O'REILLY

oscon

OPEN SOURCE CONVENTION

PORTLAND, OR
JULY 21-24, 2014

#oscon

# Is it safe to run applications in Linux Containers?

Jérôme Petazzoni
@jpetazzo

Docker Inc.
@docker

# Yes

```
/* shocker: docker PoC VMM-container breakout (C) 2014 Sebastian Krahmer
 *
 * Demonstrates that any given docker image someone is asking
 * you to run in your docker setup can access ANY file on your host,
 * e.g. dumping hosts /etc/shadow or other sensitive info, compromising
 * security of the host and any other docker VM's on it.
 *
 * docker using container based VMM: Sebarate pid and net namespace,
 * stripped caps and RO bind mounts into container's /. However
 * as its only a bind-mount the fs struct from the task is shared
 * with the host which allows to open files by file handles
 * (open_by_handle_at()). As we thankfully have dac_override and
 * dac_read_search we can do this. The handle is usually a 64bit
 * string with 32bit inodenumber inside (tested with ext4).
 * Inode of / is always 2, so we have a starting point to walk
 * the FS path and brute force the remaining 32bit until we find the
 * desired file (It's probably easier, depending on the fhandle export
 * function used for the FS in question: it could be a parent inode# or
 * the inode generation which can be obtained via an ioctl).
 * [In practise the remaining 32bit are all 0 :]
```

# Wait

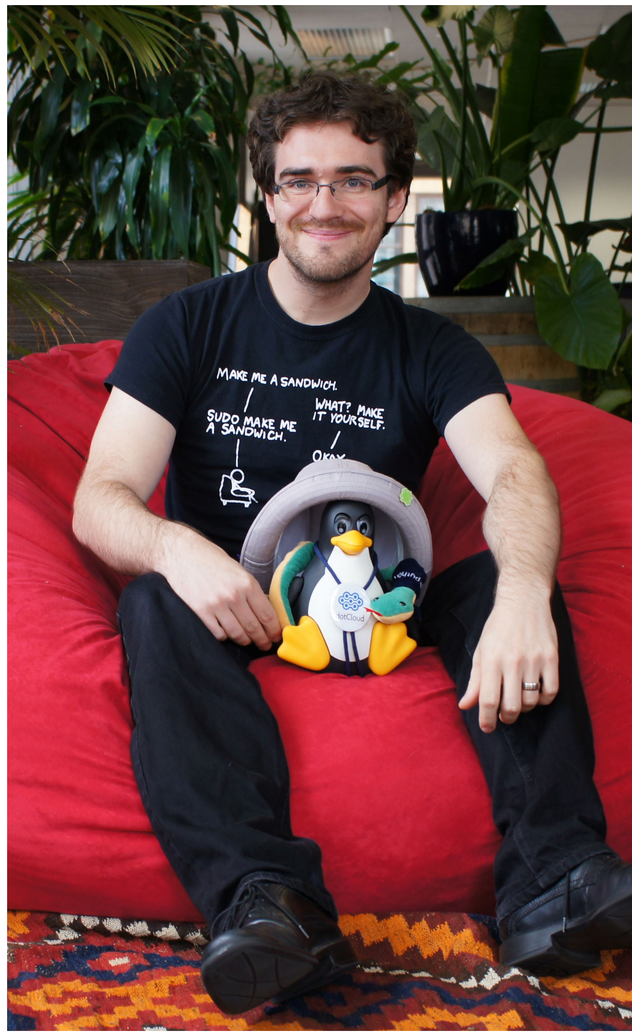
# No!

Docker has changed its security status to

**It's complicated**

# Who am I? Why am I here?

- Jérôme Petazzoni (@jpetazzo)
  - Grumpy French Linux DevOps
- Operated dotCloud PAAS for 3+ years
  - hosts arbitrary code for arbitrary users
  - each service of each app runs in a container
  - no major security issue yet (fingers crossed)
- Containerize all the things!
  - VPN-in-Docker, KVM-in-Docker, Xorg-in-Docker, Docker-in-Docker...



# What are those “containers” ? (1/3)

- Technically: ~chroot on steroids
  - a container is a set of processes
    (running on top of common kernel)
  - isolated* from the rest of the machine
    (cannot see/affect/harm host or other containers)
  - using *namespaces* to have private view of the system
    (network interfaces, PID tree, mountpoints...)
  - and *cgroups* to have metered/limited/reserved resources
    (to mitigate “bad neighbor” effect)

**Limitations may apply.*

# What are those “containers” ? (2/3)

- From a distance: looks like a VM
  - I can SSH into my container
  - I can have root access in it
  - I can install packages in it
  - I have my own eth0 interface
  - I can tweak routing table, iptables rules
  - I can mount filesystems
  - etc.

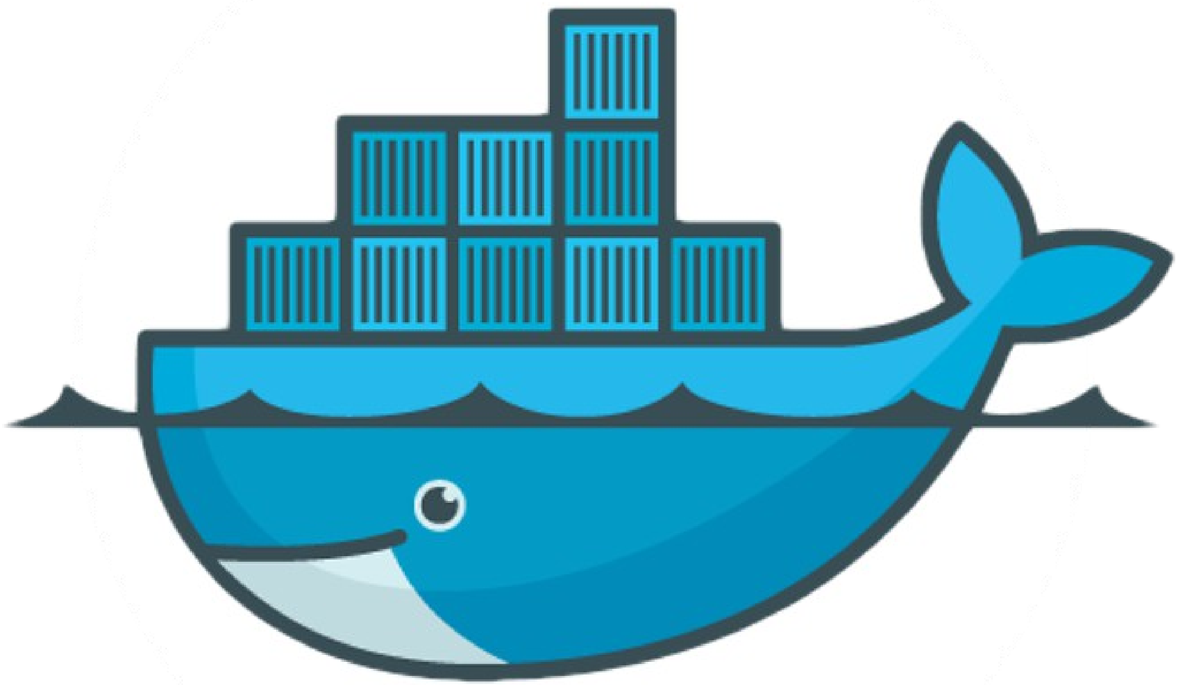
# What are those “containers” ? (3/3)

- Lightweight, fast, disposable... virtual environments
  - boot in milliseconds
  - just a few MB of intrinsic disk/memory usage
  - bare metal performance is possible
- The new way to build, ship, deploy, run your apps!

# Why is this a hot topic?

- Containers have been around for *decades*
- LXC (Linux Containers) have been around for *years*

So, what?



# Blame Docker

# Why is this a hot topic?

- Containers have been around for *decades*
- LXC (Linux Containers) have been around for *years*
- Tools like Docker made containers very easy to use
- Everybody* wants to deploy containers now
- But, oops, LXC wasn't made for security
- We want containers, and we want them now; how can we do that *safely?*

*Okay, *almost* everybody. If you're not convinced yet of the general awesomeness of Docker and containers, give them a second chance by checking my other talk this afternoon; or the Docker meet-up tomorrow at New Relic.

# Some inspirational quotes

# “LXC is not yet secure. If I want real security I will use KVM.”

—Dan Berrangé
(famous LXC hacker)

This was in 2011. The Linux Kernel has changed a  tiny little bit since then.

“From security point of view lxc is terrible and may not be consider as security solution.”

—someone on Reddit
(original spelling and grammar)

Common opinion among security experts and paranoid people.
To be fair, they have to play safe & can't take risks.

“Basically containers are not functional as security containers at present, in that **if you have root on a container you have root on the whole box.**”

—Gentoo Wiki

That's just plain false, or misleading, and we'll see why.

# “Containers do not contain.”

—Dan Walsh
(Mr SELinux)

This was earlier this year, and this guy knows what he's talking about.
Are we in trouble?

“For the fashion of Minas Tirith was such that it was built on seven levels, each delved into a hill, and about each was set a wall, and in each wall was a gate.”

—J.R.R. Tolkien

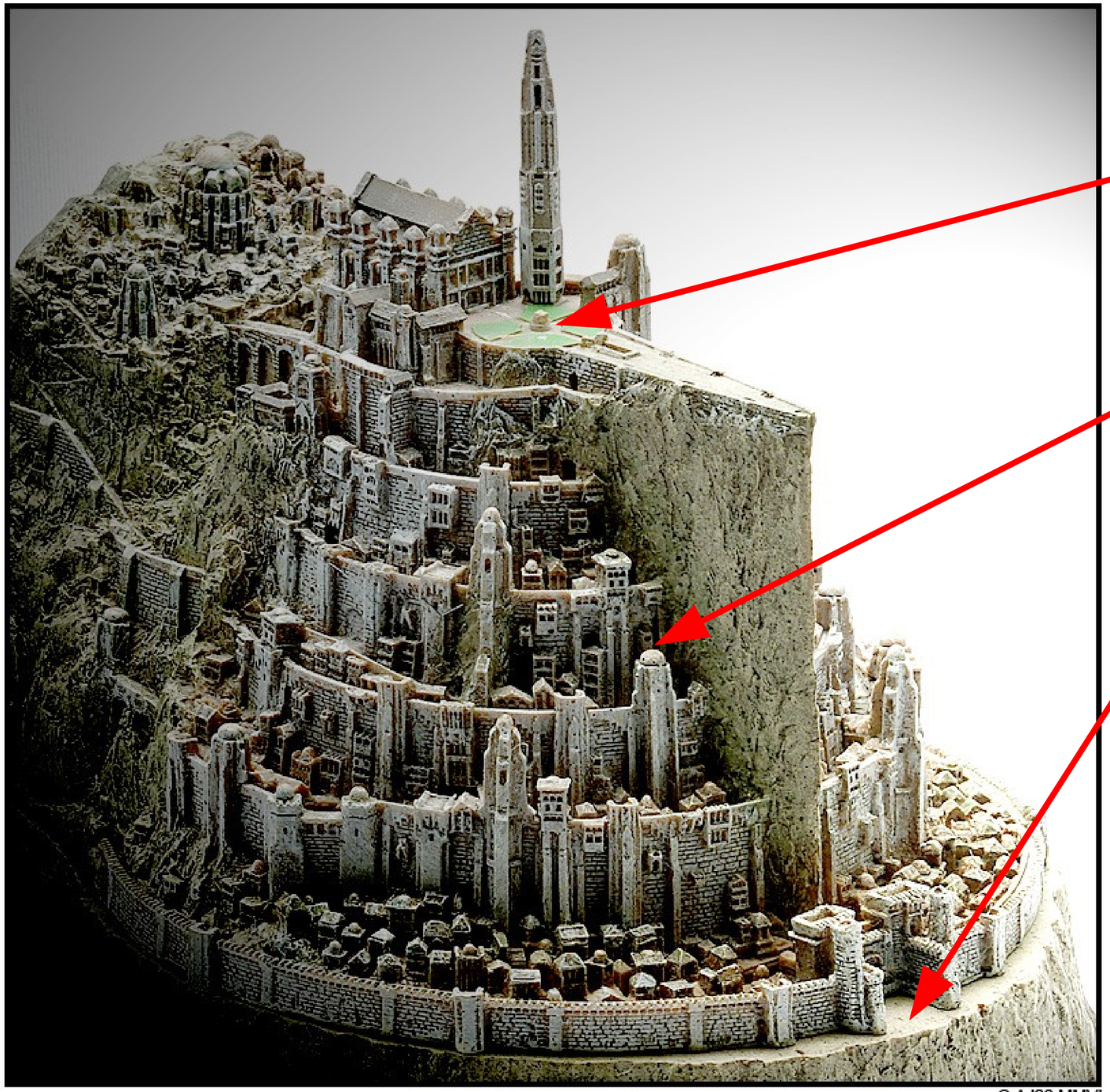
(also quoted in VAX/VMS Internals and Data Structures, ca. 1980)

# Keyword:
# *level**s***

# Let's revisit one of those quotes...

"If you have root on a container you have root on the whole box."

- Don't give root in the container in the first place
- If you really have to give root, give looks-like-root
- If that's not enough, give root but build another wall




Root in the host

Root in the container

Uruks (intruders)

# There is more than one threat model

- Regular applications
  - web servers, databases, caches, message queues, ...
- System services (high level)
  - logging, remote access, periodic command execution, ...
- System services (low level)
  - manage physical devices, networking, filesystems, ...
- Kernel
  - security policies, drivers, ...
- The special case of immutable infrastructure

# Regular applications

# Regular applications

- Apache, Nginx, MySQL, PostgreSQL, MongoDB, Redis, Memcached, Cassandra, Hadoop, RabbitMQ...
- Virtually all your programs in any language (services, web services, workers, everything!)
- They never ever need any kind of root privilege (except to install packages)
- Don't run them as root! Ever!

# Regular applications

- Risk: they run arbitrary code
  - vector: by definition, they *are* arbitrary code
  - vector: security breach causes execution of malicious code
- Fix: nothing
  - by definition, we are willing to execute arbitrary code here
- Consequence:
  assume those apps can try *anything* to break out

# Regular applications

- Risk: escalate from non-root to root
  - vector: vulnerabilities in SUID binaries
- Fix: defang SUID binaries
  - remove them
  - remove suid bit
  - mount filesystem with nosuid

# Regular applications

- Risk: execute arbitrary kernel code
  - vector: bogus syscall (e.g. vmsplice* in 2008)
- Fix: limit available syscalls
  - seccomp-bpf = whitelist/blacklist syscalls
- Fix: run stronger kernels
  - GRSEC is a good idea (stable patches for 3.14 since July 4th)
  - update often (i.e. have efficient way to roll out kernel upgrades)

*More details about that: http://lwn.net/Articles/268783/

# Regular applications

- Risk: leak to another container
  - vector: bug in namespace code; filesystem leak*
- Fix: user namespaces
  - map UID in container to a different UID outside
  - two containers run a process with UID 1000, but it's 14298 and 15398 outside
- Fix: security modules (e.g. SELinux)
  - assign different security contexts to containers
  - those mechanisms were *designed* to isolate!

*Like the one showed in the beginning of this talk!

# System services (high level)

# System services (high level)

- SSH, cron, syslog...
- You use/need them all the time
- Bad news: they typically run as root
- Good news: they don't really need root
- Bad news: it's hard to run them as non-root
- Good news: they are not arbitrary code

# System services (high level)

- Risk: running arbitrary code *as root*
  - vector: malformed data or similar
    (note: risk is pretty low for syslog/cron; much higher for SSH)
- Fix: isolate sensitive services
  - run SSH on bastion host, or in a VM
  - note: this is not container-specific
    (if someone hacks into your SSH server, you'll have a bad time)

# System services (high level)

- Risk: messing with /dev
  - vector: malicious code
- Fix: “devices” control group
  - whitelist/blacklist devices
  - fine-grained: can allow only read, write, none, or both
  - fine-grained: can specify major+minor number of device

# System services (high level)

- Risk: use of root calls (mount, chmod, iptables...)
  - vector: malicious code
- Fix: capabilities
  - break down “root” into many permissions
  - e.g. CAP_NET_ADMIN (network configuration)
  - e.g. CAP_NET_RAW (generate and sniff traffic)
  - e.g. CAP_SYS_ADMIN (big can of worms☹)
  - see `capabilities(7)`

# Interlude: CAP_SYS_ADMIN

Operations controlled by CAP_SYS_ADMIN...

- quotactl, mount, umount, swapon, swapoff
- sethostname, setdomainname
- IPC_SET, IPC_RMID on arbitrary System V IPC resources
- perform operations on trusted and security Extended Attributes
- set realtime priority
  (ioprio_set + IOPRIO_CLASS_RT)
- create new namespaces
  (clone and unshare + CLONE_NEWNS)

# System services (high level)

- Risk: messing with /proc, /sys
  - vector: malicious code
- Fix: prevent unauthorized access control
  - Mandatory Access Control (AppArmor, SELinux)
  - remount read-only, then drop CAP_SYS_ADMIN to prevent remount
- Fix: wider implementation of namespaces
  - some parts of procfs/sysfs are “namespace-aware”
  - some aren't, but can be fixed (by writing kernel code)

# System services (high level)

- Risk: leaking with UID 0
  - vector: malicious code
- Fix: user namespaces
  - already mentioned earlier
  - UID 0 in the container is mapped to some random UID outside
  - you break out: you're not root
  - you manage to issue weird syscalls: they're done as unprivileged UID
- Caveat: user namespaces are still new.
  We have to see how they behave with that!

# System services (low level)

# System services (low level)

- Device management (keyboard, mouse, screen), network and firewall config, filesystem mounts...
- You use/need some of them all the time
- But you don't need any of them in your containers
  - physical device management is done by the host
  - network configuration and filesystems are setup by the host
- Exceptions:
  - custom mounts (FUSE)
  - network appliances

# System services (low level)

- Risk: running arbitrary code *as root*
  - vector: malformed data or similar
- Fix: isolate sensitive functions
  - “one-shot” commands can be fenced in privileged context (think “sudo” but without even requiring “sudo”)
  - everything else (especially processes that are long-running, or handle arbitrary input) runs in non-privileged context
  - works well for FUSE, some VPN services

# System services (low level)

- Risk: run arbitrary code with full privileges
  - vector: needs a process running with full privileges (rare!)
  - vector: malformed data, unchecked input… classic exploit
- Fix: treat it as “kernel”
  - we'll see that immediately in the next section

# Kernel

# Kernel

- Drivers
  - they can talk to the hardware, so they can do pretty much anything
  - except: virtualize the bus and use e.g. *driver domains* (Xen)
- Network stacks
  - this *probably* has to live into the kernel for good performance
  - except: DPDK, OpenOnload... (networking stacks in userspace)
- Security policies
  - by definition, they control everything else
  - except: there might be nested security contexts some day

# Kernel

- Risk: run arbitrary code with absolute privileges
- Fix: ?

# Reality check:

if you run something which *by definition* needs full control over hardware or kernel, containers are not going to make it secure.

Please stop trying to shoot yourself in the foot safely.

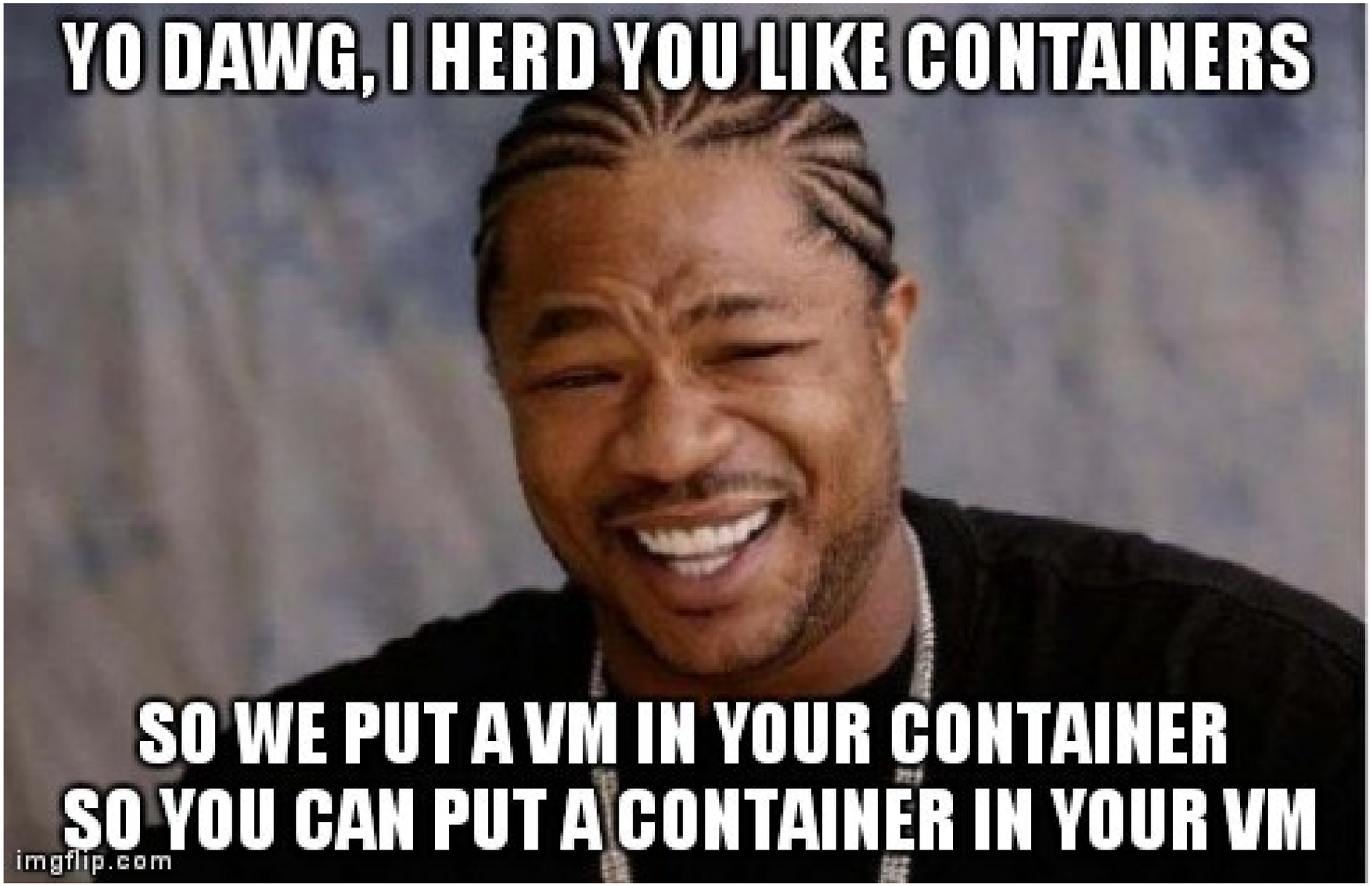
# Reality check:

if you run something which *by definition* needs full control over hardware or kernel, containers are not going to make it secure.

Please stop trying to shoot yourself in the foot safely.

# Kernel

- Risk: run arbitrary code with absolute privileges
- Fix: give it its own kernel and (virtual) hardware
  - i.e. run it in a virtual machine
  - that VM can run in a container
  - that VM can hold a container
  - run a privileged container in Docker in a VM in a container in Docker
    https://github.com/jpetazzo/docker2docker
  - inb4 xzibit meme


YO DAWG, I HERD YOU LIKE CONTAINERS
SO WE PUT A VM IN YOUR CONTAINER
SO YOU CAN PUT A CONTAINER IN YOUR VM
imgflip.com

# Immutable
# immutable
# infrastructure

# Immutable immutable infrastructure

- New rule: the whole container is read-only
- Compromise: if we must write, write to a noexec area
- Scalability has never been easier (if totally read-only)
- It's even harder for malicious users to do evil things

# Recap (in no specific order!)

- don't run things as root
- drop capabilities
- enable user namespaces
- get rid of shady SUID binaries
- enable SELinux (or AppArmor)
- use seccomp-bpf
- get a GRSEC kernel
- update kernels often
- mount everything read-only
- ultimately, fence things in VMs

# Thank you!

Questions?

- don't run things as root
- drop capabilities
- enable user namespaces
- get rid of shady SUID binaries
- enable SELinux (or AppArmor)
- use seccomp-bpf
- get a GRSEC kernel
- update kernels often
- mount everything read-only
- ultimately, fence things in VMs

See also:
- Docker in Production (1:40pm today)
- Office hour (10:40am tomorrow, expo hall Table A)
- Docker meet-up at New Relic (tomorrow evening)