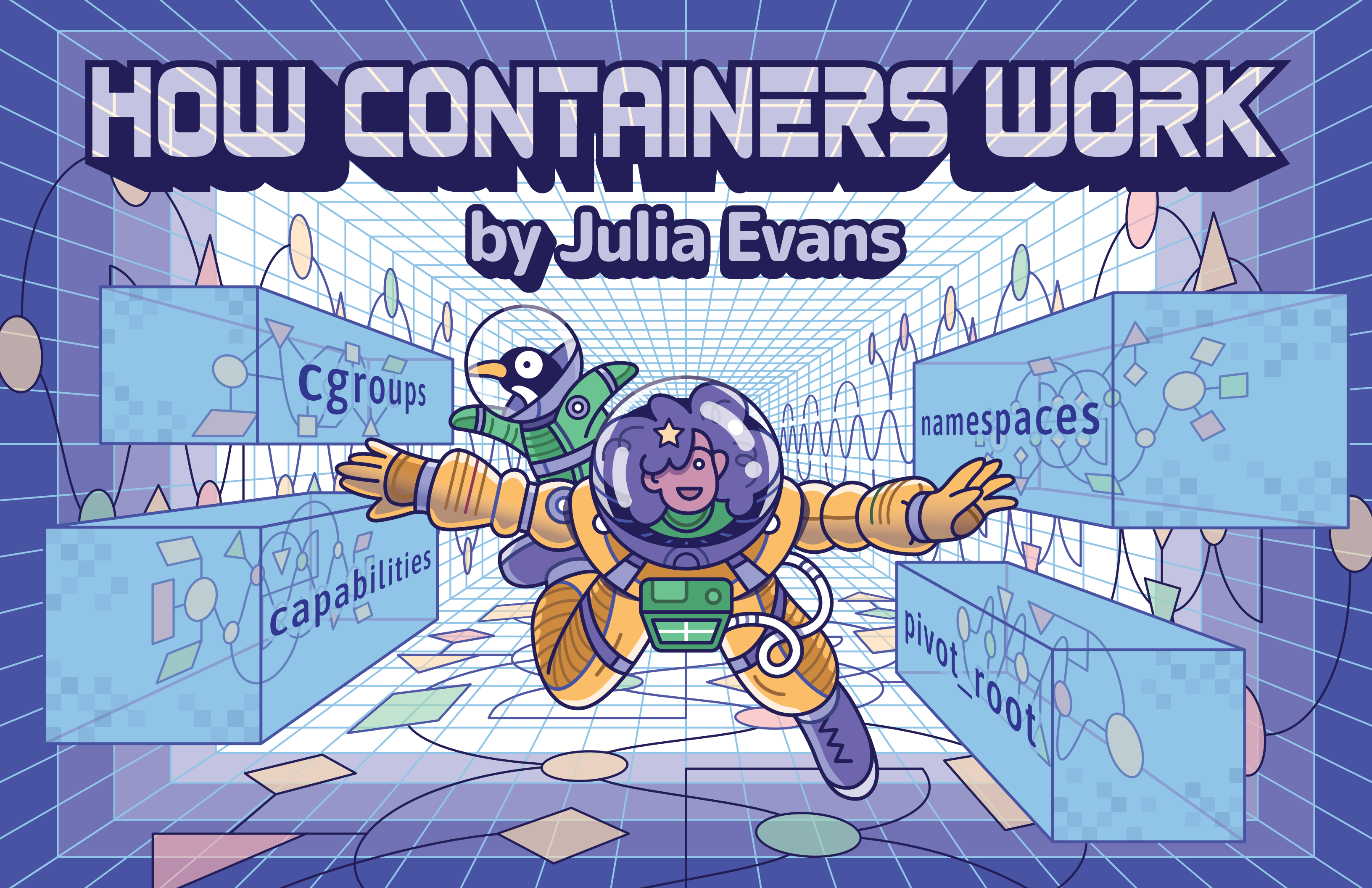
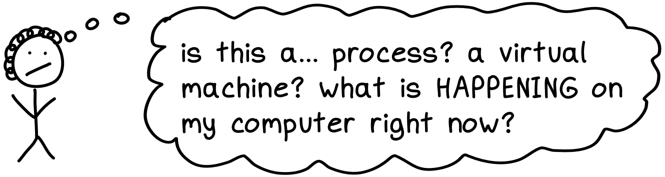
HOW CONTAINERS WORK

by Julia Evans

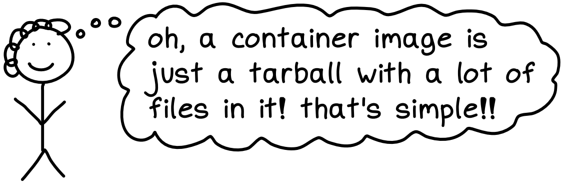


why this zine?

When I started using containers I was SO CONFUSED.


is this a... process? a virtual machine? what is HAPPENING on my computer right now?

So I decided to learn how they work under the hood!


oh, a container image is just a tarball with a lot of files in it! that's simple!!

Now I feel confident that I can solve basically any problem with containers because I understand how they work.
I hope that after reading this zine, you'll feel like that too.

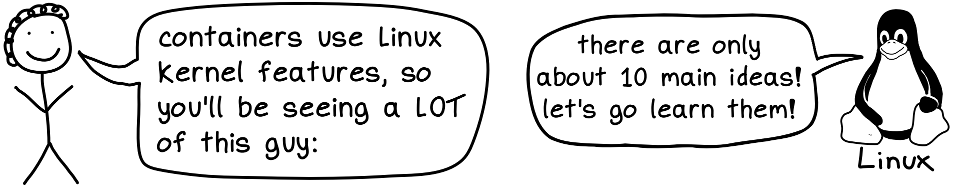
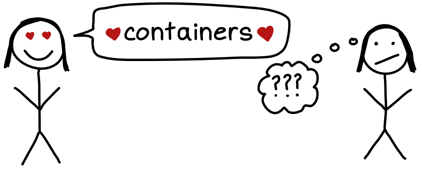

containers use Linux kernel features, so you'll be seeing a LOT of this guy:
there are only about 10 main ideas! let's go learn them!
Linux

table of contents

why containers?

there's a lot of container hype

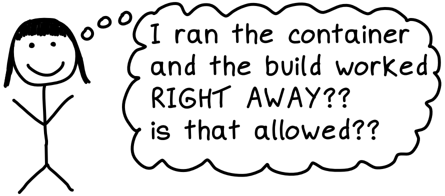


Here are 2 problems they solve...

problem: building software is annoying

```
$ ./configure
$ make all
ERROR: you have version
       2.1.1 and you need
       at least 2.2.4
```

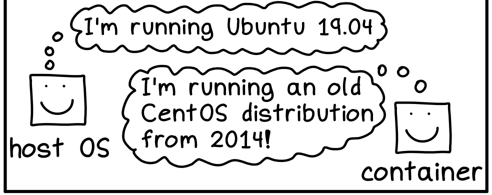
solution: package all dependencies in a ★container★


I ran the container and the build worked RIGHT AWAY?? is that allowed??

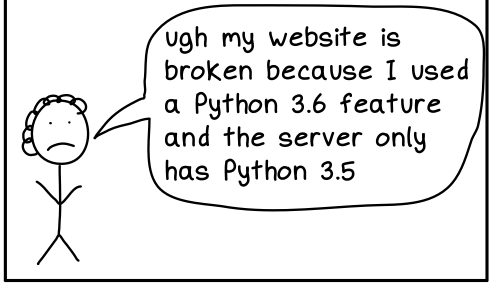
Many CI systems use containers.

containers have their own filesystem

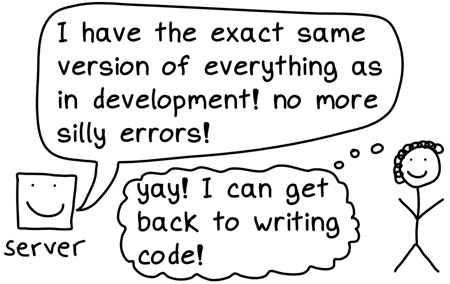
This is the big reason containers are great.



problem: deploying software is annoying too

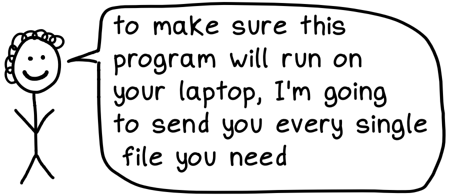

ugh my website is broken because I used a Python 3.6 feature and the server only has Python 3.5

solution: deploy a container


I have the exact same version of everything as in development! no more silly errors!
server
yay! I can get back to writing code!

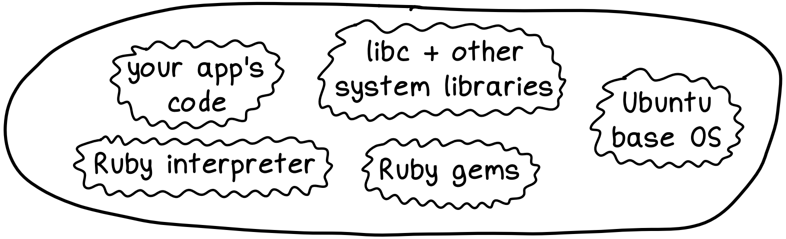
the big idea: include EVERY dependency

containers package EVERY dependency together


to make sure this program will run on your laptop, I'm going to send you every single file you need

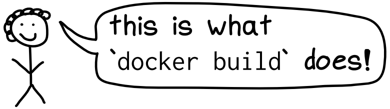
a container image is a tarball of a filesystem

Here's what's in a typical Rails app's container:



how images are built

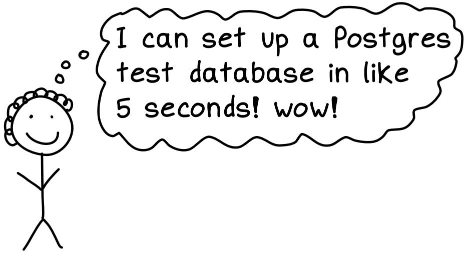
0. start with a base OS
1. install program + dependencies
2. configure it how you want
3. make a tarball of the WHOLE FILESYSTEM



running an image

1. download the tarball
2. unpack it into a directory
3. run a program and pretend that directory is its whole filesystem

images let you "install" programs really easily


I can set up a Postgres test database in like 5 seconds! wow!

containers aren't magic

These 15 lines of bash will start a container running the fish shell. Try it!

(download this script at bit.ly/containers-arent-magic)

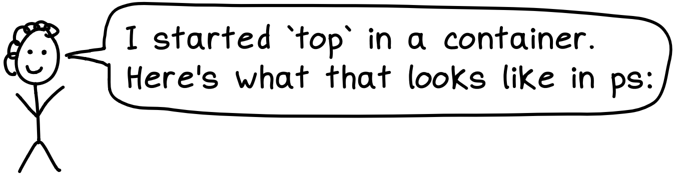
It only runs on Linux because these features are all Linux-only.

```
wget bit.ly/fish-container -O fish.tar       # 1. download the image
mkdir container-root; cd container-root
tar -xf ../fish.tar                          # 2. unpack image into a directory
cgroup_id="cgroup_$(shuf -i 1000-2000 -n 1)" # 3. generate random cgroup name
cgcreate -g "cpu,cpuacct,memory:$cgroup_id"  # 4. make a cgroup &
cgset -r cpu.shares=512 "$cgroup_id"              set CPU/memory limits
cgset -r memory.limit_in_bytes=1000000000 \  #
      "$cgroup_id"                           #
cgexec -g "cpu,cpuacct,memory:$cgroup_id" \  # 5. use the cgroup
    unshare -fmuipn --mount-proc \           # 6. make + use some namespaces
    chroot "$PWD" \                          # 7. change root directory
    /bin/sh -c "                             #
        /bin/mount -t proc proc /proc &&     # 8. use the right /proc
        hostname container-fun-times &&      # 9. change the hostname
        /usr/bin/fish"                       # 10. finally, start fish!
```

containers = processes

a container is a group of Linux processes

on a Mac, all your containers are actually running in a Linux virtual machine


I started `top` in a container. Here's what that looks like in ps:

outside the container

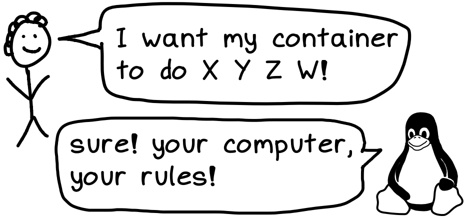
```
$ ps aux | grep top
USER PID START COMMAND
root 23540 20:55 top
bork 23546 20:57 top
```

inside the container

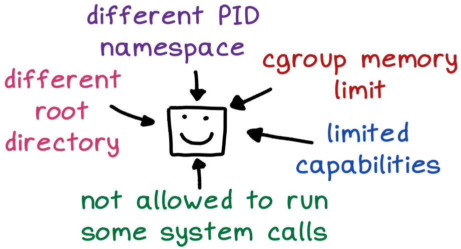
```
$ ps aux | grep top
USER PID START COMMAND
root 25 20:55 top
```

these two are the same process!

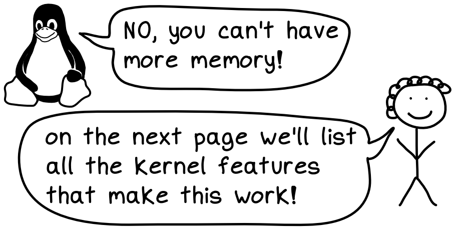
container processes can do anything a normal process can ...


I want my container to do X Y Z W!
sure! your computer, your rules!

... but usually they have 🔒 restrictions 🔒



the restrictions are enforced by the Linux kernel


NO, you can't have more memory!
on the next page we'll list all the kernel features that make this work!

container kernel features

containers use these Linux kernel features

"container" doesn't have a clear definition, but Docker containers use all of these features.

♥ pivot_root ♥

set a process's root directory to a directory with the contents of the container image

★ cgroups ★

limit memory/CPU usage for a group of processes

♥ namespaces ♥

allow processes to have their own:

- → network
- → PIDs
- → hostname
- → mounts
- → users
- + more

★ capabilities ★

security: give specific permissions

♥ seccomp-bpf ♥

security: prevent dangerous system calls

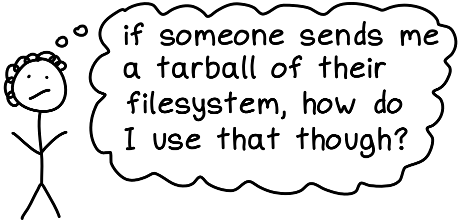
★ overlay filesystems ★

this is what makes layers work! Sharing layers saves disk space & helps containers start faster

pivot_root

a container image is a tarball of a filesystem

(or several tarballs: 1 per layer)


if someone sends me a tarball of their filesystem, how do I use that though?

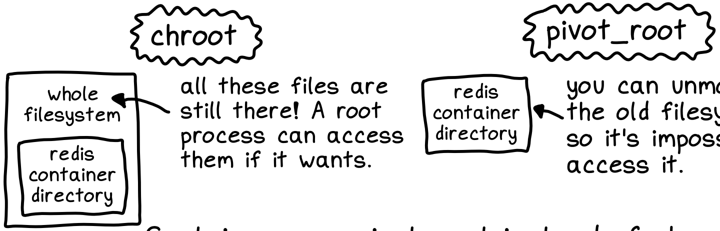
chroot: change a process's root directory

If you chroot to /fake/root, when it opens the file /usr/bin/redis it'll get /fake/root/usr/bin/redis instead.

You can "run" a container just by using chroot, like this:

```
$ mkdir redis; cd redis
$ tar -xzf redis.tar
$ chroot $PWD /usr/bin/redis
# done! redis is running!
```

programs can break out of a chroot



chroot: all these files are still there! A root process can access them if it wants.

pivot_root: you can unmount the old filesystem so it's impossible to access it.

Containers use pivot_root instead of chroot.

to have a "container" you need more than pivot_root

pivot_root alone won't let you:

→ set CPU/memory limits

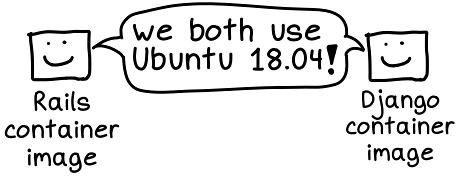
→ hide other running processes

→ use the same port as another process

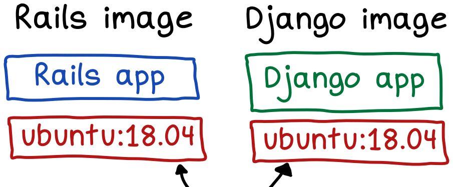
→ restrict dangerous system calls

layers

different images have similar files



reusing layers saves disk space



exact same files on disk!

a layer is a directory

```
$ ls 8891378eb*
bin/ home/ mnt/ run/ tmp/
boot/ lib/ opt sbin/ usr/
dev/ lib64/ proc/ srv/ var/
etc/ media/ root/ sys/
```

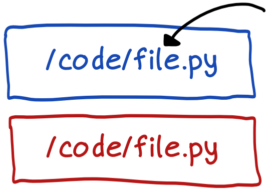
files in an ubuntu:18.04 layer

every layer has an ID

usually the ID is a sha256 hash of the layer's contents

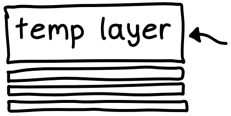
example: `8e99fae2..`

if a file is in 2 layers, you'll see the version from the top layer



this is the version you'll see in the merged image

by default, writes go to a temporary layer

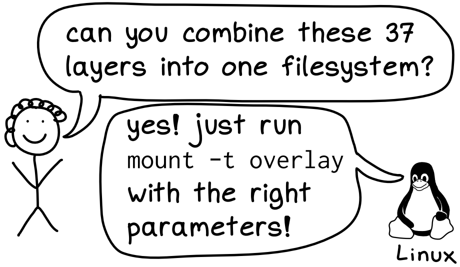


these files might be deleted after the container exits

To keep your changes, write to a directory that's mounted from outside the container

overlay filesystems

how layers work: mount -t overlay


can you combine these 37 layers into one filesystem?
yes! just run mount -t overlay with the right parameters!
Linux

mount -t overlay has 4 parameters

lowerdir:
list of read-only directories

upperdir:
directory where writes should go

workdir:
empty directory for internal use

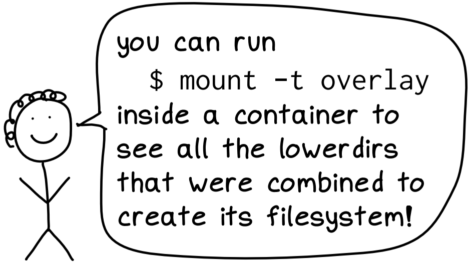
target:
the merged result

upperdir: where all writes go

when you create, change, or delete a file, it's recorded in the upperdir.

usually this starts out empty and is deleted when the container exits

lowerdir: the layers. read only.


you can run
$ mount -t overlay
inside a container to see all the lowerdirs that were combined to create its filesystem!

here's an example!

```
$ mount -t overlay overlay -o
    lowerdir=/lower,upperdir=/upper,workdir=/work /merged
$ ls /upper
cat.txt  dog.txt
$ ls /lower
dog.txt  bird.txt
$ ls /merged
cat.txt  dog.txt   bird.txt
```

the merged version of dog.txt is the one from the upper directory

container registries

sharing container images is useful

I made an image you can use to run Redis with just one command!
yay!

a registry is a server that serves images

images have an ID ← like "leff92"

and sometimes a tag

↑ like "18.04" or "latest"

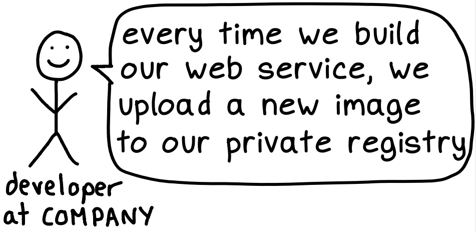
registries let you download just the layers you need

client
I already have the Ubuntu base image, I just need 0fe223
here's 0fe223!
registry

there are public container registries...

I'm going to use the latest official public Redis image to test my code!

... and private registries

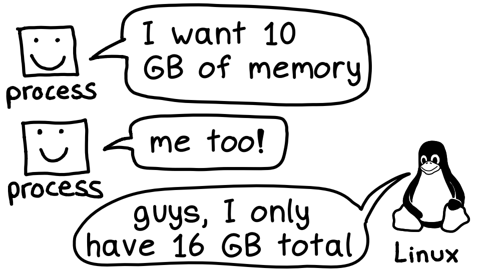

every time we build our web service, we upload a new image to our private registry
developer at COMPANY

be careful where your container images come from

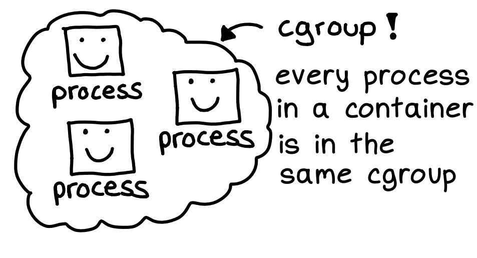
I'll just run this image from RANDOM_PERSON
... 2 months later ...
oh no! RANDOM_PERSON is mining bitcoin on my server

cgroups

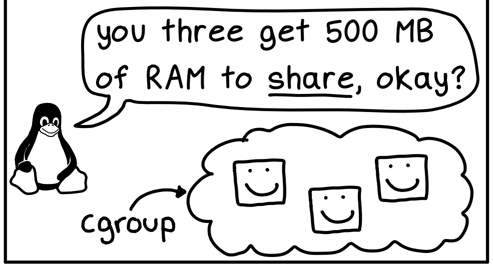
processes can use a lot of memory



a cgroup is a group of processes

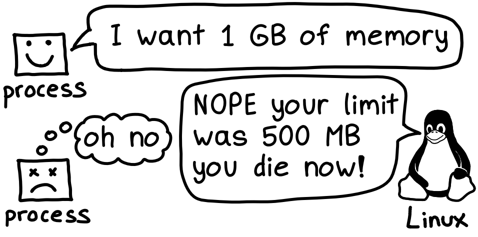


cgroups have memory/CPU limits

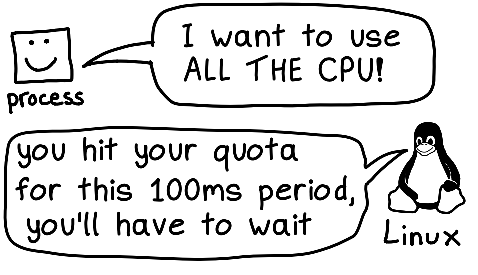


use too much memory: get OOM killed

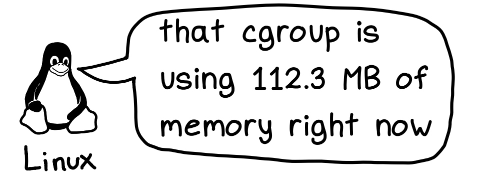
"out of memory"



use too much CPU: get slowed down


process
I want to use ALL THE CPU!
you hit your quota for this 100ms period, you'll have to wait
Linux

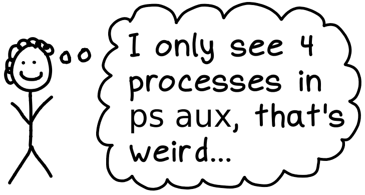
cgroups track memory & CPU usage



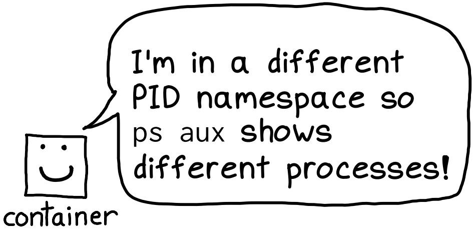
you can see it in /sys/fs/cgroup

namespaces

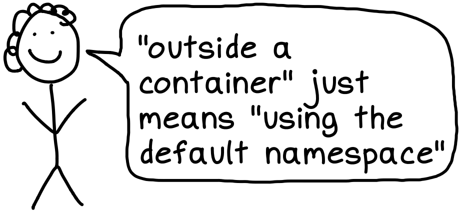
inside a container, things look different



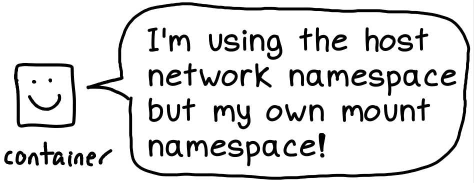
why things look different: namespaces


I'm in a different PID namespace so ps aux shows different processes!
container

there's a default ("host" namespace)


"outside a container" just means "using the default namespace"

processes can have any combination of namespaces


I'm using the host network namespace but my own mount namespace!
container

every processes has 7 namespaces

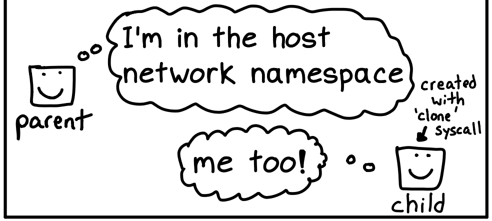
```
                PID
$ lsns -p 273
           NS TYPE
4026531835    cgroup
4026531836    pid
4026531837    user
4026531838    uts
4026531839    ipc
4026531840    mnt
4026532009    net
     namespace ID
```

you can also see a process's namespace with:

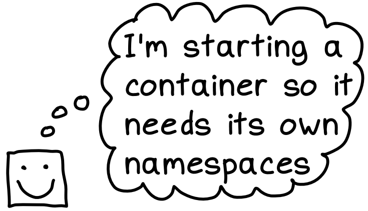
```
$ ls -l /proc/273/ns
```

how to make a namespace

processes use their parent's namespaces by default


I'm in the host network namespace
parent
me too!
created with 'clone' syscall
child

but you can switch namespaces at any time



command line tools

```
$ unshare --net COMMAND
```

run COMMAND in a new network namespace

```
$ sudo lsns
```

list all namespaces

```
$ nsenter -t PID --all COMMAND
```

run COMMAND in the same namespaces as PID

namespace system calls

★ clone ★

make a new process

★ unshare ★

make + use a namespace

★ setns ★

use an existing namespace

★clone★ lets you create new namespaces for a child process

clone(... CLONE_NEWNET)
parent
I have my own network namespace!
child

each namespace type has a ♥ man page ♥

```
$ man network_namespaces
...
A  physical network device
can live in exactly one
network namespace.
...
```

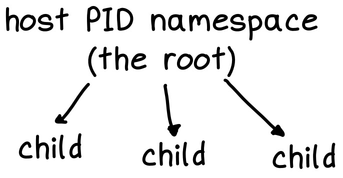
PID namespaces

the same process has different PIDs in different namespaces

PID in host	PID in container
23512	1
23513	4
23518	12

PID 1 is special

PID namespaces are in a tree


host PID namespace (the root)

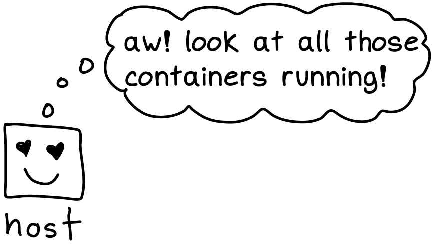
child

child

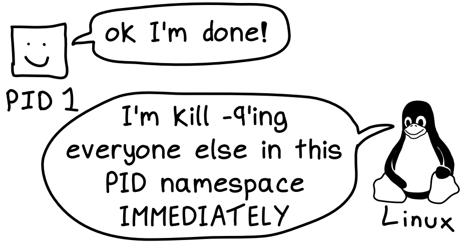
child

Often the tree is just 1 level deep (every child is a container)

you can see processes in child PID namespaces



if PID 1 exits, everyone gets killed


ok I'm done!
PID 1
I'm kill -9'ing everyone else in this PID namespace IMMEDIATELY
Linux

killing PID 1 accidentally would be bad

kill 1
container process
do you WANT everyone to die? I'm not gonna let you do that
Linux

rules for signaling PID 1

from same container:

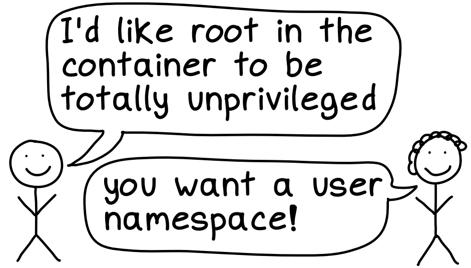
only works if the process has set a signal handler

from the host:

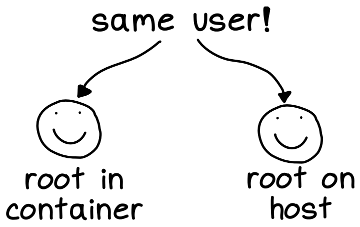
only SIGKILL and SIGSTOP are ok, or if there's a signal handler

user namespaces

user namespaces are a security feature...


I'd like root in the container to be totally unprivileged
you want a user namespace!

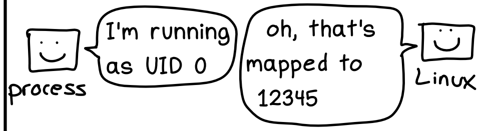
... but not all container runtimes use them



"root" doesn't always have admin access

I'm root so I can do ANYTHING right?
container process
actually you have limited capabilities so mostly you can just access files owned by root!

in a user namespace, UIDs are mapped to host UIDs



The mapping is in

/proc/self/uid_map

unmapped users show up as "nobody"

create user namespace

```
$ unshare --user bash
$ ls -l /usr/bin
.. nobody nogroup   apropos
.. nobody nogroup   apt
```

these are "actually" owned by root but we didn't map any users

how to find out if you have a separate user namespace

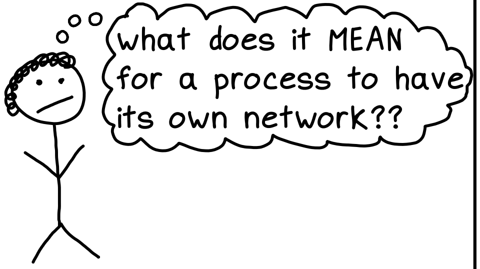
compare the results of

```
$ ls /proc/PID/ns
```

between a container process and a host process.

network namespaces

network namespaces are kinda confusing



namespaces usually have 2 interfaces

(+ sometimes more)

→ the loopback interface (127.0.0.1/8, for connections inside the namespace)

→ another interface (for connections from outside)

every server listens on a port and network interface(s)

0.0.0.0:8080

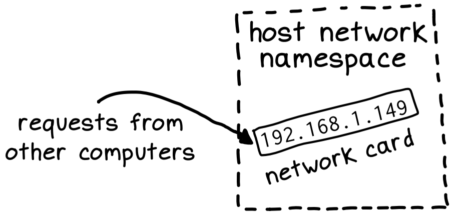
means

"port 8080 on every network interface in my namespace"

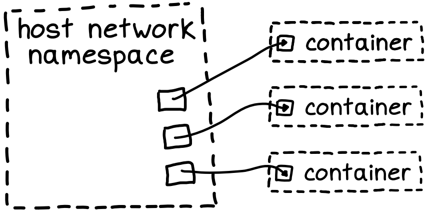
127.0.0.1 stays inside your namespace

I'm listening on 127.0.0.1
server
that's fine but nobody outside your network namespace will be able to make requests to you!

your physical network card is in the host network namespace

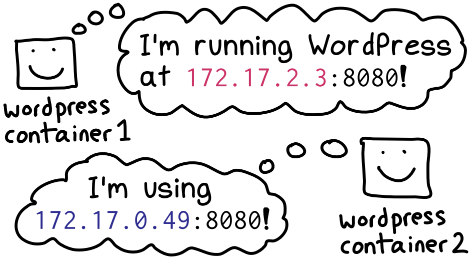


other namespaces are connected to the host namespace with a bridge



container IP addresses

containers often get their own IP address


I'm running WordPress at 172.17.2.3:8080!
wordpress container 1
I'm using 172.17.0.49:8080!
wordpress container 2

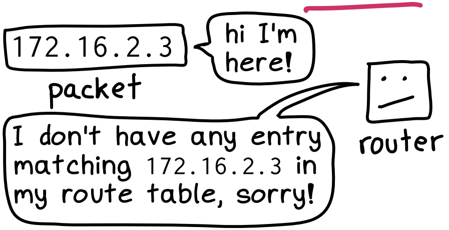
containers use private IP addresses

```
192.168.*.*
10.*.*.*
172.16.*.*
-> 172.32.*.*
```

reserved for private networks (RFC 1918)

This is because they're not directly on the public internet

for a packet to get to the right place, it needs a route


172.16.2.3
packet
hi I'm here!
router
I don't have any entry matching 172.16.2.3 in my route table, sorry!

inside the same computer, you'll have the right routes

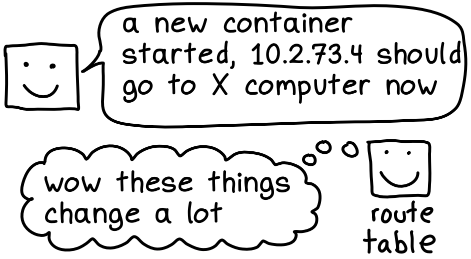
same computer:

```
  $ curl 172.16.2.3:8080
  <html>....
```

different computer:

```
  $ curl 172.16.2.3:8080
  .... no reply ....
```

distributing the right routes is complicated

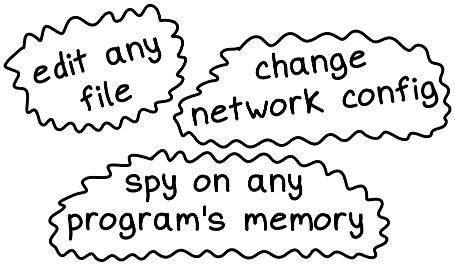

a new container started, 10.2.73.4 should go to X computer now
wow these things change a lot
route table

cloud providers have systems to make container IPs work

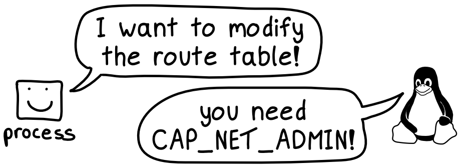
In AWS this is called an "elastic network interface"

capabilities

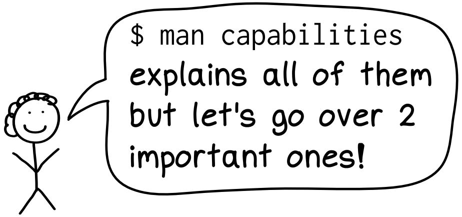
we think of root as being all-powerful...



... but actually to do "root" things, a process needs the right ★capabilities★



there are dozens of capabilities



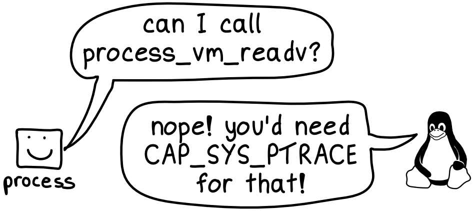
CAP_SYS_ADMIN

lets you do a LOT of things. avoid giving this if you can!

CAP_NET_ADMIN

allow changing network settings

by default containers have limited capabilities


can I call process_vm_readv?
process
nope! you'd need CAP_SYS_PTRACE for that!

$ getpcaps PID

print capabilities that PID has

getcap / setcap

system calls: get and set capabilities!

seccomp-bpf

all programs use system calls

read 2000 bytes from this file
program
here you go!
Linux

rarely-used system calls can help an attacker

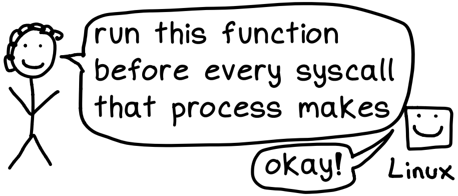
request_key

reboot

process_vm_readv

read memory from another process

seccomp-BPF lets you run a function before every system call



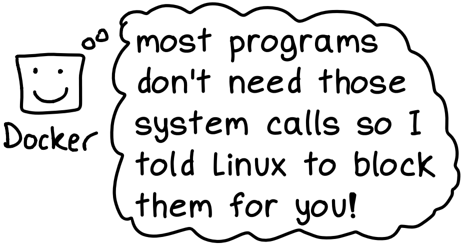
the function decides if that syscall is allowed

example function:

```
if name in allowed_list {
    return true;
}
return false;
```

this means the syscall doesn't happen!

Docker blocks dozens of syscalls by default

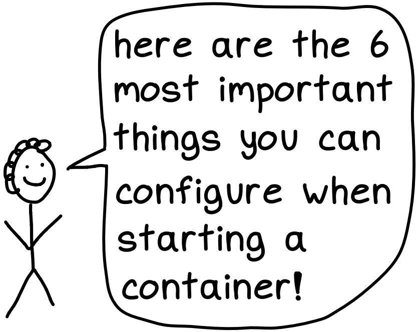

most programs don't need those system calls so I told Linux to block them for you!
Docker

2 ways to block scary system calls

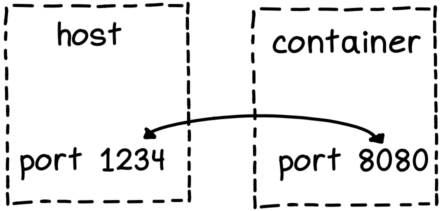
1. limit the container's capabilities
2. set a seccomp-bpf whitelist

You should do both!

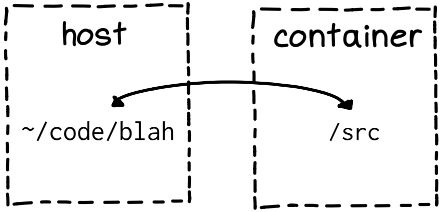
configuration options


here are the 6 most important things you can configure when starting a container!

map a port to the host



mount directories from the host



set capabilities

add seccomp-bpf filters

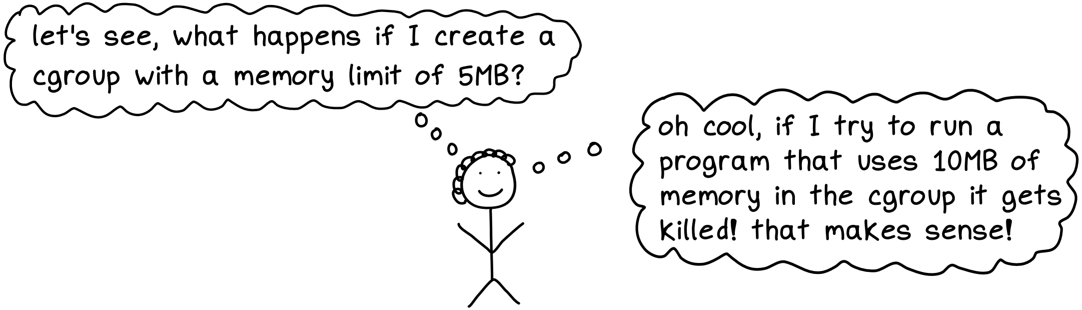
set memory and CPU limits

use the host network namespace

usually the default is to use a new network namespace!

♡ thanks for reading ♡

I did a bunch of the research for this zine by reading the man pages.
But, much more importantly, I experimented -- a lot!


let's see, what happens if I create a cgroup with a memory limit of 5MB?
oh cool, if I try to run a program that uses 10MB of memory in the cgroup it gets killed! that makes sense!

So, if you have access to a Linux machine, try things out!
Mount an overlay filesystem! Create a namespace! See what happens!

credits

Cover art: Vladimir Kašiković
Editing: Dolly Lanuza, Kamal Marhubi

♡ this?
more at
★ wizardzines.com ★