



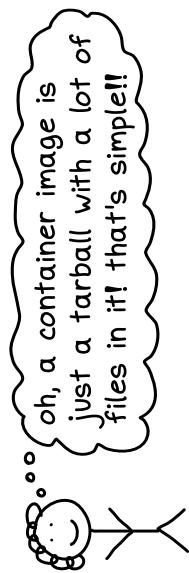
☞ this?  
more at  
★ wizardzines.com ★

# why this zine?

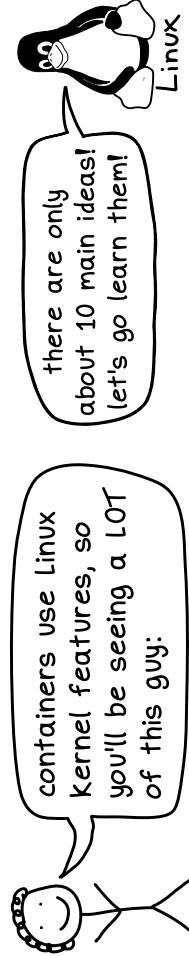
When I started using containers I was SO CONFUSED.



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



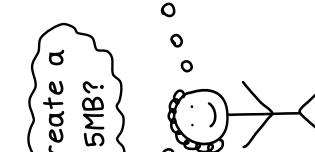
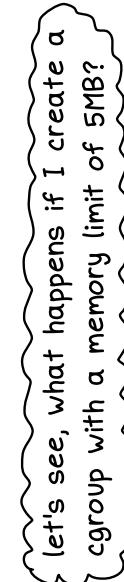
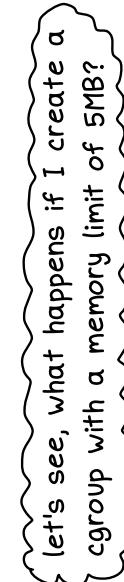
Now I feel confident that I can solve basically any problem with containers because I understand how they work.  
I hope that after reading this zine, you'll feel like that too.



## ♡ 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!



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

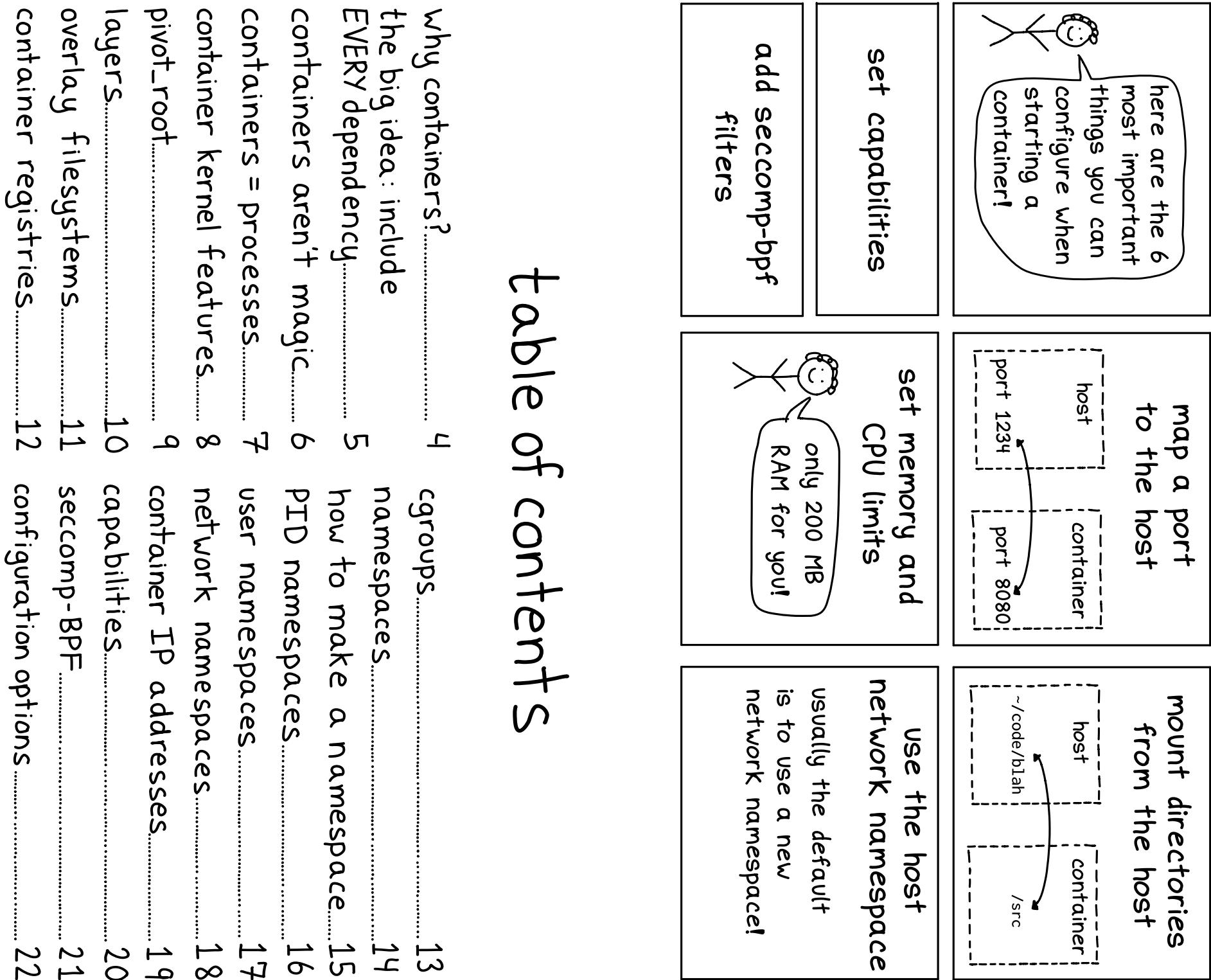
Copy editing: Courtney Johnson

# Configuration options

22

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

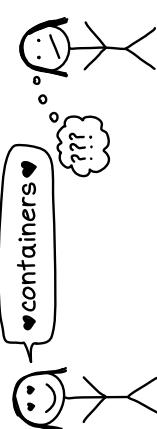
## table of contents



# why containers?

4

there's a lot of container 'hype'!



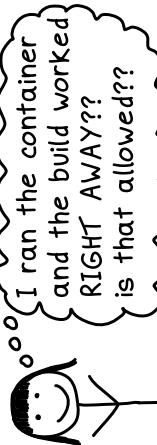
Here are 2 problems they solve...

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 ★



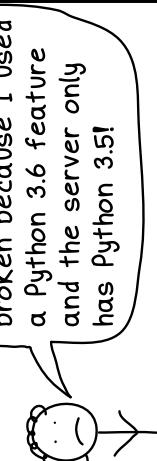
I ran the container and the build worked RIGHT AWAY?? is that allowed?? Many CI systems use containers.

containers have their own filesystem

This is the big reason containers are great.

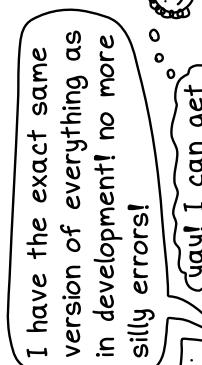
I'm running Ubuntu 19.04  
I'm running an old CentOS distribution from 2014!  
host OS container

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!

yay! I can get back to writing code!

21

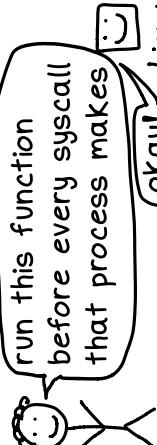
# seccomp-bpf

all programs use system calls

read 2000 bytes from this file

program here you go! Linux

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



run this function before every syscall that process makes

Okay! Linux!

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!

2 ways to block scary system calls

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

You should do both!

# capabilities

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

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

"root" things, a process needs the right

★capabilities★

I want to modify the route table!  
process

you need CAP\_NET\_ADMIN!

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!

there are dozens of capabilities

\$ man capabilities explains all of them.

But let's go over 2 important ones!

## the big idea: include EVERY dependency 5

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  
Ruby interpreter  
Ruby gems  
libc + other system libraries  
Ubuntu base OS

### 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

this is what 'docker build' does!

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
cgroup_id="cgroup_$(shuf -i 1000-2000 -n 1)" # 2. unpack image into a directory
cgcreate -g "cpu,cpuacct,memory:$cgroup_id" # 3. generate random cgroup name
cgset -r cpu.shares=512 "$cgroup_id"
cgset -r memory.limit_in_bytes=10000000000 \ # 4. make a cgroup &
                                             # set CPU/memory limits
                                             # "$cgroup_id"
cexec -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!
```

## 19 Container IP addresses

Containers often get their own IP address

wordpress container1: I'm running WordPress at 172.17.2.3:8080!  
wordpress container2: I'm using 172.17.0.49:8080!

Containers use private IP addresses

192.168.\*.\* } reserved for private networks (RFC 1918)  
10.\*.\*.\* }  
172.16.\*.\* } -> 172.32.\*.\*  
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 hi I'm here!

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

a new container started, 10.2.73.4 should go to X computer now

wow these things change a lot

cloud providers have systems to make container IPs work

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

route table

# network namespaces

18

network namespaces are kinda confusing

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

I'm listening on 127.0.1

that's fine, but nobody outside your network namespace will be able to make requests to you!

**127.0.0.1 stays inside your namespace**

**127.0.0.1 stays inside your namespace**

a container is a group of Linux processes

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

**your physical network card is in the host network namespace**

**your physical network card is in the host network namespace**

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 ↗**

**other namespaces are connected to the host namespace with a bridge**

"port 8080 on every network interface in my namespace" means "port 8080 on every port and network interface(s)"

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 are the same process!

**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!

## containers = processes

7

# container kernel features

8

## 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  
only 500 MB of RAM for you!



### namespaces ♥

allow processes to have their own:

- network → mounts
- PIDs → users
- hostname + 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

|7

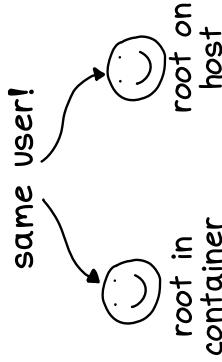
# User namespaces

## User namespaces are a security feature...

I'd like root in the container to be totally unprivileged

you want a user namespace!

same user!

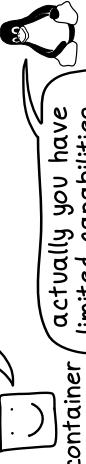


... but not all container runtimes use them

## "root" doesn't always have admin access

I'm root so I can do ANYTHING right?

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

oh, that's mapped to 12345  
I'm running as UID 0

The mapping is in /proc/self/uid\_map :

nobody nogroup apt  
... nobody nogroup apropos  
nobody nogroup apt  
... these are "actually" owned by root but we didn't map any users

## unmapped users show up as "nobody"

create user namespace  
\$ unshare --user bash

\$ ls -l /usr/bin

... nobody nogroup apropos  
nobody nogroup apt  
... 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

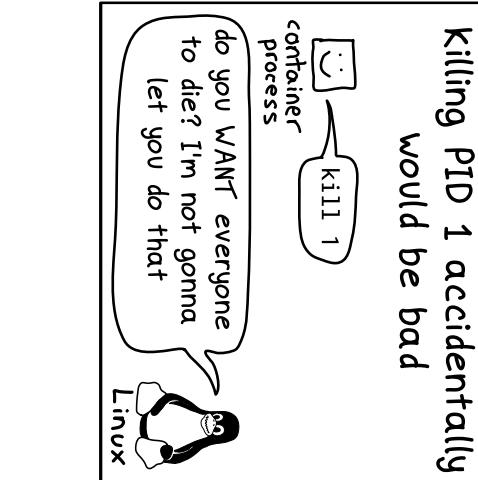
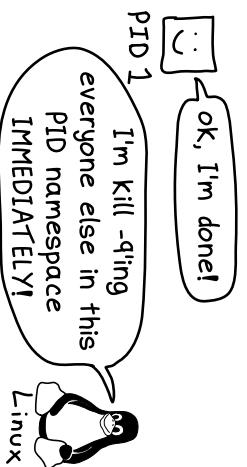
compare the results of  
\$ ls /proc/PID/ns  
between a container process and a host process

# PID namespaces

the same process has different PIDs in different namespaces

PID in host	PID in container
23512	①
23513	4 PID 1 is
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)

aw! look at all those containers running!



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

## 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

redis container directory

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

pivot\_root

redis container directory

so it's impossible to access it.

9

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

Containers use pivot\_root instead of chroot.

# layers

10

different images  
have similar files

 we both use Ubuntu 18.04!  
 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!

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  
 /code/file.py  
 /code/file.py  
this is the  
version you'll  
see in the  
merged image

15

## how to make a namespace

processes use their  
parent's namespaces  
by default

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

but you can switch  
namespaces at any time

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


namespace  
system calls

clone ★

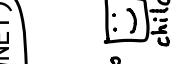
make a new process

★ unshare ★

make + use a namespace

sets ★

use an existing namespace

★ clone( ... CLONE\_NEWNET )  
parent  
I have my own  network namespace!

each namespace  
type has a  
man page ♥

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

# namespaces

14

inside a container,  
things look different

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

why things look different:  
because they're in  
different namespaces!

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

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!

## 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!

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
```

the merged version of dog.txt is

the one from the upper directory

```
$ ls /merged
```

```
cat.txt dog.txt bird.txt
```

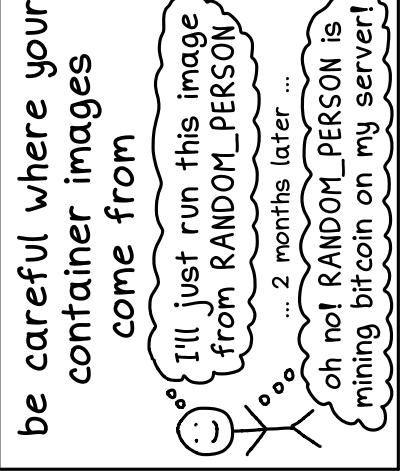
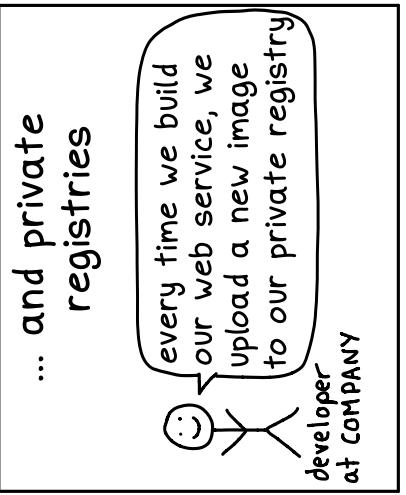
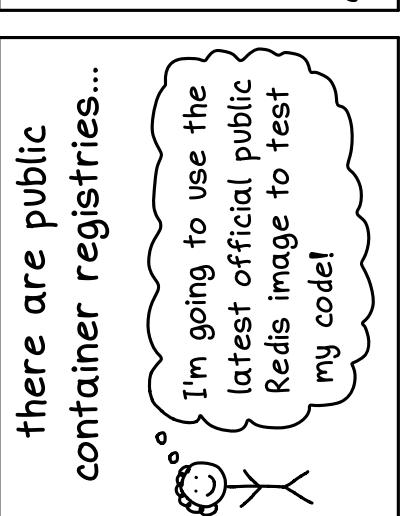
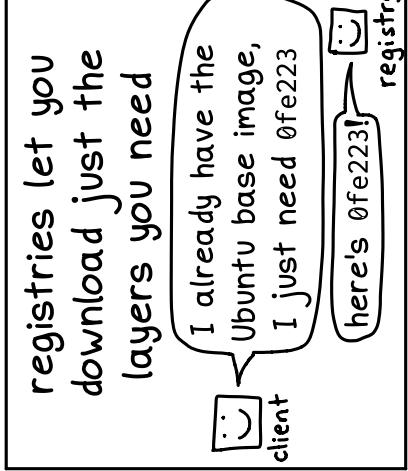
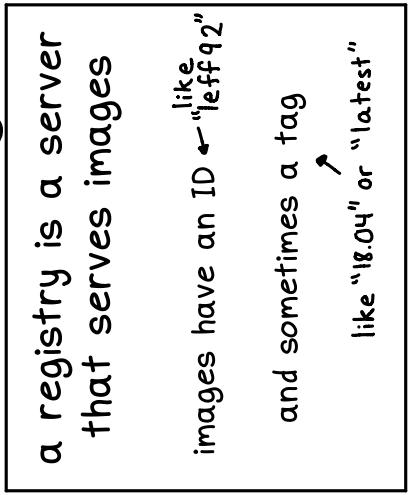
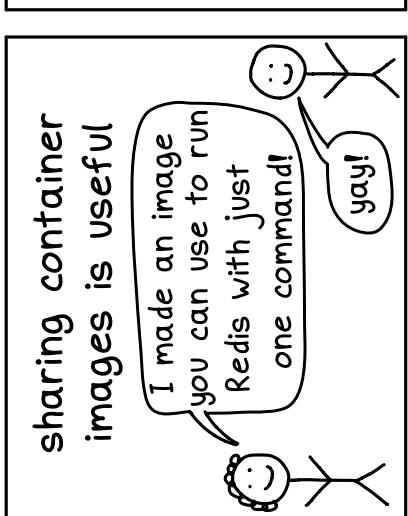
every process has  
7 namespaces

PID	NS TYPE
4026531835	cgroup
4026531836	pid
4026531837	uts
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

