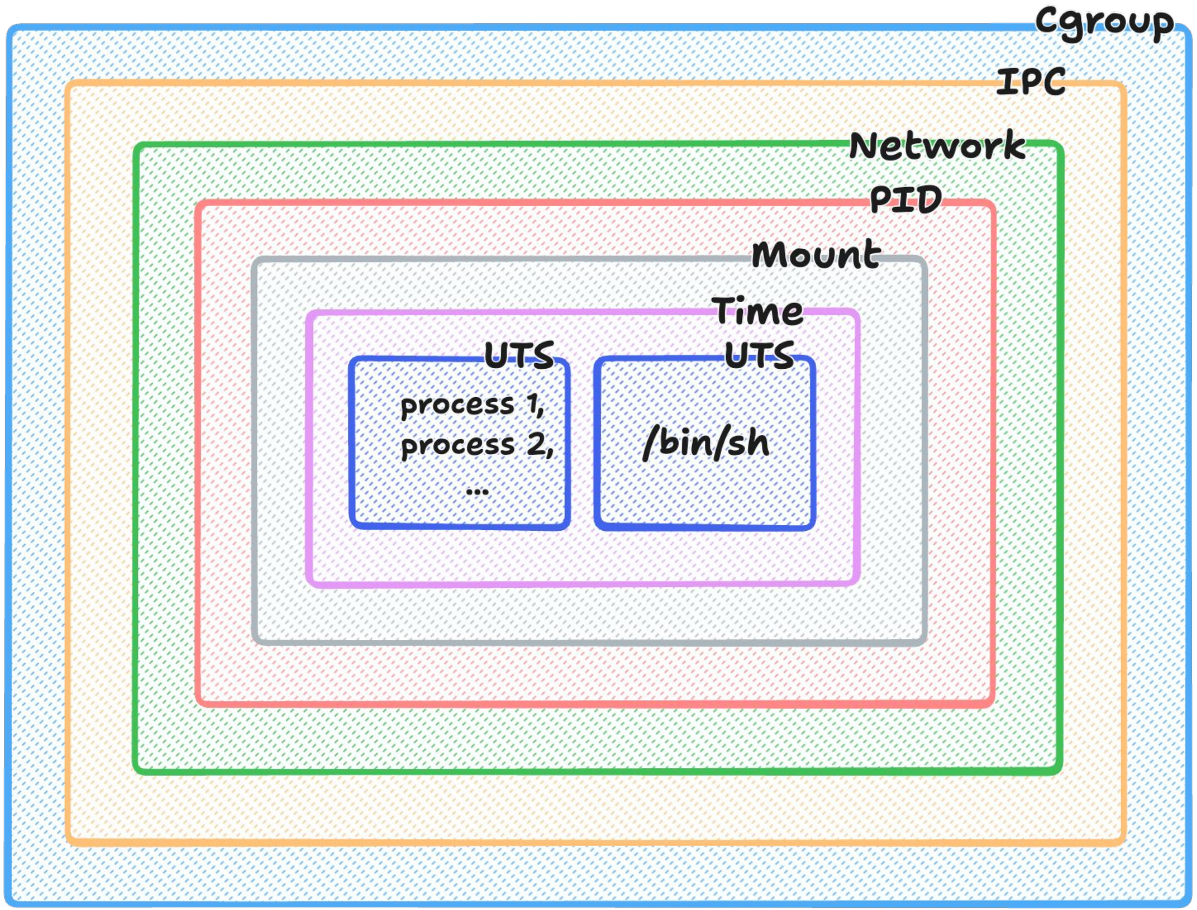
# Types

- Control Groups (Cgroups)
- Inter Process Communication (IPC)
- Network
- Mount
- Process ID (PID)
- Time
- User
- Unix Time Sharing (UTS)

# Default namespace layout



# What we want to do in demo



# DEMO

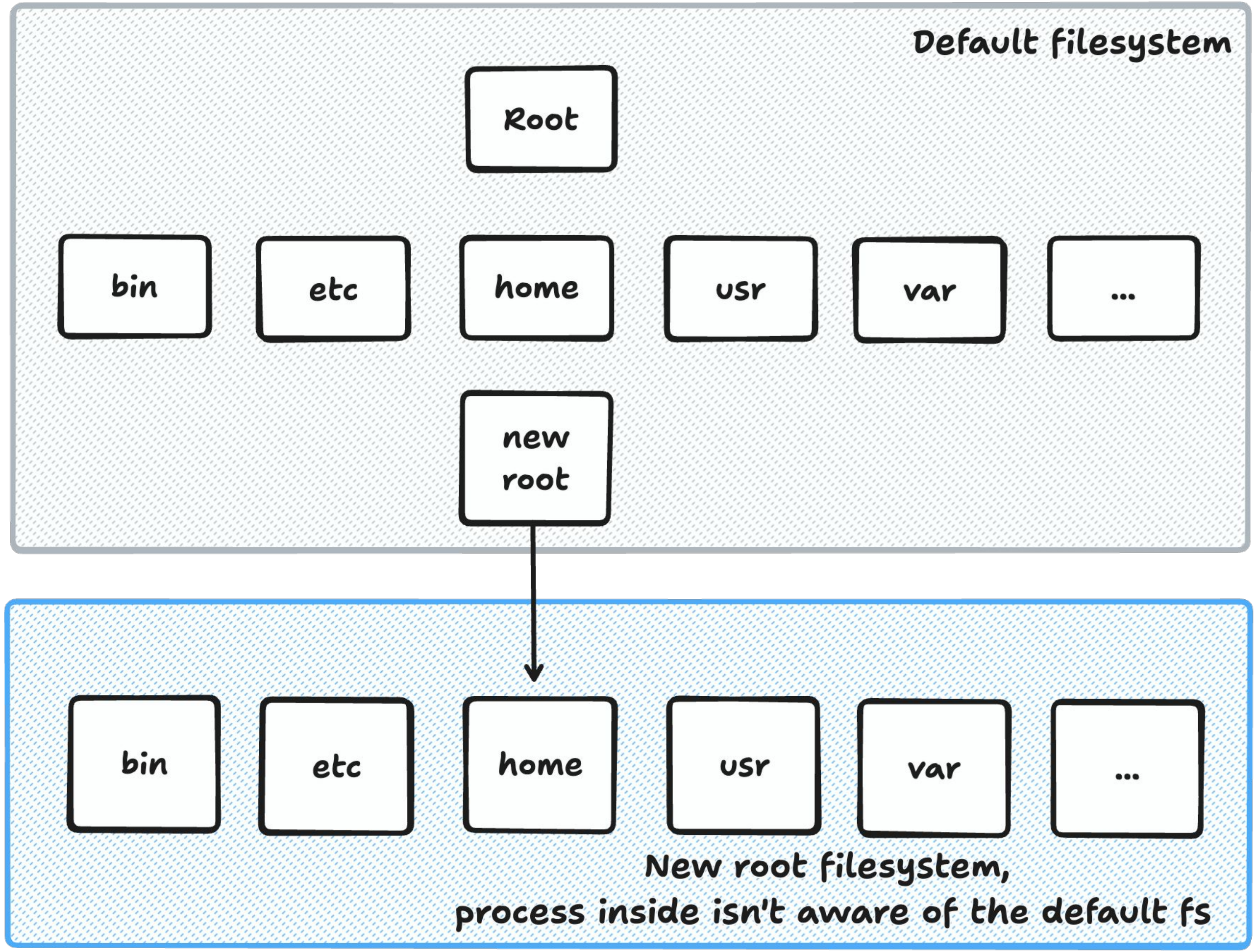
- Unix Time Sharing namespace
- Process ID namespace

# Chroot

Encapsulation

# Chroot

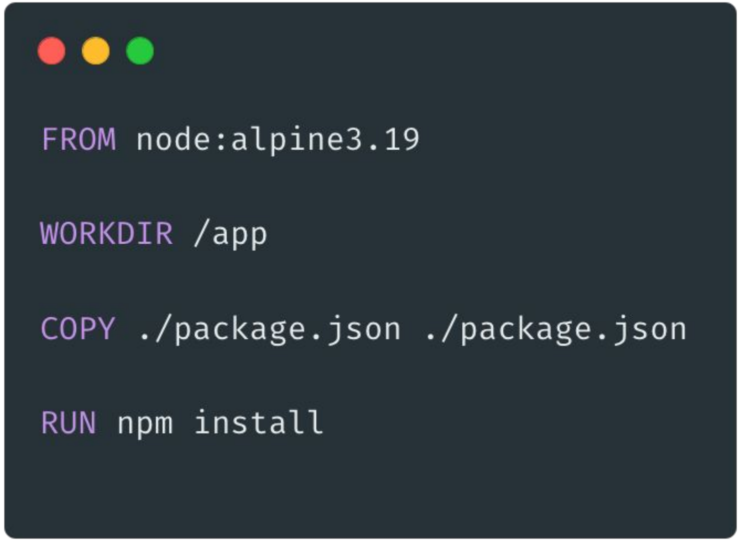
- Used to run command or interactive shell with a new root directory.


Default filesystem
Root
bin
etc
home
usr
var
...
new root
bin
etc
home
usr
var
...
New root filesystem,
process inside isn't aware of the default fs

# DEMO

- Chroot.

# Docker image layering

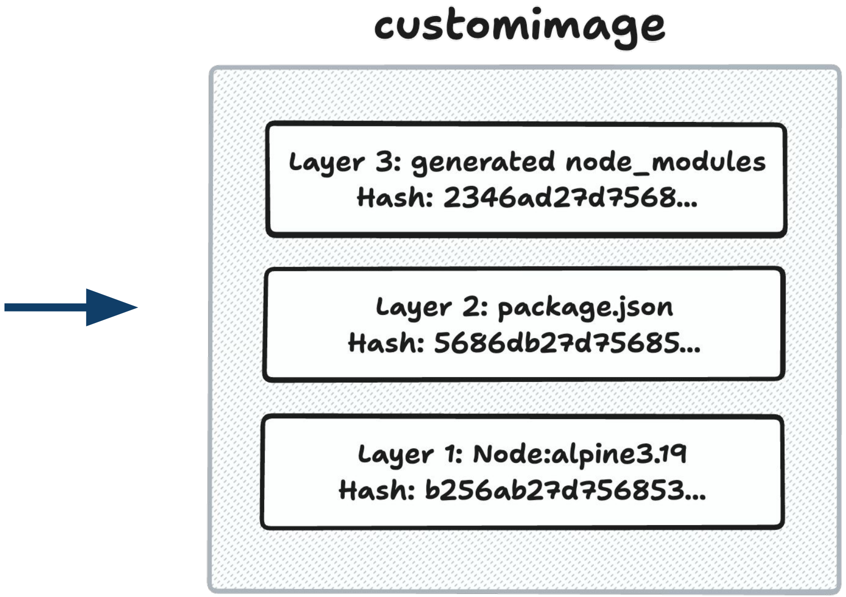


```
FROM node:alpine3.19

WORKDIR /app

COPY ./package.json ./package.json

RUN npm install
```



customimage

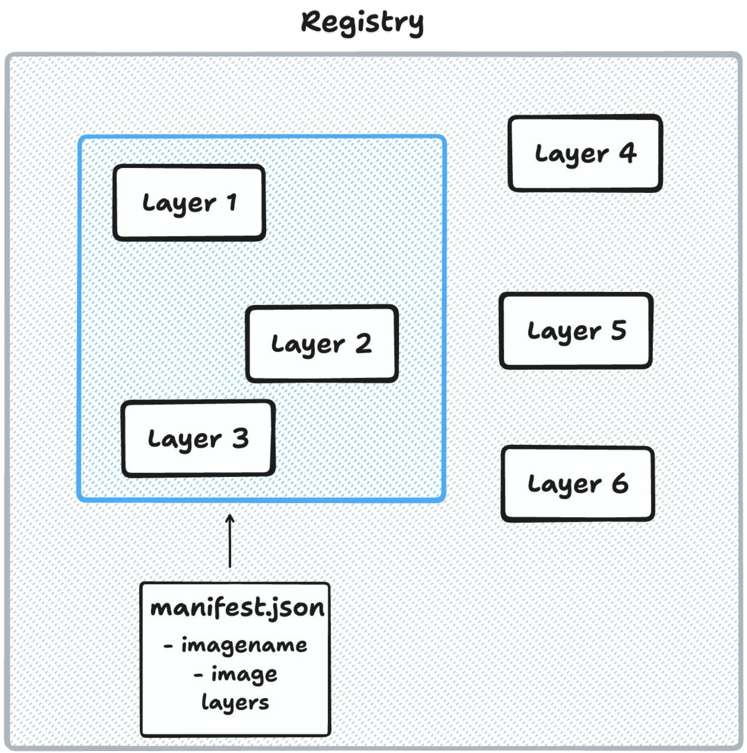
Layer 3: generated node_modules
Hash: 2346ad27d7568...

Layer 2: package.json
Hash: 5686db27d75685...

Layer 1: Node:alpine3.19
Hash: b256ab27d756853...

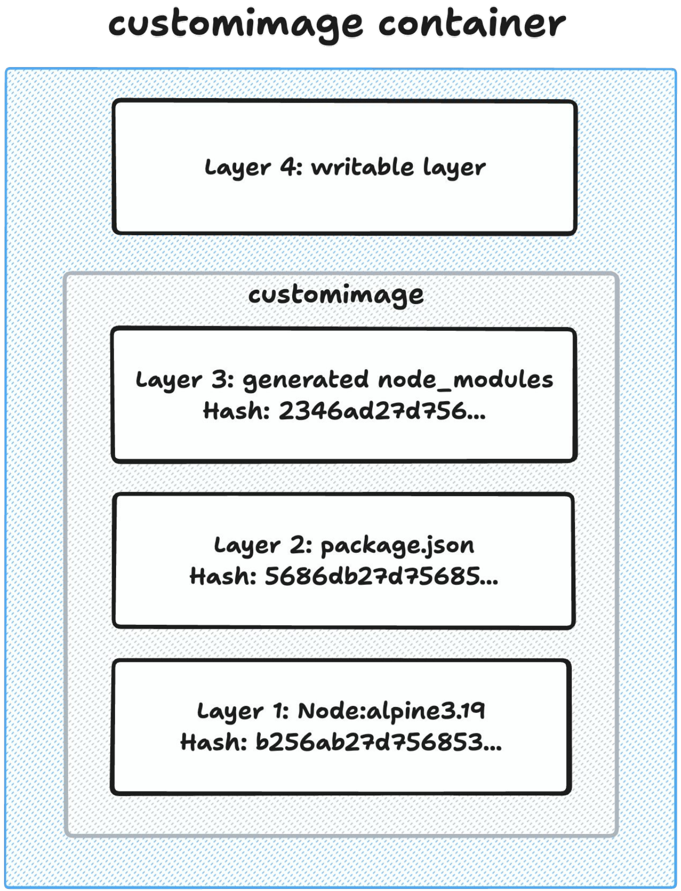
# Layers inside container registry

- Layers are independent entities in a Docker registry.
- Manifest.json defines the composition of a Docker image



# Once used

- A new writable layer on top of the image's read-only layers is created.
- Allowing for the original image to remain unchanged and reusable.



# DEMO

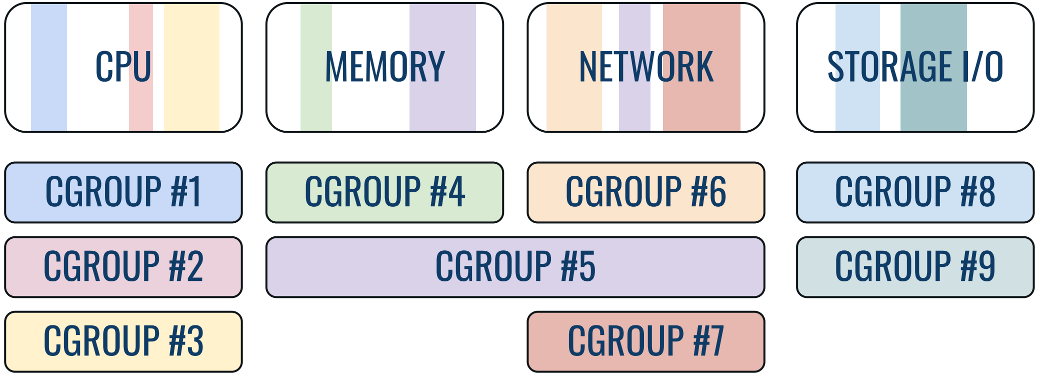
- Docker image inspection.

# Controls Groups

Resource limitations

# Cgroups

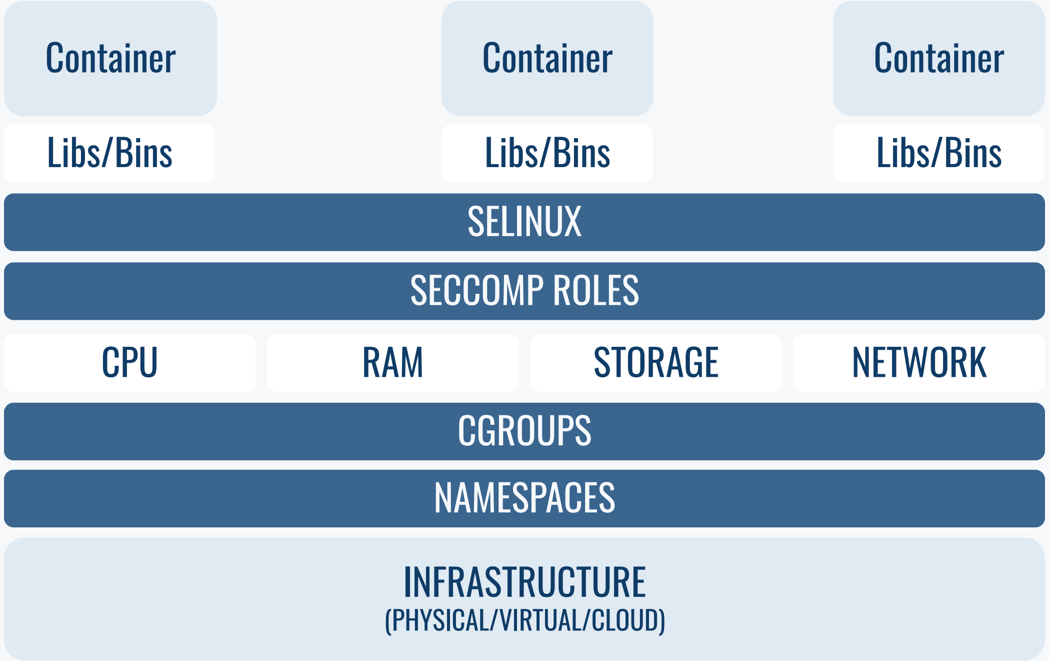
- Allow you to allocate resources (CPU time, system memory, network bandwidth, or combinations of these resources among processes)
- Cgroups are organized hierarchically (child cgroups inherit some of the attributes of their parents)



# DEMO

- Cgroups in actions.

# Let's put everything together.



Container Container Container

Libs/Bins Libs/Bins Libs/Bins

SELINUX

SECCOMP ROLES

CPU RAM STORAGE NETWORK

CGROUPS

NAMESPACES

INFRASTRUCTURE
(PHYSICAL/VIRTUAL/CLOUD)

# Container escapes

And how to be “safer”

# Be careful when using

- **–privileged** flag (124k files on Github)
- **-v /var/run/docker.sock:/var/run/docker.sock** mount (171k files on Github)
- **–cap-add=*** flags (52 files on Github)

# DEMO

- Escape from a privileged container.
- Escape out of container with mounted docker.sock.

3fs

# QUESTIONS?