docker

#dockertour


docker
docker

Docker

December 2014—Docker 1.4

Jérôme Petazzoni (@jpetazzo)

- Grumpy French DevOps
  - Go away or I will replace you with a very small shell script
- Runs everything in containers
  - VPN, firewalls
  - KVM, Xorg
  - Docker
  - ...

Let's start with

Questions

Raise your hand if you have ...

- Tried Docker (online tutorial)

Raise your hand if you have ...

- Tried Docker (online tutorial)
- Tried the *real* Docker (e.g. deployed remote VM)

Raise your hand if you have ...

- Tried Docker (online tutorial)
- Tried the *real* Docker (e.g. deployed remote VM)
- Installed Docker locally (e.g. with boot2docker)

Raise your hand if you have ...

- Tried Docker (online tutorial)
- Tried the *real* Docker (e.g. deployed remote VM)
- Installed Docker locally (e.g. with boot2docker)
- Written a Dockerfile (and built it!)

Raise your hand if you have ...

- Tried Docker (online tutorial)
- Tried the *real* Docker (e.g. deployed remote VM)
- Installed Docker locally (e.g. with boot2docker)
- Written a Dockerfile (and built it!)
- An image on Docker Hub (pushed or autobuilt)

Raise your hand if you have ...

- Tried Docker (online tutorial)
- Tried the *real* Docker (e.g. deployed remote VM)
- Installed Docker locally (e.g. with boot2docker)
- Written a Dockerfile (and built it!)
- An image on Docker Hub (pushed or autobuilt)
- Deployed Docker images for dev/QA/test/prod...

Agenda

- Where we come from
- What is Docker and Why it matters
- What are containers
- The Docker ecosystem
- Developing with Docker

from dotCloud

to Docker

What is dotCloud?

- Platform-as-a-Service
- Deploy with `git clone && dotcloud push`
- Compares to Heroku
- First « polyglot » PaaS ever (yay!)
- Built on top of LXC and AUFS
- Custom kernels (2.6.38+setns+grsec+aufs+fixes)

What is dotCloud?

- Platform-as-a-Service
- Deploy with `git clone && dotcloud push`
- Compares to Heroku
- First « polyglot » PaaS ever (yay!)
- Built on top of LXC and AUFS
- Custom kernels (2.6.38+setns+grsec+aufs+fixes)

dotCloud from 30,000ft above

- Micro-services architecture (100+ services)
  - git, hg, rsync repositories
  - builders for different languages (Python, Ruby, PHP, Java, Node.js, Perl, ...)
  - different data stores (MySQL, PostgreSQL, Redis, MongoDB...)
  - TCP port mappers, HTTP load balancers (switched from Nginx to custom Hipache)
  - and of course: billing, users, metrics, logging...

dotCloud container management

- Some platform-wide services
- Some per-host components:
  - containers, builder, deployer, hostinfo, oomkn, metrics, diskwatcher, unfreezer...
- Simple scheduling service
  - distributed, lock-based, non-deterministic, single-resource, bin-packing algorithm

The problem

- Containers are handled by multiple components
- Locking abounds
- More time spent to debug concurrency issues, than implementing features (sometimes)
- Container management code cannot be in a container
- Different deployment mechanisms for customer code and for platform code

The solution

- One daemon to manage them all
- No concurrent access, no locking, no problem
- Simple code with less dependencies (easier deployment)

Docker is born!

- docker.py
- (not to be confused with today's docker-py)

Can we do better?

- It's Python
- It's not Ruby
- It's easy to install, but can we make it easier?

Thoughts...

- Let's redo it in Ruby!
- But then it won't be Python (duh!)
- We can't even Ruby
- We don't want our engineering team to quit
- Deployment of Ruby code is just as bad as deployment of Python code anyway

Thoughts...

- Let's redo it in Node.js!
- Bad cultural/technical fit
- Deployment of Node.js code is just as bad as deployment of Python code anyway

Thoughts...

- Let's redo it in Java!
- C'est cela, oui...

Golang

- It's not Python
- It's not Ruby
- It's not Java
- It's not Node.js
- It compiles to a single, quasi-static binary

Docker is reborn!

- February 2013: private repo, with liberal access (~200 people had access and helped to review, contribute, give feedback, etc.)
- March 2013: Docker 0.1 released at PyCon
- Requires LXC, AUFS
- Works on Debian/Ubuntu kernels

Stop.
Demo time.

```
root@dockerhost:~#
```

Community response

« Five stars, pls code again »

First milestones

- 0.1.0 (2013-03-23), initial public release
- 0.2.0 (2013-04-23), automatic bridge setup
- 0.3.0 (2013-05-06), volumes
- 0.4.0 (2013-06-03), API, docker build
- 0.5.0 (2013-07-17), host volumes, UDP ports
- 0.6.0 (2013-08-22), privileged mode

The road to 1.0

- 0.7.0 (2013-11-25), links, storage drivers (AUFS, DM, VFS)
- 0.8.0 (2014-02-04), BTRFS, OSX CLI
- 0.9.0 (2014-03-10), native exec driver
- 0.10.0 (2014-04-08), TLS API support
- 0.11.0 (2014-05-07), SELinux, DNS links, --net
- 0.12.0 (2014-06-05), pause/unpause

Life after 1.0

- 1.0.0 (2014-06-09), released at DockerCon
- 1.1.0 (2014-07-03), .dockerignore, logs --tail
- 1.2.0 (2014-08-20), auto-restart policies, capability add/drop, fine-grained device access
- 1.3.0 (2014-10-14), docker exec, docker start
- 1.4.0 (2014-12-11), overlayfs
- In progress: volumes, composition, hosts

Initial goals
vs
Docker now

Initial goals

- LXC container engine to build a PaaS
- Containers as lightweight VMs
(complete with syslog, cron, ssh...)
- Part of a bigger puzzle
(other parts: builders, load balancers...)

Docker now

- Build, ship, and run any app, anywhere

Say again?

- Build: package your application in a container
- Ship: move that container from a machine to another
- Run: execute that container (i.e. your application)
- Any application: anything that runs on Linux
- Anywhere: local VM, cloud instance, bare metal...

build

Dockerfiles

- Recipe to build a container
- Start `FROM` a base image
- `RUN` commands on top of it
- Easy to learn, easy to use

```
FROM ubuntu:14.04

RUN apt-get update
RUN apt-get install -y nginx
RUN echo 'Hi, I am in your container!' \
    >/usr/share/nginx/html/index.html

CMD nginx -g "daemon off;"

EXPOSE 80
```

```
docker build -t jpetazzo/web .
docker run -d -P jpetazzo/web
```

```
root@dockerhost:~#
```

« docker build » goodness

- Takes a snapshot after each step
- Re-uses those snapshots in future builds
- Doesn't re-run slow steps (package install...) when it is not necessary


DOCKWISE

ship

Docker Hub

- `docker push` an image to the Hub
- `docker pull` this image from any machine

```
root@dockerhost:~#
```

Why does this matter?

Deploy reliably & consistently


WORKED FINE IN DEV
OPS PROBLEM NOW

Deploy reliably & consistently

- Images are self-contained, independent from host
- If it works locally, it will work on the server
- *With exactly the same behavior*
- Regardless of versions
- Regardless of distros
- Regardless of dependencies

run

Execution is *fast* and *lightweight*

- Let's start a few containers

```
root@dockerhost:~#
```


seaco
HAMBURG SÜD

Execution is *fast* and *lightweight*

- Containers have no* overhead
  - Lies, damn lies, and other benchmarks:

    http://qiita.com/syoyo/items/bea48de8d7c6d8c73435
    http://www.slideshare.net/BodenRussell/kvm-and-docker-lxc-benchmarking-with-openstack

*For some definitions of « no »

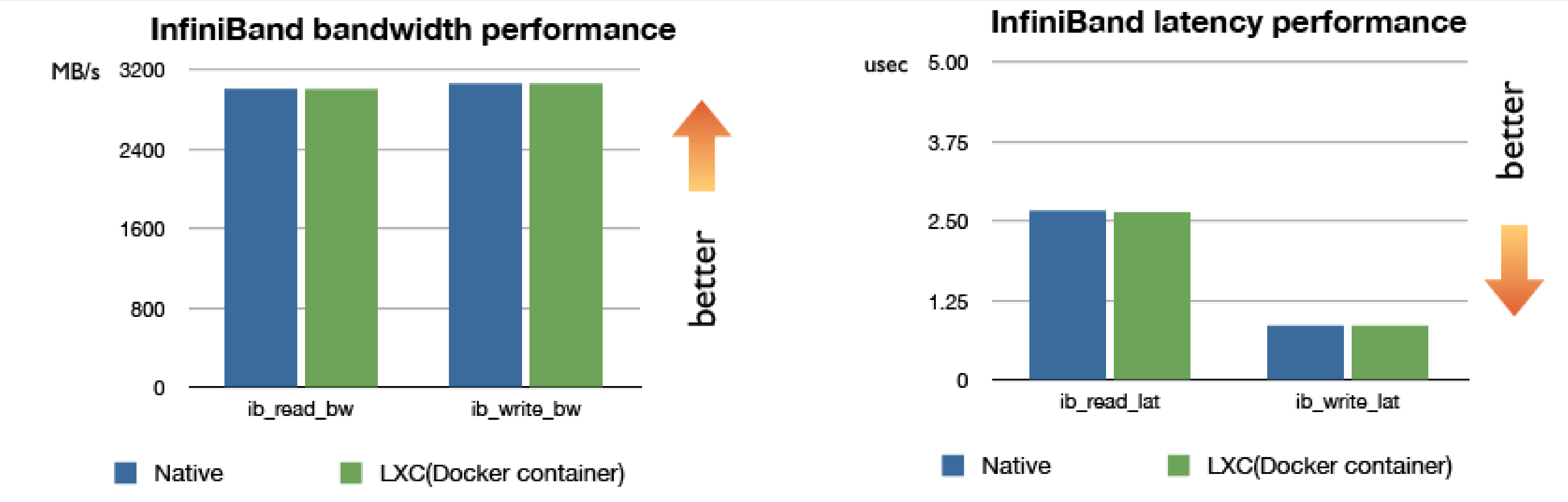
Benchmark: container creation

```
$ time docker run ubuntu echo hello world
hello world
real 0m0.258s
```

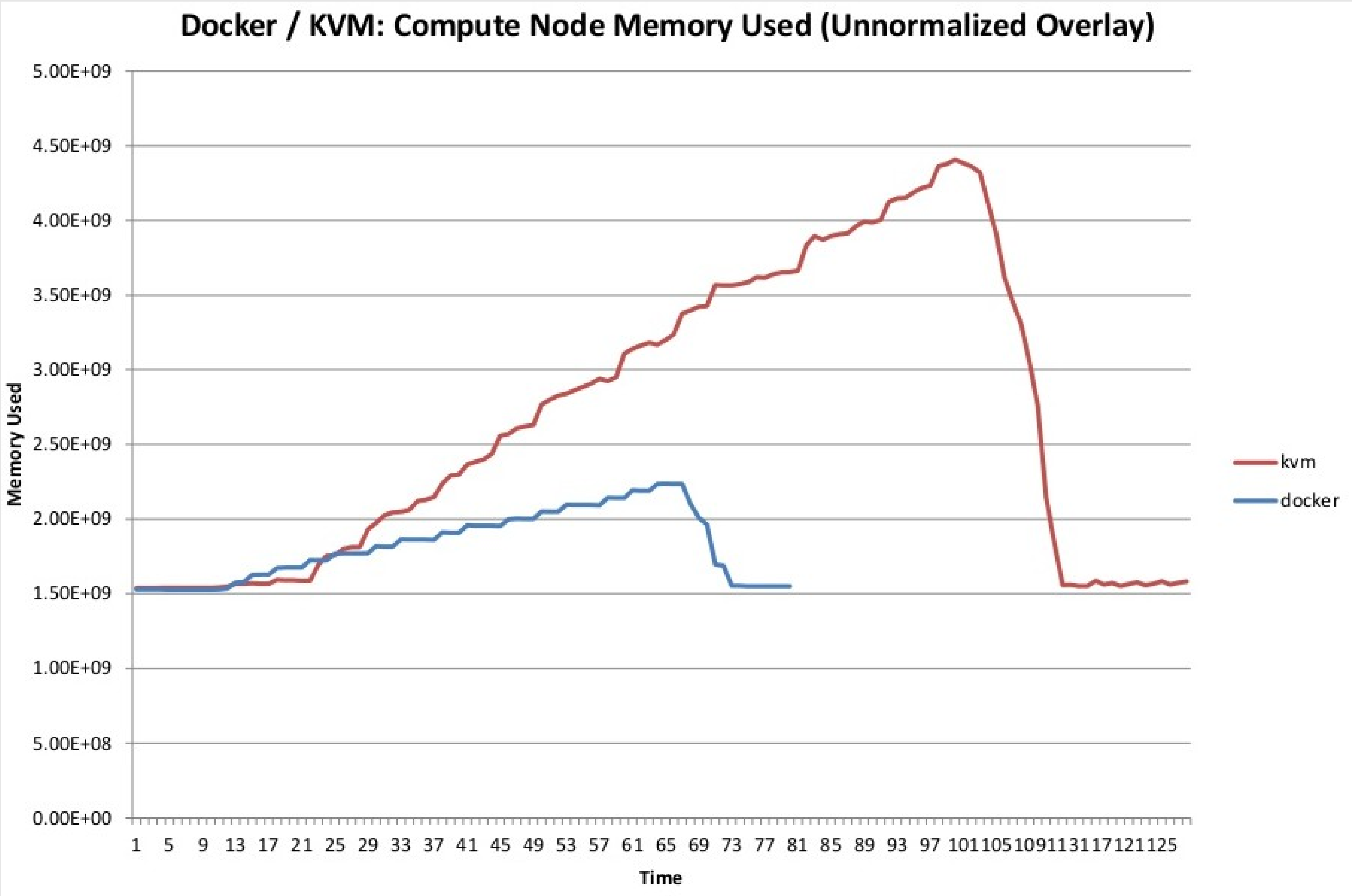
Disk usage: less than 100 kB

Memory usage: less than 1.5 MB

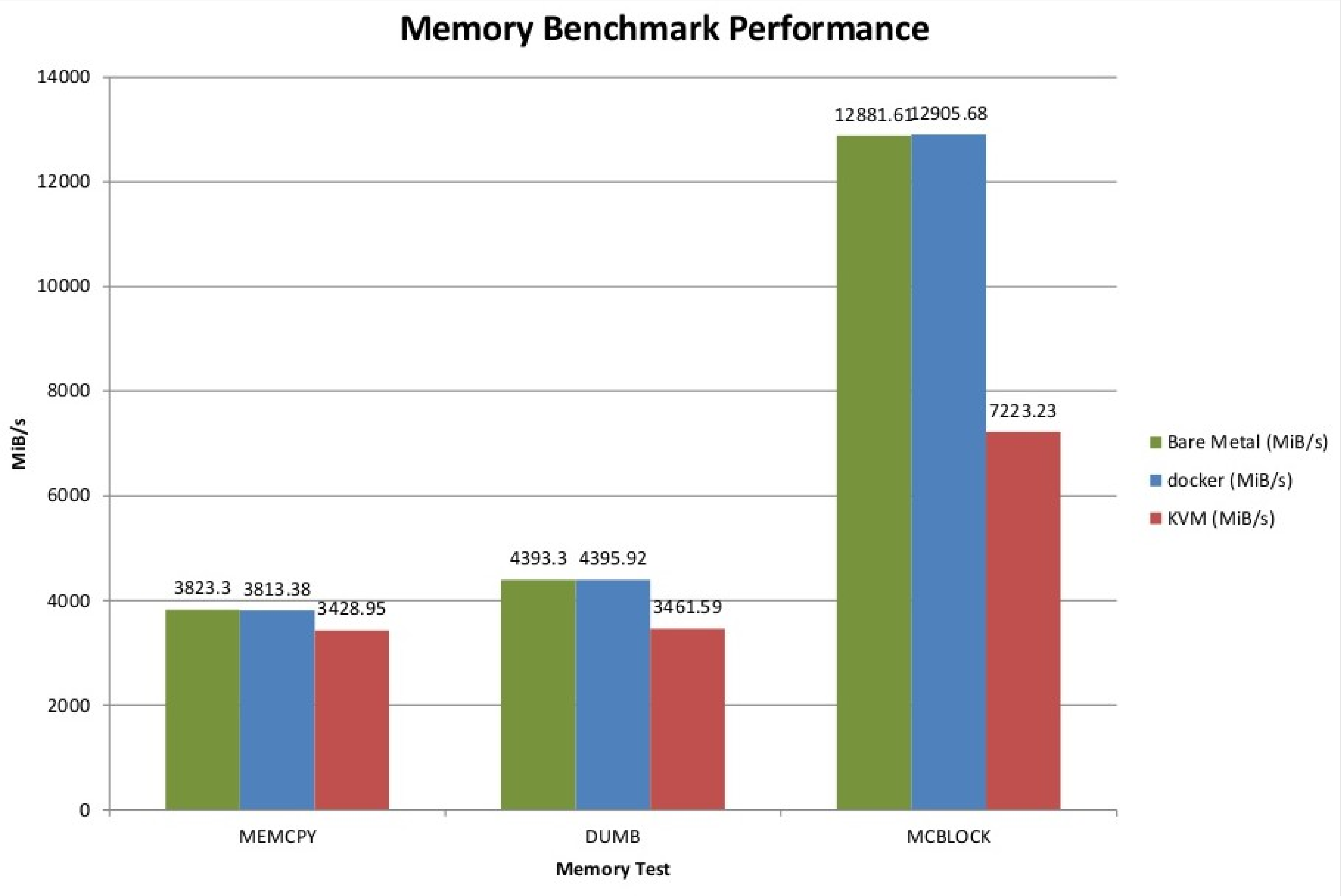
Benchmark: infiniband throughput and latency



Benchmark: booting OpenStack instances



Benchmark: memory speed



Is there really *no* overhead at all?

- Processes are isolated,
  but run straight on the host
- Code path in containers
  = code path on native
- CPU performance
  = native performance
- Memory performance
  = a few % shaved off for (optional) accounting
- Network and disk I/O performance
  = small overhead; can be reduced to zero

any app

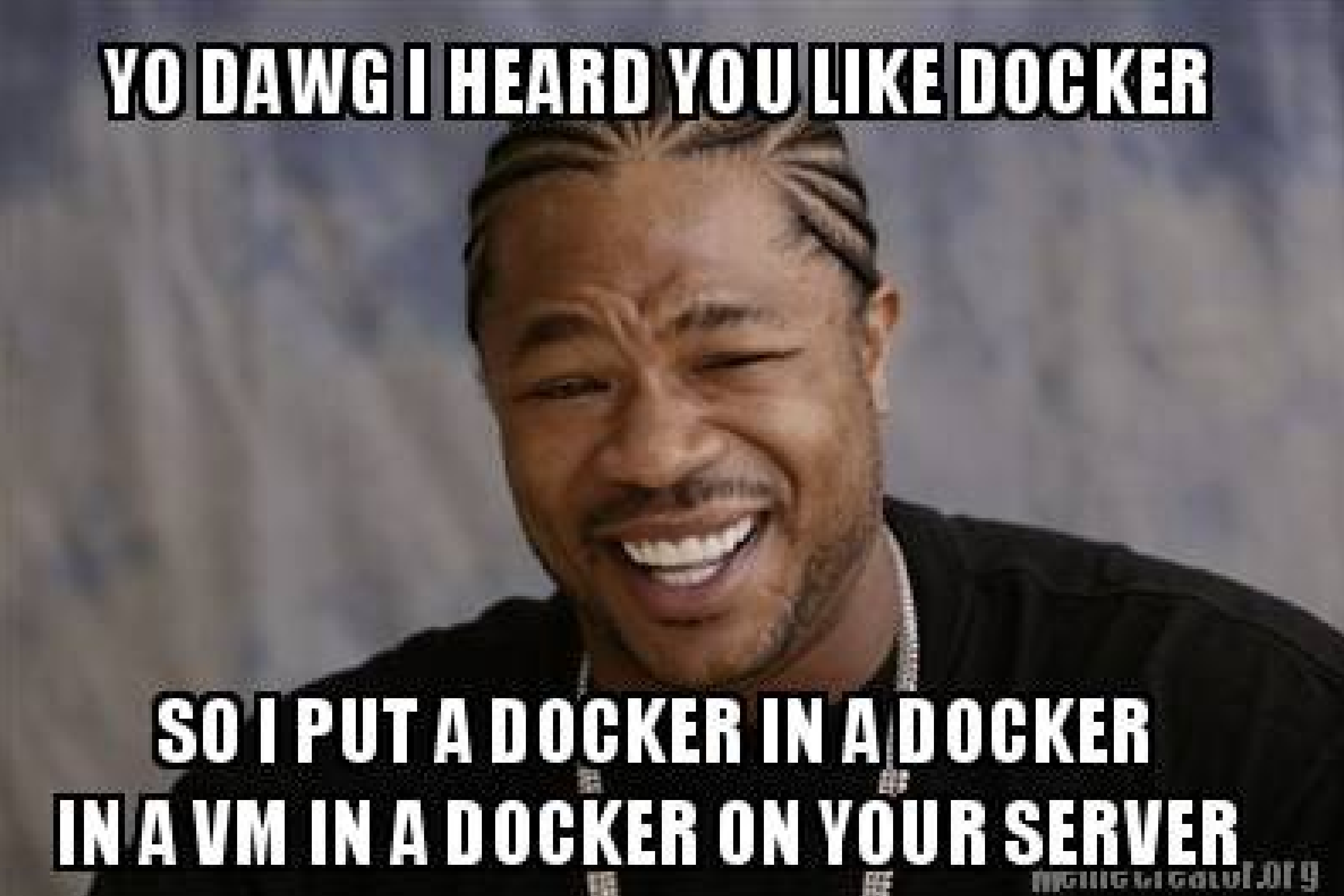
If it runs on Linux, it will run in Docker

- Web apps
- API backends
- Databases (SQL, NoSQL)
- Big data
- Message queues
- … and more

docker

If it runs on Linux, it will run in Docker

- Firefox-in-Docker
- Xorg-in-Docker
- VPN-in-Docker
- Firewall-in-Docker
- Docker-in-Docker
- KVM-in-Docker

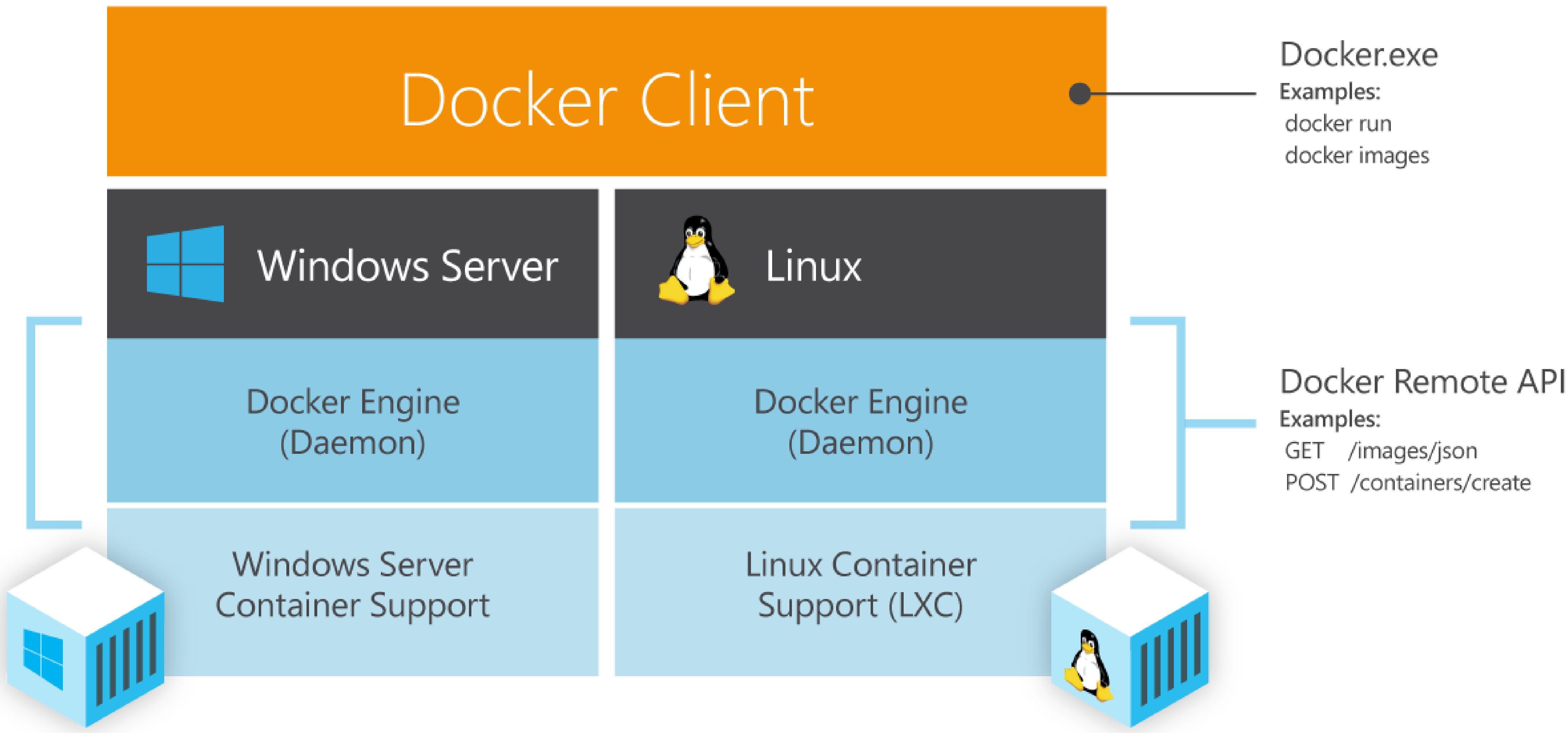

YO DAWG I HEARD YOU LIKE DOCKER
SO I PUT A DOCKER IN A DOCKER
IN A VM IN A DOCKER ON YOUR SERVER

anywhere

Deploy almost anywhere

- Linux servers
- VMs or bare metal
- Any distro
- Kernel 3.8+
  (or 2.6.32 that comes with RHEL/CentOS 6.5)
- Intel 64 bits (x86_64)

Deploy ~~almost~~ anywhere



Deploy ~~almost~~ anywhere

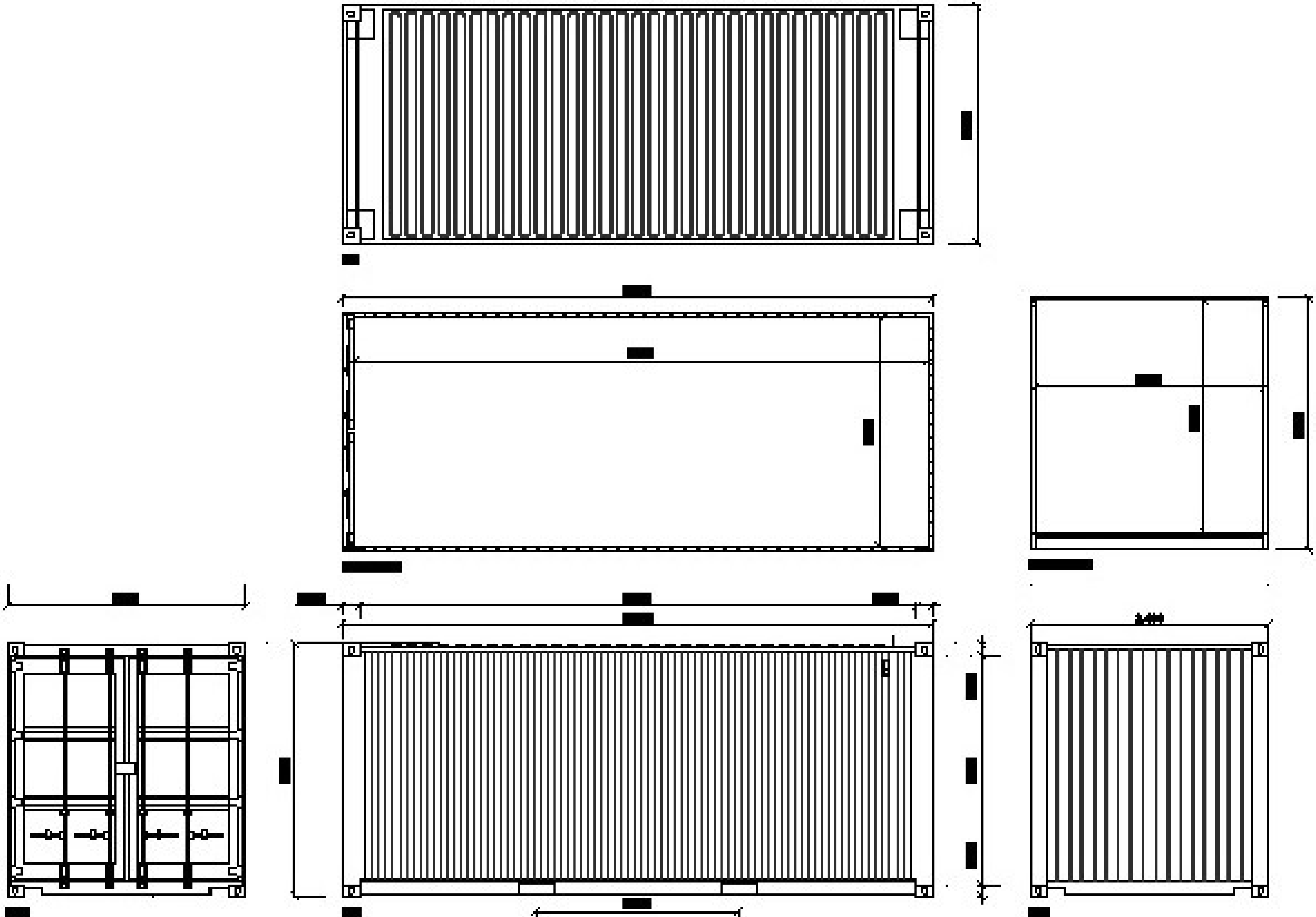
- Some people run Docker on:
  - Intel 32 bits
  - ARM 32 and 64 bits
  - MIPS
  - Power8
  - Older kernels **(please don't)**
- Note: the Docker Hub registry is not arch-aware (yet!) so you will need to find your own base images.

docker

Science

Docker can help ...

- If it works on my machine, it works on the cluster
- Shrinkwrap code and data for future reuse *(recomputability)*
- Small but durable recipes (≠VM images)
- Never again:
  - juggle with 3 different, incompatible Fortran compilers
  - wave dead chickens to get that exotic lib to link with IDL
  - figure out which version of LAPACK works with that code
  - … and what obscure flag coaxed it into compiling last time



Tell me more about those *containers.*

High level approach: it's a lightweight VM

- Own process space
- Own network interface
- Can run stuff as root
- Can have its own /sbin/init (different from the host)

« Machine Container »

Low level approach:
it's chroot on steroids

- Can also *not* have its own /sbin/init
- Container = isolated process(es)
- Share kernel with host
- No device emulation (neither HVM nor PV)

« Application Container »

How does it work?
Isolation with namespaces

- pid
- mnt
- net
- uts
- ipc
- user

pid namespace

```
jpetazzo@tarrasque:~$ ps aux | wc -l
212

jpetazzo@tarrasque:~$ sudo docker run -t -i ubuntu bash
root@ea319b8ac416:/# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  18044  1956 ?        S    02:54   0:00 bash
root        16  0.0  0.0  15276  1136 ?        R+   02:55   0:00 ps aux
```

(That's 2 processes)

mnt namespace

```
jpetazzo@tarrasque:~$ wc -l /proc/mounts
32 /proc/mounts


root@ea319b8ac416:/# wc -l /proc/mounts
10 /proc/mounts
```

net namespace

```
root@ea319b8ac416:/# ip addr

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever

22: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 2a:d1:4b:7e:bf:b5 brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.3/24 brd 10.1.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::28d1:4bff:fe7e:bfb5/64 scope link
       valid_lft forever preferred_lft forever
```

uts namespace

```
jpetazzo@tarrasque:~$ hostname
tarrasque

root@ea319b8ac416:/# hostname
ea319b8ac416
```

ipc namespace

```
jpetazzo@tarrasque:~$ ipcs
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 3178496    jpetazzo   600        393216     2          dest
0x00000000 557057     jpetazzo   777        2778672    0
0x00000000 3211266    jpetazzo   600        393216     2          dest


root@ea319b8ac416:/# ipcs
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

`user` namespace

- No demo, integration in progress
- UID 0→1999 in container C1 is mapped to UID 10000→11999 in host;
  UID 0→1999 in container C2 is mapped to UID 12000→13999 in host; etc.
- Will add one extra layer of security

How does it work?
Isolation with cgroups

- memory
- cpu
- blkio
- devices

memory cgroup

- Keeps track pages used by each group:
  - file (read/write/mmap from block devices; swap)
  - anonymous (stack, heap, anonymous mmap)
  - active (recently accessed)
  - inactive (candidate for eviction)
- Each page is « charged » to a group
- Pages can be shared (e.g. if you use any COW FS)
- Individual (per-cgroup) limits and out-of-memory killer

memory cgroup

- Keeps track pages used by each group:
  - file (read/write/mmap from block devices; swap)
  - anonymous (stack, heap, anonymous mmap)
  - active (recently accessed)
  - inactive (candidate for eviction)
- Each page is « charged » to a group
- Pages can be shared (e.g. if you use any COW FS)
- Individual (per-cgroup) limits and out-of-memory killer

```
root@dockerhost:~#
```

cpu and cpuset cgroups

- Keep track of user/system CPU time
- Set relative weight per group
- Pin groups to specific CPU(s)
  - Can be used to « reserve » CPUs for some apps
  - This is also relevant for big NUMA systems

`blkio` cgroups

- Keep track IOs for each block device
  - read vs write; sync vs async
- Set relative weights
- Set throttle (limits) for each block device
  - read vs write; bytes/sec vs operations/sec

Note: earlier versions (<3.8) didn't account async correctly.
3.8 is better, but use 3.10 and above for best results.

special case: `devices` cgroups

- Controls read/write/mknod permissions
- Typically:
  - allow: /dev/{tty,zero,random,null}...
  - deny: everything else
  - maybe: /dev/net/tun, /dev/fuse, /dev/kvm, /dev/dri...
- Fine-grained control for GPU, virtualization, etc.
- ~a bit like PCI pass-through

How does it work?
Copy-on-write storage

- Create a new machine instantly
  (Instead of copying its whole filesystem)
- Storage keeps track of what has changed
- Multiple storage plugins available
  (AUFS, device mapper, BTRFS, overlayfs, VFS)

Storage options

	Union Filesystems (AUFS, overlayfs)	Copy-on-write block devices	Snapshotting filesystems
Provisioning	Superfast Supercheap	Average Cheap	Fast Cheap
Changing small files	Superfast Supercheap	Fast Costly	Fast Cheap
Changing large files	Slow (first time) Inefficient (copy-up!)	Fast Cheap	Fast Cheap
Diffing	Superfast	Slow	Superfast
Memory usage	Efficient	Inefficient (at high densities)	Inefficient (but may improve)
Drawbacks	Random quirks AUFS not mainline Overlayfs bleeding edge	Higher disk usage Great performance (except diffing)	ZFS not mainline BTRFS not as nice
Bottom line	**Ideal for PAAS, CI/CD, high density things**	**Works everywhere, but slow and inefficient**	Will be great once memory usage is fixed


TT166 PSA
TT 095
PSA

Docker's Ecosystem

Docker: the cast

- Docker Engine
- Docker Hub
- Docker, the community
- Docker Inc, the company

Docker Engine

- Open Source engine to **commoditize** LXC
- Uses copy-on-write for quick provisioning
- Written in Go, runs as a daemon, comes with a CLI
- Everything exposed through a REST API
- Allows to **build** images in standard, reproducible way
- Allows to **share** images through **registries**
- Defines **standard format** for containers
  (stack of layers; 1 layer = tarball+metadata)

… Open Source?

- Nothing up the sleeve, everything on the table
  - Public GitHub repository: https://github.com/docker/docker
  - Bug reports: GitHub issue tracker
  - Mailing lists: docker-user, docker-dev (Google groups)
  - IRC channels: #docker, #docker-dev (Freenode)
  - New features: GitHub pull requests (see `CONTRIBUTING.md`)
  - Docker Governance Advisory Board (elected by contributors)

Docker Hub

Collection of services to make Docker more useful.

- Library of official base images
- Public registry
  (push/pull your images for free)
- Private registry
  (push/pull secret images for $)
- Automated builds
  (link github/bitbucket repo; trigger build on commit)
- More to come!

Docker, the community

- >700 contributors
- ~20 core maintainers
- >40,000 Dockerized projects on GitHub
- >60,000 repositories on Docker Hub
- >25000 meetup members,
  >140 cities, >50 countries
- >2,000,000 downloads of boot2docker

Docker Inc, the company

- Headcount: ~70
- Revenue:
  - t-shirts and stickers featuring the cool blue whale
  - SAAS delivered through Docker Hub
  - Support & Training

Developing with Docker

One-time setup

- On your dev env (Linux, OS X, Windows)
  - boot2docker (25 MB VM image)
  - Natively (if you run Linux)
- On your servers (Linux)
  - Packages (Ubuntu, Debian, Fedora, Gentoo, Arch...)
  - Single binary install (Golang FTW!)
  - Easy provisioning on Azure, Rackspace, Digital Ocean...
  - Special distros: CoreOS, Project Atomic, Ubuntu Core

Authoring images with a Dockerfile

- Minimal learning curve
- Rebuilds are easy
- Caching system makes rebuilds faster
- Single file to define the whole environment

Authoring images with a Dockerfile

- Minimal learning curve
- Rebuilds are easy
- Caching system makes rebuilds faster
- ~~Single file to define the whole environment~~
- Single file to define the whole component

docker

CONTAINERS

They're stable, they said. Stack them, they said.

Running multiple containers

Fig

Fig

- Run your stack with *one* command: `fig up`
- Describe your stack with *one* file: `fig.yml`
- Example: Python+Redis webapp

```
web:
  build: .
  command: python app.py
  ports:
   - "5000:5000"
  volumes:
   - .:/code
  links:
   - redis:redis

redis:
  image: redis
```

```
root@dockerhost:~#
```

Per-project setup

- Write Dockerfiles
- Write fig.yml file(s)
- Test the result
  (i.e.: Make sure that « git clone ; fig up » works on new Docker machine works fine

Per-developer setup

- Make sure that they have Docker (boot2docker or other method)
- git clone ; fig up
- Done

Development workflow

- Edit code
- Iterate locally or in a container
(use volumes to share code between local machine and container)
- When ready to test « the real thing », fig up

Going to production

- There are *many* options
- I actually wrote a full 45-minutes talk about « Docker to production »

Implementing CI/CD

- Each time I commit some code, I want to:
  - build a container with that code
  - test that container
  - if the test is successful, promote that container

Docker Hub to the rescue

- Automated builds let you link github/bitbucket repositories to Docker Hub repositories
- Each time you push to github/bitbucket:
  - Docker Hub fetches your changes,
  - builds new containers images,
  - pushes those images to the registry.

Coming next on Docker Hub...

- Security notifications
- Automated deployment to Docker hosts
- Docker Hub Enterprise
  (all those features, on *your* infrastructure)

Summary

With Docker, I can:

- put my software in containers
- run those containers anywhere
- write recipes to automatically build containers
- use Fig to effortlessly start stacks of containers
- automate testing, building, hosting of images, using the Docker Hub

Would You Like To Know More?

- Get in touch on Freenode IRC channels #docker #docker-dev
- Ask me tricky questions jerome@docker.com
- Get your own Docker Hub on prem sales@docker.com
- Follow us on Twitter @docker, @jpetazzo


docker
docker