a
Gentle Introduction
to
Docker
and
All Things Containers

Outline

- **Whom is this for?**
- What's the problem?
- What's a Container?
- Docker 101
- Docker images
- Docker deployment
- Docker future

Devs

- all languages
- all databases
- all O/S
- targetting Linux systems

Docker will eventually be able to target FreeBSD, Solaris, and maybe OS X.

Ops

- any distro[1]
- any cloud[2]
- any machine (physical, virtual...)
- recent kernels[3]

[1] *as long as it's Ubuntu or Debian ☺ others coming soon*
[2] *as long as they don't ship with their custom crappy kernel*
[3] *at least 3.8; support for RHEL 2.6.32 on the way*

CFO, CIO, CTO, ...

- LESS overhead!
- MOAR consolidation!
- MOAR agility!
- LESS costs!

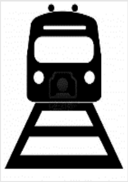
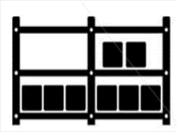
Outline

- Whom is this for?
- **What's the problem?**
- What's a Container?
- Docker 101
- Docker images
- Docker deployment
- Docker future

The Matrix From Hell

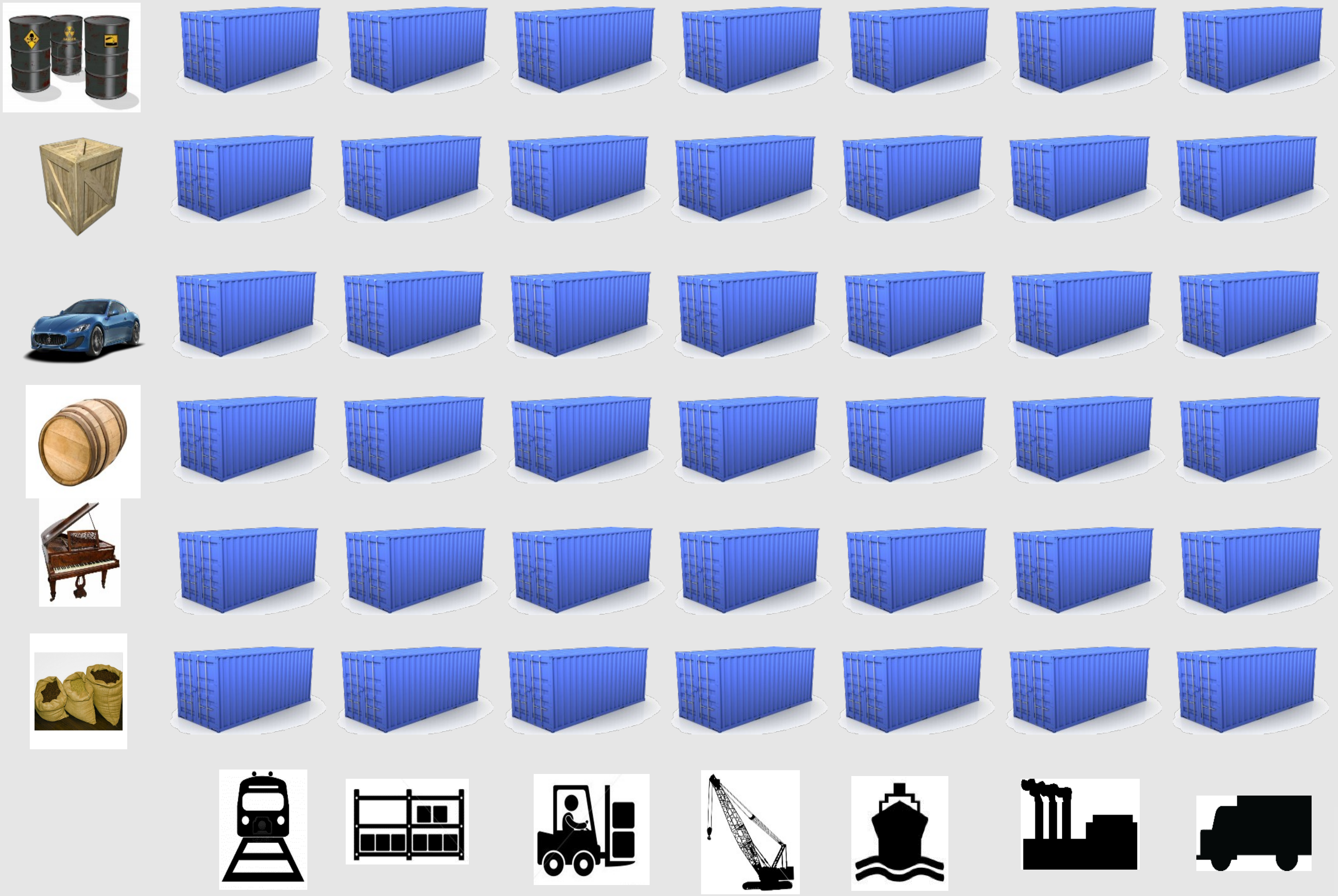
django web frontend	?	?	?	?	?	?
node.js async API	?	?	?	?	?	?
background workers	?	?	?	?	?	?
SQL database	?	?	?	?	?	?
distributed DB, big data	?	?	?	?	?	?
message queue	?	?	?	?	?	?
	my laptop	your laptop	QA	staging	prod on cloud VM	prod on bare metal

Another Matrix from Hell

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution:
the *intermodal shipping container*

Solved!



Solution to the deployment problem:
the *Linux* container

Linux containers...

Units of software delivery (**ship it!**)

- run everywhere
  - regardless of kernel version
  - regardless of host distro
  - (but container and host architecture must match*)
- run anything
  - if it can run on the host, it can run in the container
  - i.e., if it can run on a Linux kernel, it can run

*Unless you emulate CPU with qemu and binfmt

Outline

- Whom is this for?
- What's the problem?
- **<u>What's a Container?</u>**
- Docker 101
- Docker images
- Docker deployment
- Docker future

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 »

Separation of concerns: Dave the Developer

- inside my container:
  - my code
  - my libraries
  - my package manager
  - my app
  - my data

Separation of concerns: Oscar the Ops guy

- outside the container:
  - logging
  - remote access
  - network configuration
  - monitoring

How does it work?
Isolation with namespaces

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

How does it work?
Isolation with cgroups

- memory
- cpu
- blkio
- devices

If you're serious about security, you also need…

- capabilities
  - okay: cap_ipc_lock, cap_lease, cap_mknod, cap_net_admin, cap_net_bind_service, cap_net_raw
  - troublesome: cap_sys_admin (mount!)
- think twice before granting root
- grsec is nice
- seccomp (very specific use cases); seccomp-bpf
- beware of full-scale kernel exploits!

How does it work?
Copy-on-write storage

- unioning filesystems
  (AUFS, overlayfs)
- snapshotting filesystems
  (BTRFS, ZFS)
- copy-on-write block devices
  (thin snapshots with LVM or device-mapper)

This is now being integrated with low-level LXC tools as well!

Efficiency

Compute efficiency: *almost* no overhead

- processes are isolated, but run straight on the host
- CPU performance = native performance
- memory performance = a few % shaved off for (optional) accounting
- network performance = small overhead; can be reduced to zero

Storage efficiency: many options!

	Union Filesystems	Snapshotting Filesystems	Copy-on-write block devices
Provisioning	Superfast Supercheap	Fast Cheap	Fast Cheap
Changing small files	Superfast Supercheap	Fast Cheap	Fast Costly
Changing large files	Slow (first time) Inefficient (copy-up!)	Fast Cheap	Fast Cheap
Diffing	Superfast	Superfast (ZFS) Kinda meh (BTRFS)	Slow
Memory usage	Efficient	Efficient	Inefficient (at high densities)
Drawbacks	Random quirks AUFS not mainline !AUFS more quirks	ZFS not mainline BTRFS not as nice	Higher disk usage Great performance (except diffing)
Bottom line	**Ideal for PAAS and high density things**	**This might be the Future**	**Dodge Ram 3500**

Outline

- Whom is this for?
- What's the problem?
- What's a Container?
- **Docker 101**
- Docker images
- Docker deployment
- Docker future

docker

Docker-what?

- Open Source engine to **commoditize** LXC
- using copy-on-write for quick provisioning

STOP!

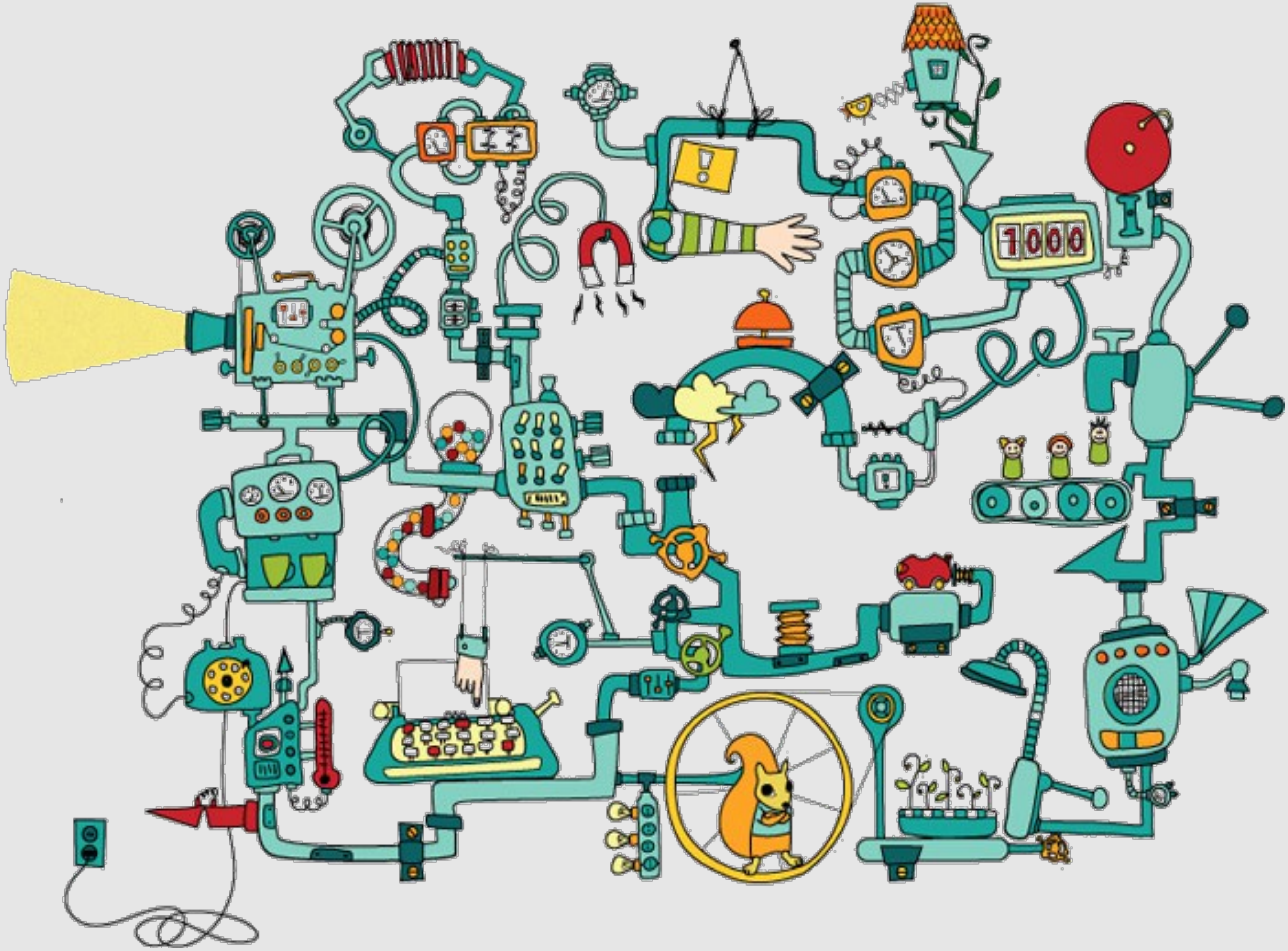
~~HAMMER~~ DEMO TIME.

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

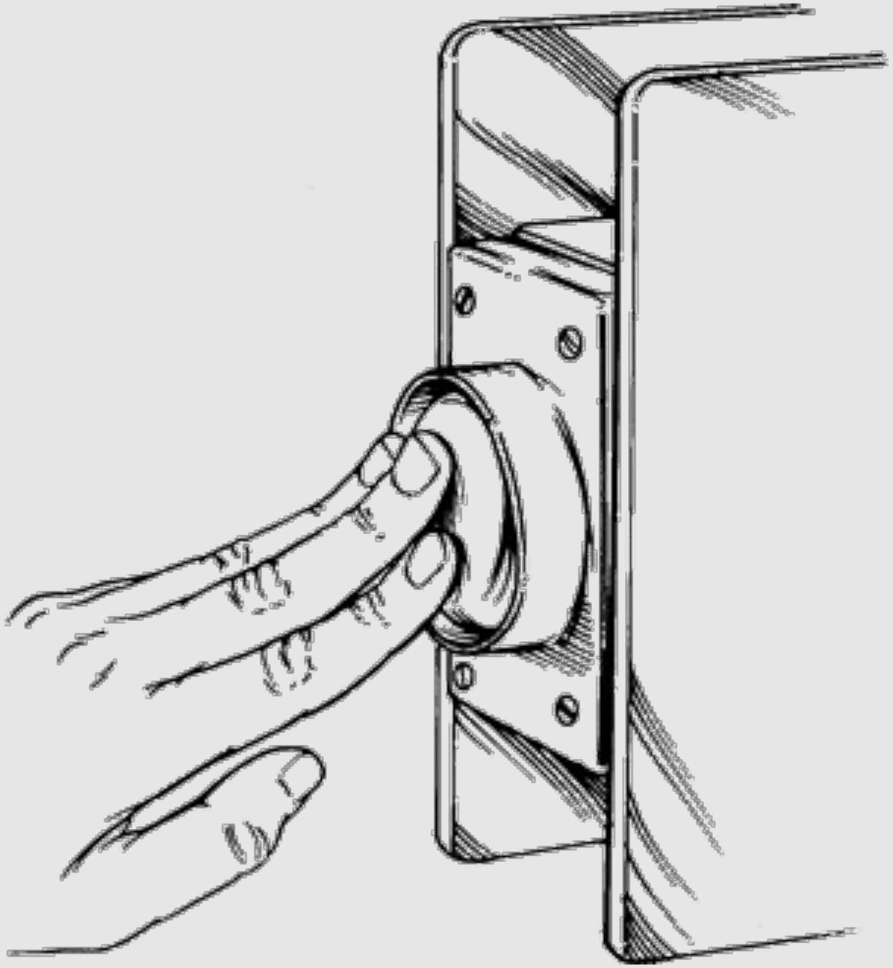
Yes, but...

- « I don't need Docker;
  I can do all that stuff with LXC tools, rsync, some scripts! »
- correct on all accounts;
  but it's also true for apt, dpkg, rpm, yum, etc.
- the whole point is to **commoditize**,
  i.e. make it ridiculously easy to use

Containers before Docker



Containers after Docker



What this really means…

- instead of writing « very small shell scripts » to manage containers, write them to do the rest:
  - continuous deployment/integration/testing
  - orchestration
- = use Docker as a building block
- re-use other people images (yay ecosystem!)

Docker-what?
The Big Picture

- Open Source engine to commoditize LXC
- using copy-on-write for quick provisioning
- allowing to **create and share** *images*
- **standard format** for containers
  (stack of layers; 1 layer = tarball+metadata)
- standard, *reproducible* way to *easily* build
  trusted images (Dockerfile, Stackbrew...)

Docker-what?
Under The Hood

- rewrite of dotCloud internal container engine
  - original version: Python, tied to dotCloud's internal stuff
  - released version: Go, legacy-free
- the Docker daemon runs in the background
  - manages containers, images, and builds
  - HTTP API (over UNIX or TCP socket)
  - embedded CLI talking to the API
- Open Source (GitHub public repository + issue tracking)
- user and dev mailing lists
- FreeNode IRC channels #docker, #docker-dev

Outline

- Whom is this for?
- What's the problem?
- What's a Container?
- Docker 101
- **<u>Docker images</u>**
- Docker deployment
- Docker future

Authoring images with run/commit

1) docker run ubuntu bash
2) apt-get install this and that
3) docker commit <containerid> <imagename>
4) docker run <imagename> bash
5) git clone git://.../mycode
6) pip install -r requirements.txt
7) docker commit <containerid> <imagename>
8) repeat steps 4-7 as necessary
9) docker tag <imagename> <user/image>
10) docker push <user/image>

Authoring images with a Dockerfile

```
FROM ubuntu

RUN apt-get -y update
RUN apt-get install -y g++
RUN apt-get install -y erlang-dev erlang-manpages erlang-base-hipe ...
RUN apt-get install -y libmozjs185-dev libicu-dev libtool ...
RUN apt-get install -y make wget

RUN wget http://.../apache-couchdb-1.3.1.tar.gz | tar -C /tmp -zxf-
RUN cd /tmp/apache-couchdb-* && ./configure && make install

RUN printf "[httpd]\nport = 8101\nbind_address = 0.0.0.0" >
    /usr/local/etc/couchdb/local.d/docker.ini

EXPOSE 8101
CMD ["/usr/local/bin/couchdb"]
```

```
docker build -t jpetazzo/couchdb .
```

Authoring Images with Trusted Builds

0) create a GitHub account

On index.docker.io:
1) create a Docker account
2) link it with your GitHub account
3) enable Trusted Builds on any public repo

On your dev env:
4) git add Dockerfile
5) git commit
6) git push

Authoring Images with Chef/Puppet/Ansible/Salt/...

Plan A: « my other VM is a container »

- write a Dockerfile to install $YOUR_CM
- start tons of containers
- run $YOUR_CM in them

Good if you want a mix of containers/VM/metal

But slower to deploy, and uses more resources

Authoring Images with Chef/Puppet/Ansible/Salt/...

Plan B: « the revolution will be containerized »

- write a Dockerfile to install $YOUR_CM
- … and *run* $YOUR_CM as part of build process
- deploy fully baked images

Faster to deploy

Easier to rollback

Outline

- Whom is this for?
- What's the problem?
- What's a Container?
- Docker 101
- Docker images
- **Docker deployment**
- Docker future

Running containers

- SSH to Docker host and manual pull+run
- REST API (feel free to add SSL certs, OAUth...)
- OpenStack Nova
- OpenStack Heat
- who's next? OpenShift, CloudFoundry?
- multiple Open Source PAAS built on Docker (Cocaine, Deis, Flynn...)

Orchestration & Service Discovery (0.6.5)

- you can name your containers
- they get a generated name by default (red_ant, gold_monkey...)
- you can link your containers

```
docker run -d -name frontdb
docker run -d -link frontdb:sql frontweb
```

→ container frontweb gets one bazillion environment vars

Orchestration & Service Discovery roadmap

- currently single-host
- problem:
  how do I link with containers on other hosts?
- solution:
  ambassador pattern!
  - app container runs in its happy place
  - other things (Docker, containers...) plumb it

Orchestration roadmap

- currently static
- problem: what if I want to…
  move a container?
  do a master/slave failover?
  WebScale my MangoDB cluster?
- solution:
  dynamic discovery!