



o this?
more at
★ wizardzines.com ★

HOW CONTAINERS WORK

by Julia Evans

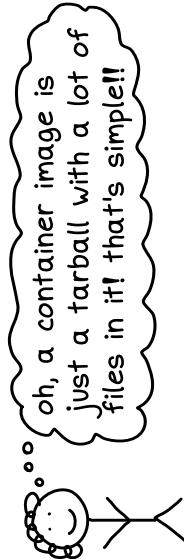
why this zine?

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!

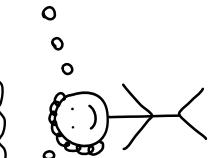
Linux

♡ thanks for reading ♡

23

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

Copy editing: Courtney Johnson

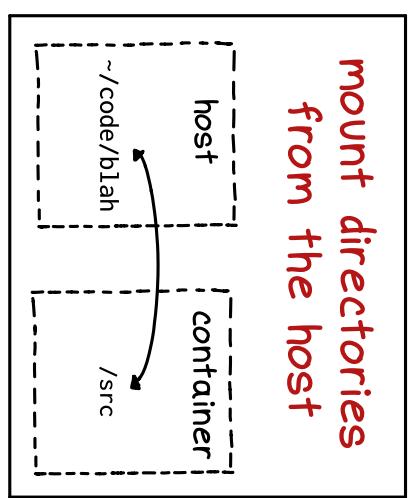
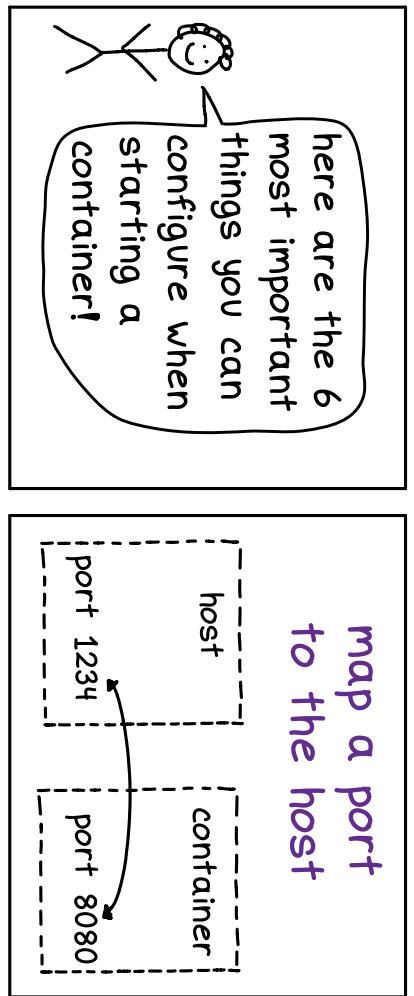
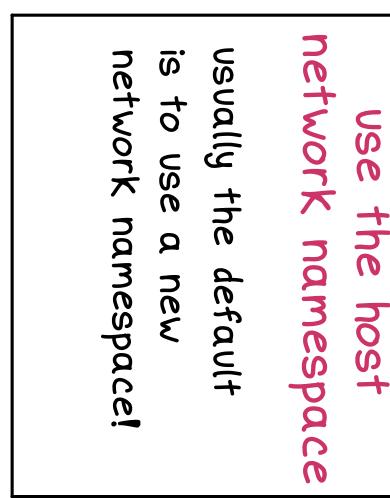
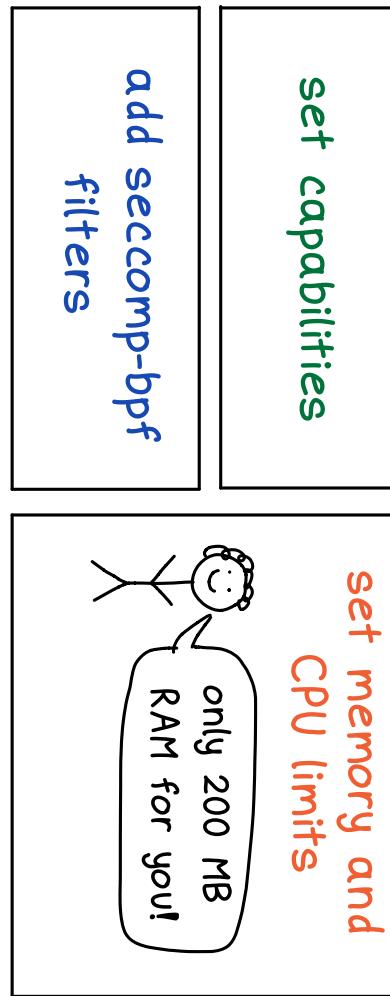
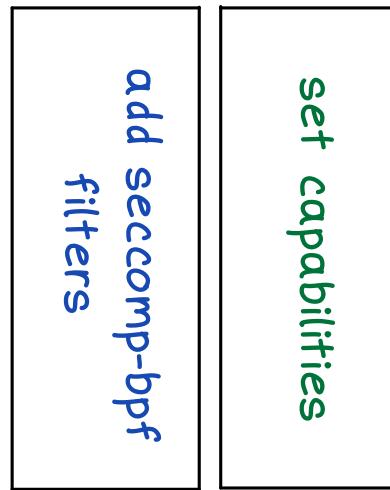
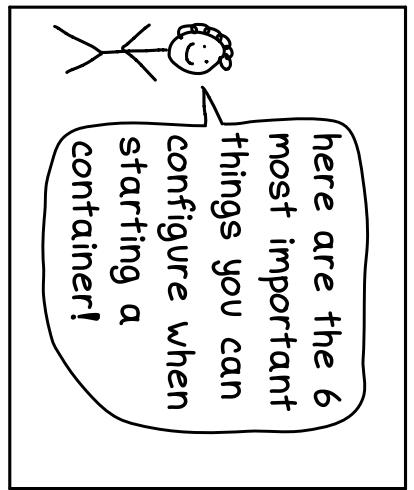
configuration options 22

22

configuration options

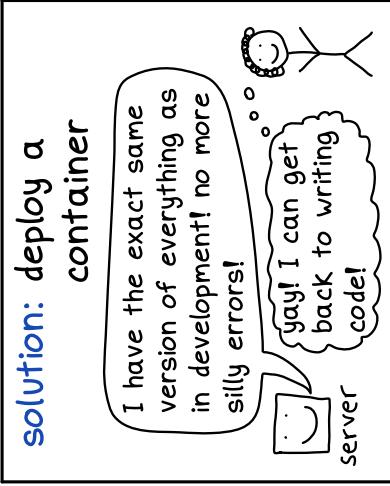
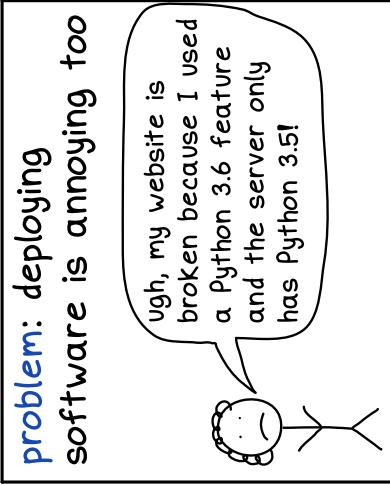
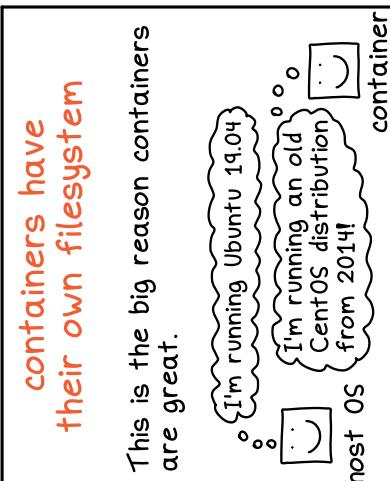
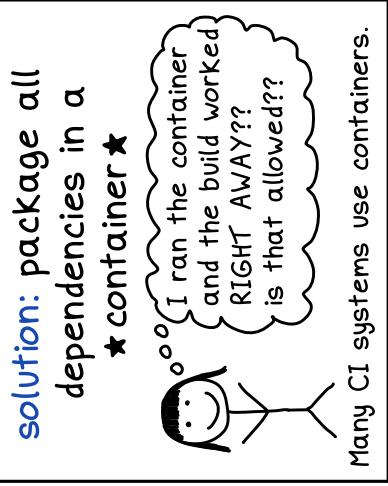
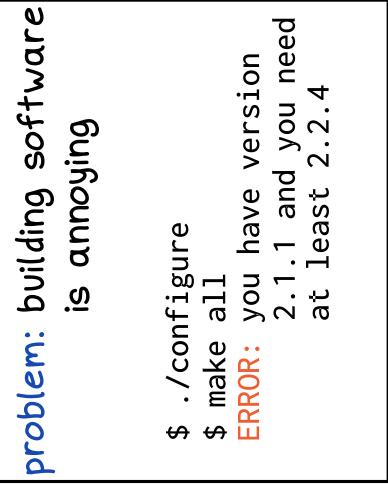
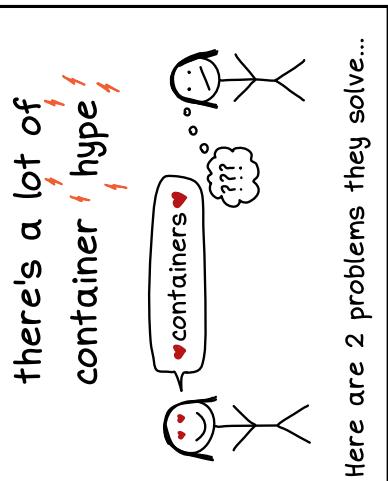
table of contents

why containers?	4
the big idea: include EVERY dependency	5
containers aren't magic	6
containers = processes	7
container kernel features	8
pivot_root	9
layers	10
overlay filesystems	11
container registries	12
capabilities	13
namespaces	14
how to make a namespace	15
PID namespaces	16
user namespaces	17
network namespaces	18
container IP addresses	19
capabilities	20
seccomp-BPF	21
configuration options	22



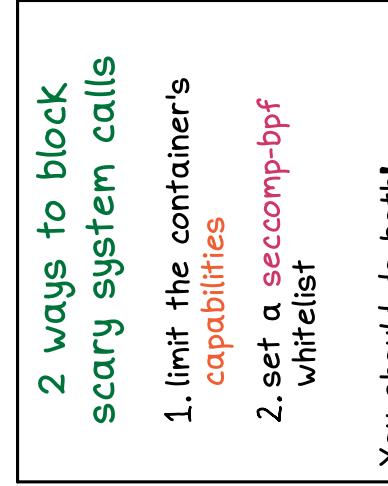
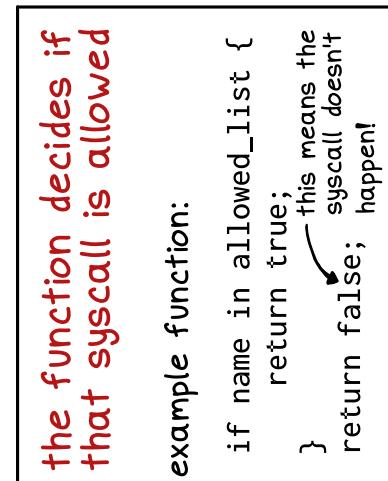
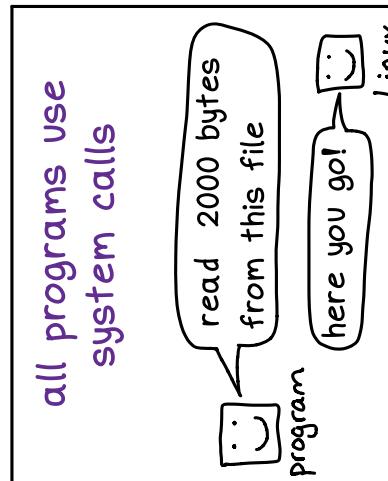
why containers?

4



seccomp-bpf

21



capabilities

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

edit any file
change network config
spy on any program's memory

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

★capabilities★

I want to modify the route table!
you need CAP_NET_ADMIN!

\$ man capabilities
explains all of them.
But let's go over 2 important ones!

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_vmlinux?
nope! you'd need CAP_SYS_PTRACE for that!

\$ getpcaps PID

print capabilities that PID has

getcap / setcap

system calls:
get and set capabilities!

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:

your app's code
libc + other system libraries
Ubuntu base OS
Ruby interpreter
Ruby gems

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

this is what 'docker build' does!

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

6

These 15 lines of bash will start a container running the fish shell. Try it!
(download this script at bit.ly/containers-arent-magic)

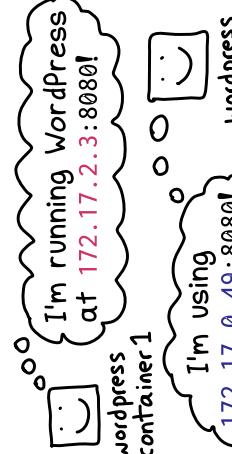
It only runs on Linux because **these features are all Linux-only**.

```
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 "$cgroup_id"
cgset -r memory.limit_in_bytes=1000000000 \      # set CPU/memory limits
"$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 &&
hostname container-fun-times &&
/usr/bin/fish"                                  # 8. use the right /proc
                                                # 9. change the hostname
                                                # 10. finally, start fish!
```

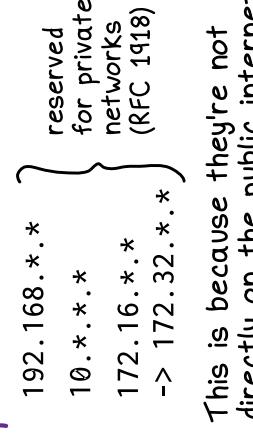
Container IP addresses

Containers often get their own IP address



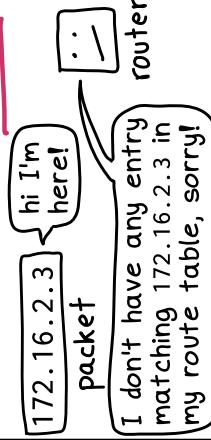
Container IP addresses

Containers use private IP addresses



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

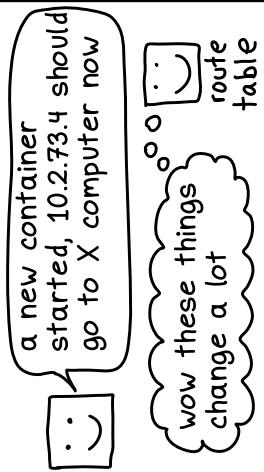
for a packet to get to the right place, it needs a route



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



In AWS, this is called an "elastic network interface"

network namespaces

18

containers = processes

7

a container is a group of Linux processes

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

127.0.0.1 stays inside your namespace

I'm listening on 127.0.0.1

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

requests from other computers

192.168.1.149 network card

network namespaces are kinda confusing

what does it MEAN for a process to have its own network??

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

I want my container to do X Y Z W!

... but usually they have restrictions 🔒

different PID
namespace
root directory
different memory limit
cgrouplimit
not allowed to run some system calls

other namespaces are connected to the host namespace with a bridge

inside the container

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

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)

0.0.0.0:8080 means "port 8080 on every network interface in my namespace"

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!

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"

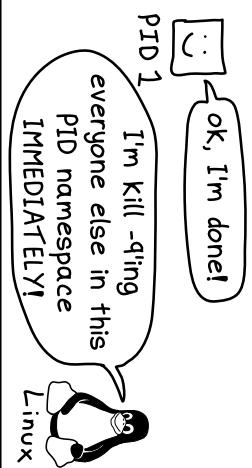
PID namespaces

16

the same process has different PIDs in different namespaces

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

**if PID 1 exits,
everyone gets killed**



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)

Killing PID 1 accidentally would be bad



you can see processes in child PID namespaces
aw! look at all those containers running!

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?

programs can break out of a chroot

chroot

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

redis
container directory

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!
```

to have a "container" you need more than pivot_root

9

Containers use pivot_root instead of chroot.

redis
container directory

chroot

redis
container directory

pivot_root

- set CPU/memory limits
- hide other running processes
- use the same port as another process
- restrict dangerous system calls

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

layers

10

different images have similar files

we both use Ubuntu 18.04!
 Rails container image
 Django container image

reusing layers saves disk space

Rails image Django image
Rails app Django app
Ubuntu:18.04 Ubuntu:18.04
exact same files on disk!

\$ 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
/code/file.py /code/file.py

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

by default, writes go to a temporary layer
temp layer → might be deleted after the container exits
To keep your changes, write to a directory that's mounted from outside the container

how to make a namespace

processes use their parent's namespaces by default

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

but you can switch namespaces at any time

I'm starting a container so it needs its own namespaces!
parent child

15

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

each namespace type has a man page!

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

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

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

namespace system calls

★clone★ make a new process
★unshare★ make + use a namespace
★sets★ use an existing namespace

namespaces

14

inside a container,
things look different

I only see 4
processes in
ps aux, that's
weird...

there's a default
("host" namespace)

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

why things look different:
; namespaces:

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

processes can have
any combination
of namespaces

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

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!

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

the layers. read only.

here's an example!

```
$ mount -t overlay overlay -o
    lowerdir=/lower,upperdir=/upper,workdir=/work /merged
```

```
$ ls /upper
cat.txt dog.txt
```

1s /lower

```
dog.txt bird.txt
$ ls /merged
cat.txt dog.txt
```

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

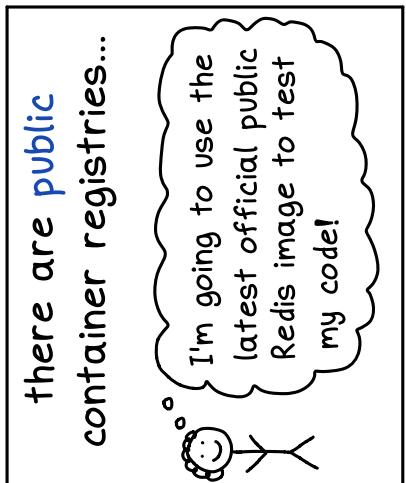
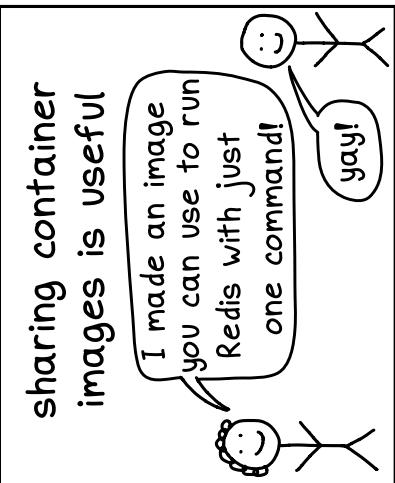
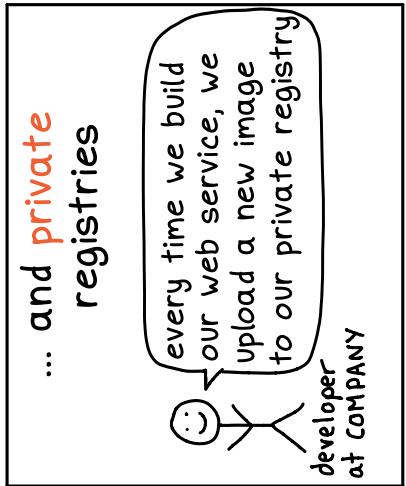
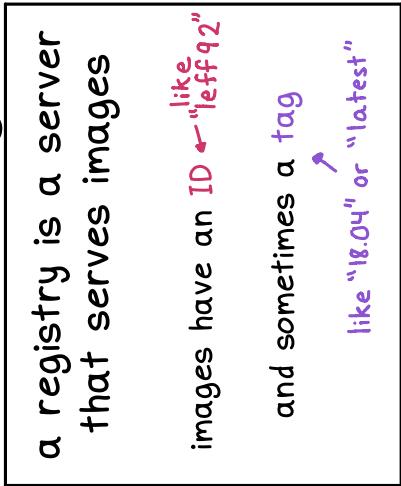
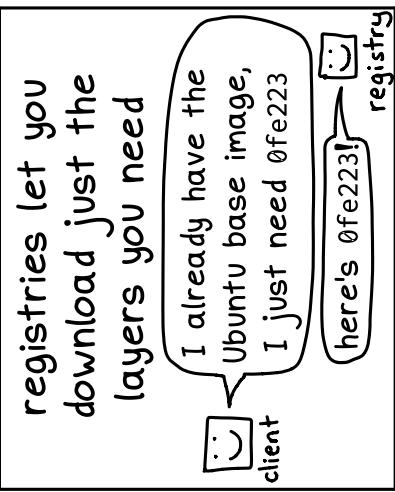
every process has
7 namespaces

	PID	NS	TYPE
4026531835		cgroup	
4026531836		pid	
4026531837		uts	user
4026531838		ipc	
4026531840		mnt	
4026532009	↑	net	

you can also see a
process's namespace with:
\$ ls -l /proc/273/ns

container registries

12



13

cgroups

