O'REILLY

oscon

OPEN SOURCE CONVENTION

PORTLAND, OR
JULY 21-24, 2014

#oscon

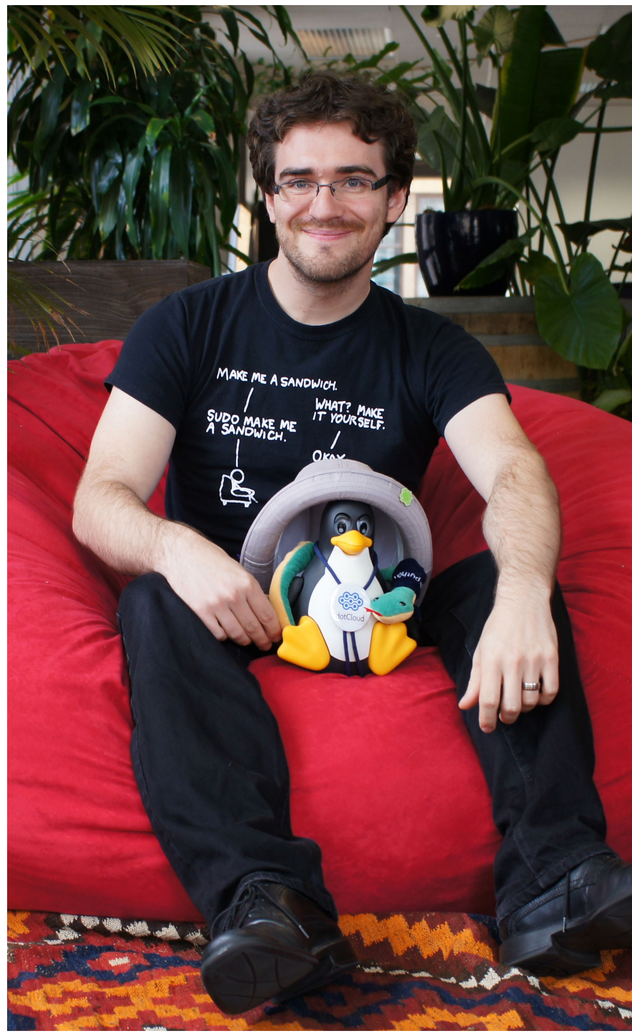
Docker in production

Jérôme Petazzoni
@jpetazzo

Docker Inc.
@docker

Jérôme Petazzoni (@jpetazzo)

- Grumpy French DevOps
  - Go away or I will replace you with a very small shell script
- Wrote dotCloud PAAS deployment tools
  - EC2, LXC, Puppet, Python, Shell, ØMQ...
- Docker contributor
  - Security, networking...
- Runs all kinds of crazy things in Docker
  - Docker-in-Docker, VPN-in-Docker, KVM-in-Docker, Xorg-in-Docker...



Outline

- Quick recap on Docker and its 1.0 release
- “Solved” problems: install, build, distribute
- Service discovery & general plumbing
- Orchestration (running many containers)
- Performance (measuring it & improving it)
- Configuration management
- Sysadmin chores: logging, backups, remote access

One-slide elevator pitch about Docker

- Docker is an Open Source engine for containers
  - build, ship, run your applications within containers (=lightweight VMs)
- Docker enables separation of concerns
  - devs put their apps in containers
  - ops run the containers
- It's (probably) one of the most active FOSS projects today
  - more than 500 contributors in the last year
  - includes major contributions from e.g. Google, Red Hat...

Docker ~~1.0~~ ~~1.1~~ 1.1.1 is here!

- Docker 1.0 released last month for DockerCon
- Random pick of recent features:
  - pause/unpause (helps to get consistent commit/snapshot)
  - SELinux (for, you know, security)
  - network superpowers with `docker run --net …`
- More importantly: it's stamped “production-ready”
  - you can buy support contracts, training...
    (in addition to the traditional t-shirts and stickers☺)

Installation

- On your dev machine: boot2docker
  - tiny VM (25 MB), works with all virtualization types
  - wrapper script (OS X only) to run docker CLI locally
  - future improvements: shared volumes with `docker run -v …`
- On your servers: which distro?
  - use something recent (Ubuntu 14.04 LTS, RHEL 7, Fedora 20...)
  - special distros: CoreOS, Project Atomic — new but promising

Build with Dockerfiles

```
FROM ubuntu:14.04
MAINTAINER Docker Education Team <education@docker.com>

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
```

Build with Dockerfiles

- Great for most purposes
  - caching system allows full rebuilds that are still fast
- Drawbacks (a.k.a. work in progress)
  - separate build/run environments
    (don't ship that 5 GB build image if you just need the 10 MB artifact)
  - entitlement, credentials, and other secrets
    (what if the build process needs to access a private repository?)
- Workarounds
  - use *two* Dockerfiles; keep Dockerfiles and images private

Distribute and ship images

- Docker Hub
  - docker push, docker pull: it's magic!
  - public and private images
  - no *on prem* version yet; but it's one of the most requested features
- Run your own registry
  - `docker run registry # “docker run -P” to expose it to LAN`
  - defaults to local storage
  - can use cloud object storage (Swift, GCE, S3, Elliptics...)

Distribute and ship images

- Hack around `docker load/save`
  - load/save works with plain tarballs
  - put them wherever you want them
  - https://github.com/blake-education/dogestry (much image, such docker, wow)
- Work in progress: pluggable transports
  - many things are *damn good* at moving diffs (git, rsync...)
  - can we borrow something from them?

Service discovery

- There's more than one way to do it
  - inject everything we need through environment
    `docker run -e DB_HOST=… -e DB_PORT=… -e …`
  - bind-mount a configuration file into the container
    `docker run -v /etc/docker/config/myapp.yaml:/config.yaml …`
  - resolve everything we need through a highly-available key-value store (zookeeper, etcd, consul...)
  - resolve everything we need through DNS (consul, skydns, skydock, dnsmasq...)

How do they compare?

Let's grade those different methods!

But first, let's look at

links

Docker links

```
docker run -d --name frontdb mysqlimage
docker run -d --link frontdb:sql webimage
```

- DNS entries are created in containers
- Environment variables are injected in 2nd container

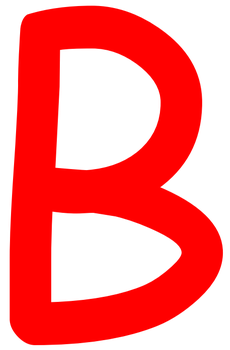
```
SQL_PORT=tcp://172.17.0.10:5432
SQL_PORT_5432_TCP=tcp://172.17.0.10:5432
SQL_PORT_5432_TCP_ADDR=172.17.0.10
SQL_PORT_5432_TCP_PORT=5432
SQL_PORT_5432_TCP_PROTO=tcp
```

- Doesn't work across multiple Docker hosts

Service discovery: environment variables

- Easy to integrate in your code
  - is there any language that does *not* support environment variables?
- Easy to setup
  - start services, lookup ports, inject variables
- Even easier with links
  - fully automatic if using only one host
- Static
  - if a service moves, cannot update environment variables

Environment variables:



Service discovery: bind-mount configuration file

- Easy to integrate in your code
  - again, is there a language without a decent JSON/YAML parser?
- Easy to setup
  - just like environment variables, but generate a file
- Kind of dynamic
  - it's possible to update the configuration files while services run
- But not really
  - services have to detect the change and reload the file

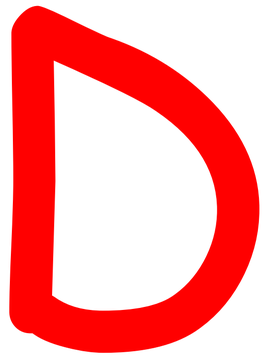
Bind-mount configuration file:

B

Service discovery: key-value store

- Harder to integrate in your code
  - HTTP requests instead of `getenv` are not too hard, but still
- Harder to setup
  - must setup the key-value store; on multiple nodes
- Kind of dynamic
  - most of those key-value stores support “watch” operation
- But not really
  - services still have to detect the change and reload the file

Key-value stores:

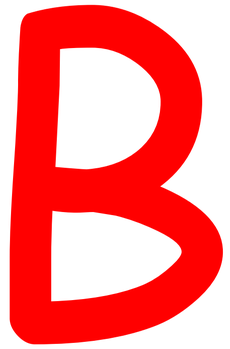


Service discovery: DNS

- Easy to integrate in your code
  - in most cases, no integration is needed at all, works out of the box
- Harder to setup*
  - must setup a DNS system that you can easily update
- Dynamic
  - you can update DNS zones, no problem
- No “push”, but...
  - services won't detect a change, but if something wrong happens (and results into a disconnection) they might re-resolve and retry

*Except on a single host, if you use links, since they automatically create DNS entries.

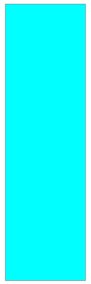
DNS:



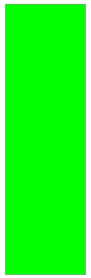
Are we doomed?

Links, take two

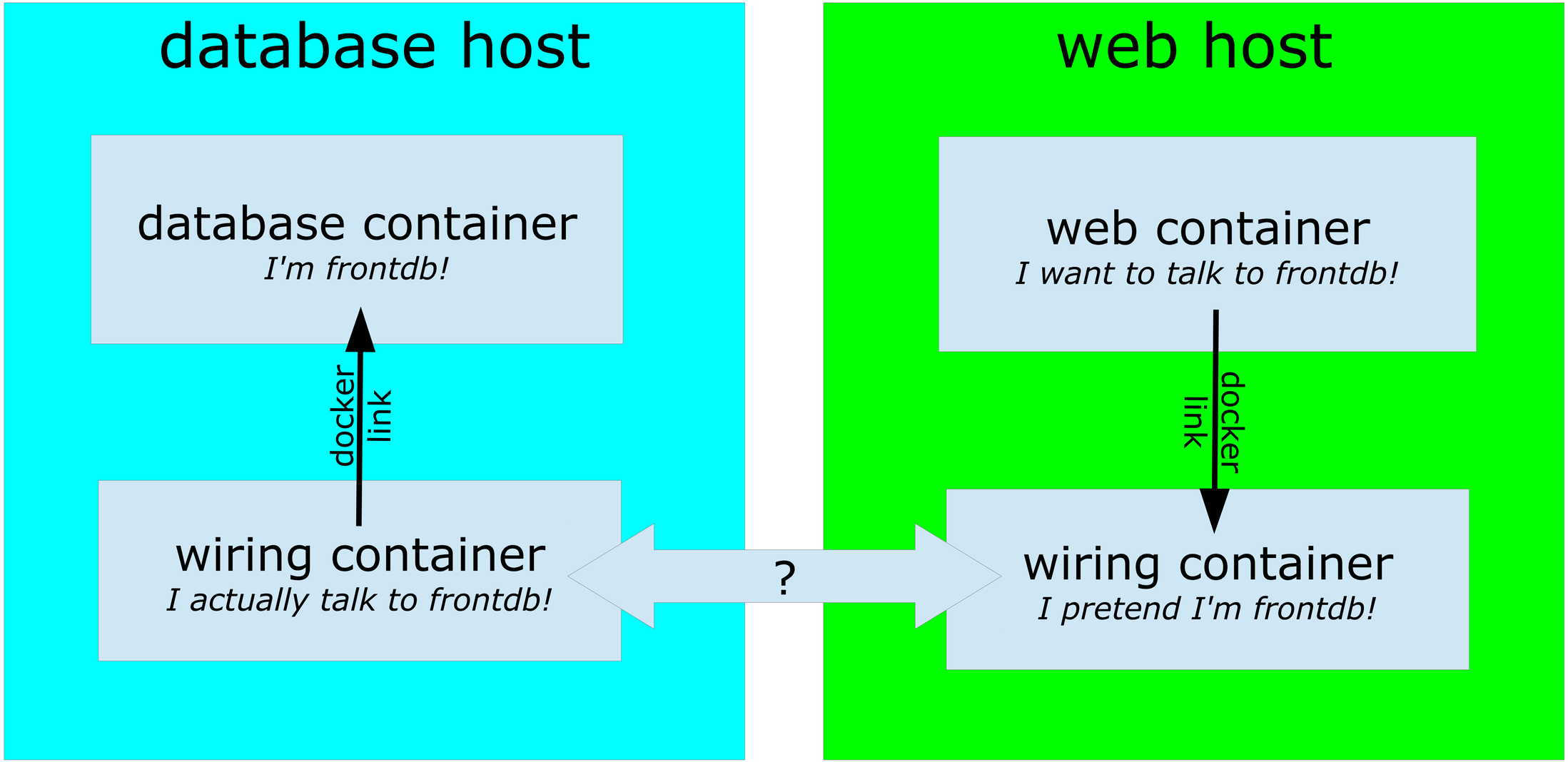
The ambassador pattern

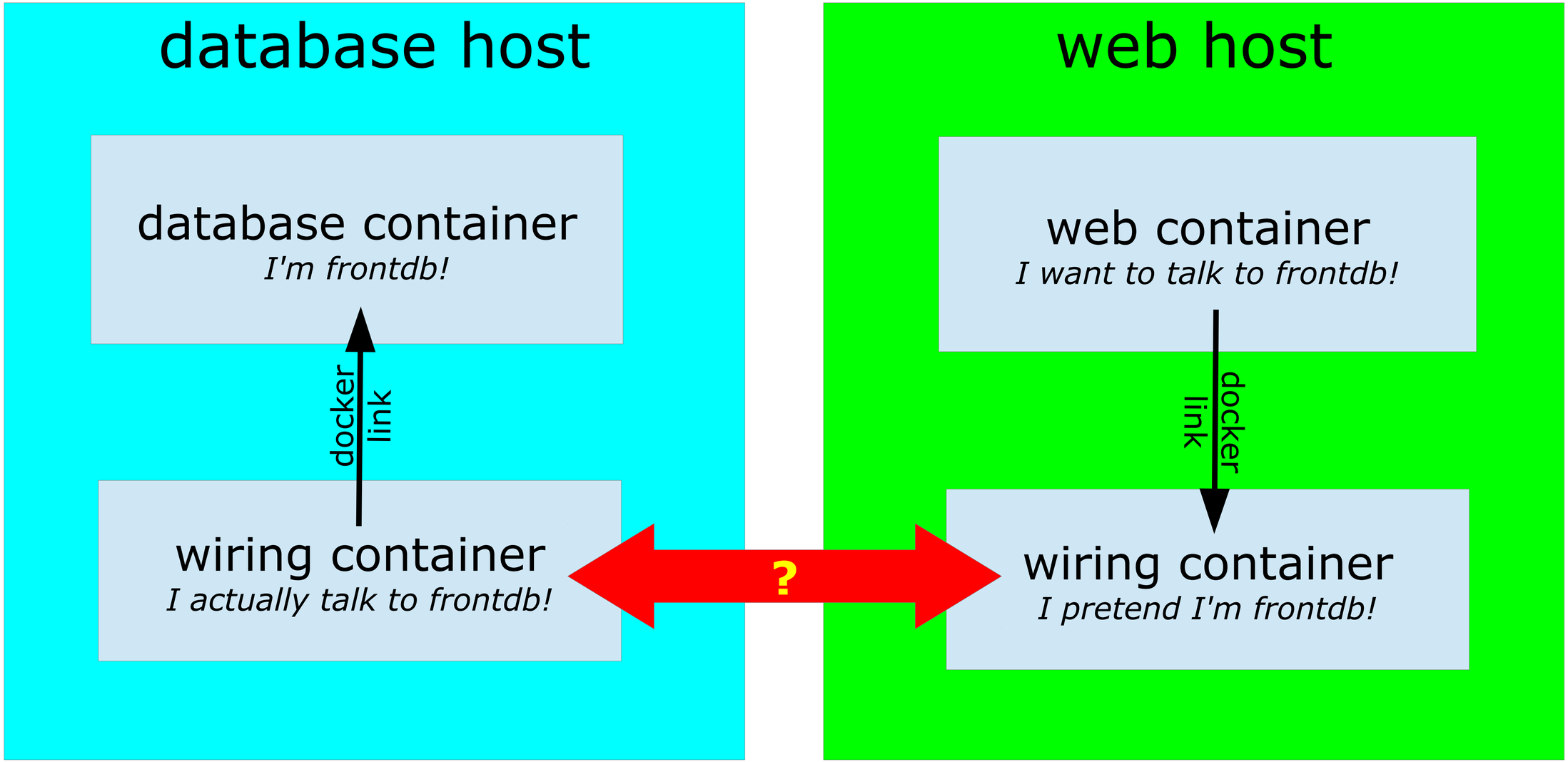


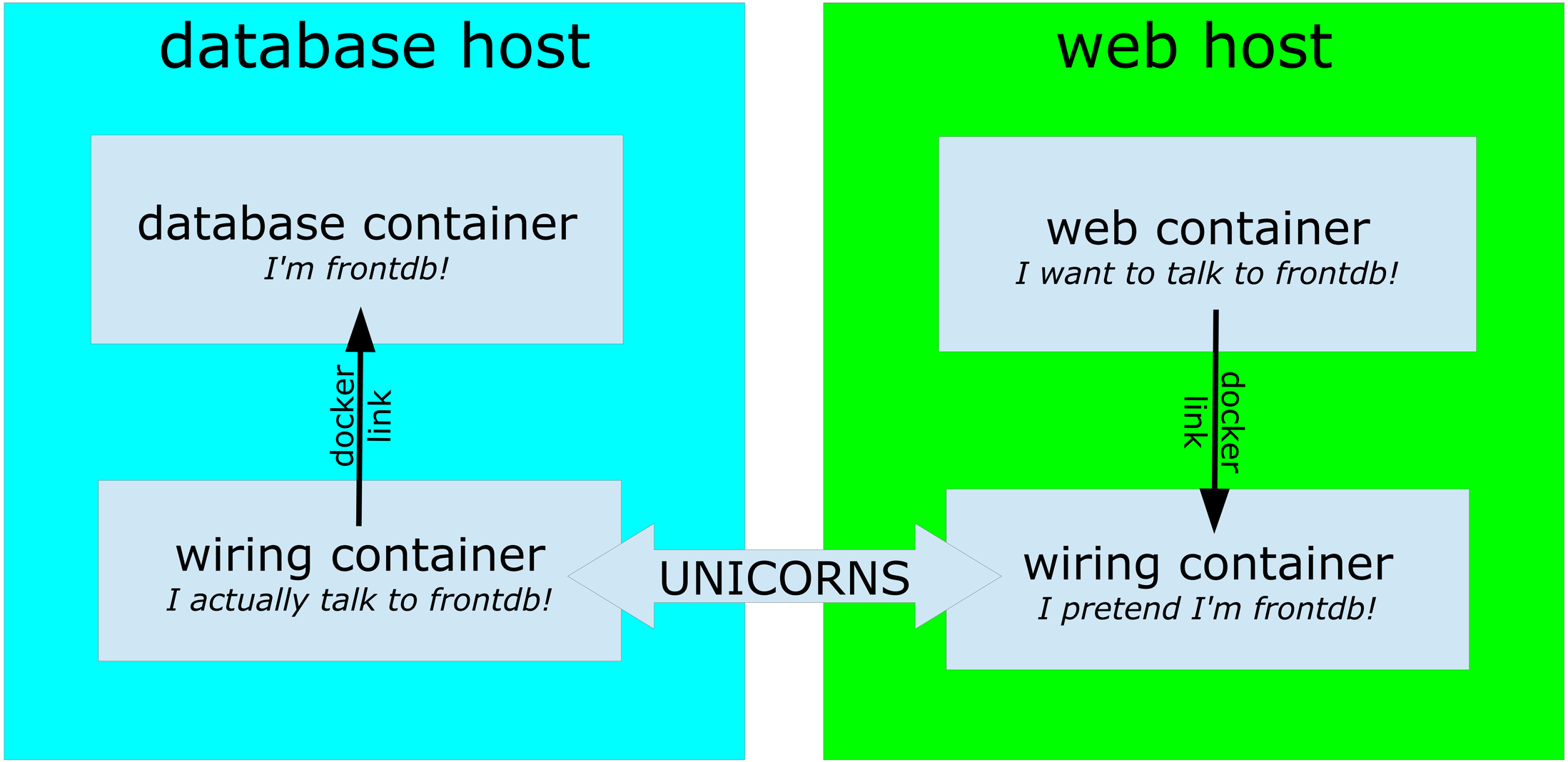
```
host 1 (database)
docker run -d -name frontdb mysqlimage
docker run -d -link frontdb:sql wiring
```



```
host 2 (web tier)
docker run -d -name frontdb wiring
docker run -d -link frontdb:sql nginximage
```


database host
database container
I'm frontdb!
docker
link
wiring container
I actually talk to frontdb!
?
web host
web container
I want to talk to frontdb!
docker
link
wiring container
I pretend I'm frontdb!


database host
database container
I'm frontdb!
docker link
wiring container
I actually talk to frontdb!
?
web host
web container
I want to talk to frontdb!
docker link
wiring container
I pretend I'm frontdb!


database host
database container
I'm frontdb!
docker link
wiring container
I actually talk to frontdb!
UNICORNS
web host
web container
I want to talk to frontdb!
docker link
wiring container
I pretend I'm frontdb!

“...Unicorns?”

- Work in progress, but you can look at:
  - Docksul
    https://github.com/progrium/docksul
  - Grand Ambassador
    https://github.com/cpuguy83/docker-grand-ambassador
- Or roll your own
  - use some highly-available key-value store (yup, they're back too!)
  - HAProxy, stunnel, iptables...

Service discovery: links with ambassadors

- Easy to integrate in your code
  - it's still environment variables
- Easy to setup in dev, harder in production
  - use normal links in dev; get the big guns out only in prod
- Dynamic
  - the ambassadors can reroute traffic if necessary

Ambassadors:

But warning: construction area

(They're still work in progress)

Orchestration

- There's more than one way to do it (again!)
  - describe your stack in files (Fig, Maestro-NG, Ansible and other CMs)
  - submit requests through an API (Mesos)
  - implement something that looks like a PAAS (Flynn, Deis, OpenShift)
  - the “new wave” (Kubernetes, Centurion, Helios...)
  - OpenStack (because OpenStack can do *everything*!)

Introducing the Docker orchestration flowchart

Do you (want to) use OpenStack?

- Yes
  - if you are building a PAAS, keep an eye on Solum
    (and consider contributing)
  - if you are moving VM workloads to containers, use Nova
    (that's probably what you already have; just enable the Docker driver)
  - otherwise, use Heat
    (and use Docker resources in your Heat templates)
- No
  - go to next slide

Are you looking for a PAAS?

- Yes
  - CloudFoundry (Ruby, but increasing % Go)
  - Deis (Python, Docker-ish, runs on top of CoreOS)
  - Dokku (A few 100s of line of Bash!)
  - Flynn (Go, bleeding edge)
  - OpenShift geard (Go)
- Choose wisely (or go to the next slide)
  - http://blog.lusis.org/blog/2014/06/14/paas-for-realists/

    "I don't think ANY of the current private PaaS solutions are a fit right now."

How many Docker hosts do you have?

- Only one per app or environment
  - Fig
- A few (up to ~10)
  - Maestro-NG
  - your favorite CM (e.g. Ansible has a nice Docker module)
- A lot
  - Mesos
  - have a look at (and contribute to) the “new wave” (Centurion, Helios, Kubernetes...)

Work in progress: `libswarm`

- Run <something> that...
  - exposes the Docker API
  - talks to real Docker hosts
  - spins Docker hosts up and down as needed
  - takes care of scheduling, plumbing, scaling...
- Use your normal client to talk to that <something>
  - it looks like a Docker host
  - but it's an elastic, scalable, dynamic, magic Docker host
- https://github.com/docker/libswarm

Performance: measure things

- cgroups give us per-container...
  - CPU usage
  - memory usage (fine-grained: cache and resident set size)
  - I/O usage (per device, reads vs writes, in bytes and in ops)
- cgroups don't give us...
  - network metrics (have to do tricks with network namespaces)

https://github.com/google/cadvisor

http://jpetazzo.github.io/2013/10/08/docker-containers-metrics/

Performance: tweak things

- There isn't much to tweak!
  - CPU: native
  - I/O: native on volumes
    (make sure that your data set etc. is on volumes)
  - memory: no overhead *if you disable memory accounting*
    (userful for HPC; probably not for everything else)
  - network: no overhead if you run with “--net host”
    (useful for >1 Gb/s workloads)
    (or if you have a high packet rate; e.g. VOIP, gaming...)

Configuration management

- There is more than one way do to it (surprise!)
- If you don't use a CM system yet, you don't have to
  - If you're familiar with a CM system, you can use it to encode small-scale deployments (up to, say, 10 nodes)
- Using CM to manage Docker *hosts* makes sense
- But Dockerfiles will be great for *apps* themselves
- If you *really* want to keep using your recipes, here's how to integrate!

Configuration management, if you want to mix VMs and containers

- Author a single generic Docker image with your favorite CM, “locked and loaded”
- When creating a container from that image, you give it its identity (certificate/node name/...)
- When the container starts, it contacts the server, which gives it its configuration (manifests, cookbooks...)
- After a moment, it will converge to desired state
- Downside: slow to converge; not 100% reliable

Configuration management, if you want to mix VMs and containers

- Author a single generic Docker image with your favorite CM, "locked and loaded"
- When creating a container from that image, you give it its identity (certificate, node name/...)
- When the container starts, it contacts the server, which gives its configuration (manifests, cookbooks...)
- After a moment, it will converge to desired state
- Downside: slow to converge; not 100% reliable

NOT RECOMMENDED

Configuration management, the “immutable infrastructure” way

- Author a single generic Docker image with your favorite CM, to be used as a base for other images
- Author other Docker images:

```
FROM me/my_base_puppet_image
ADD manifests/ /etc/puppet/manifests
RUN puppet apply --certname db1138.dystopia.io
```

- Once the image is baked, you don't have to fry it (i.e. it's ready to run without extra steps)
- Downside: build new image to make a change (can be seen as an advantage)

Configuration management, the "immutable infrastructure" way

- Author a single generic Docker image with your favorite CM, to be used as a base for other images
- Author other Docker images:

```
FROM me/my_base_puppet_image
ADD manifests/ /etc/puppet/manifests
RUN puppet apply --certname c01138.dystopia.io
```

- Once the image is baked, you don't have to fry it (i.e. it's ready to run without extra steps)
- Downside: build new image to make a change (can be seen as an advantage)

SLIGHTLY BETTER (BUT STILL KIND OF MEH)

Sysadmin chores

- Backups
- Logging
- Remote access

We all know that those are just a small sample of the many boring, necessary evil deeds that sysadmins must commit once in a while.

File-level backups

- Use volumes

```
docker run --name mysqldata -v /var/lib/mysql busybox true
docker run --name mysql --volumes-from mysqldata mysql
docker run --rm --volumes-from mysqldata mysqlbackup \
tar -cJf- /var/lib/mysql | stream-it-to-the-cloud.py
```

- Of course, you can use anything fancier than tar (e.g. rsync, tarsnap...)

Data-level backups

- Use links

```
docker run --name mysql mysql

docker run --rm --link mysql:db mysqlbackup \
mysqldump --all-databases | stream-it-to-the-cloud.py
```

- Can be combined with volumes
  - put the SQL dump on a volume
  - then backup that volume with file-level tools (previous slide)

Logging for legacy apps

- Legacy = let me write to eleventy jillion arbitrary files in /var/lib/tomcat/logs!
- Solution: volumes

```
docker run --name logs -v /var/lib/tomcat/logs busybox true
docker run --name tomcat --volumes-from logs my_tomcat_image
```

  - Inspect logs:
    `docker run --rm --volumes-from logs ubuntu bash`
  - Ship logs to something else:
    `docker run --name logshipper --volumes-from logs sawmill`

Logging for dockerized apps

- Dockerized = I only write to stdout
- Solution: Docker CLI/API

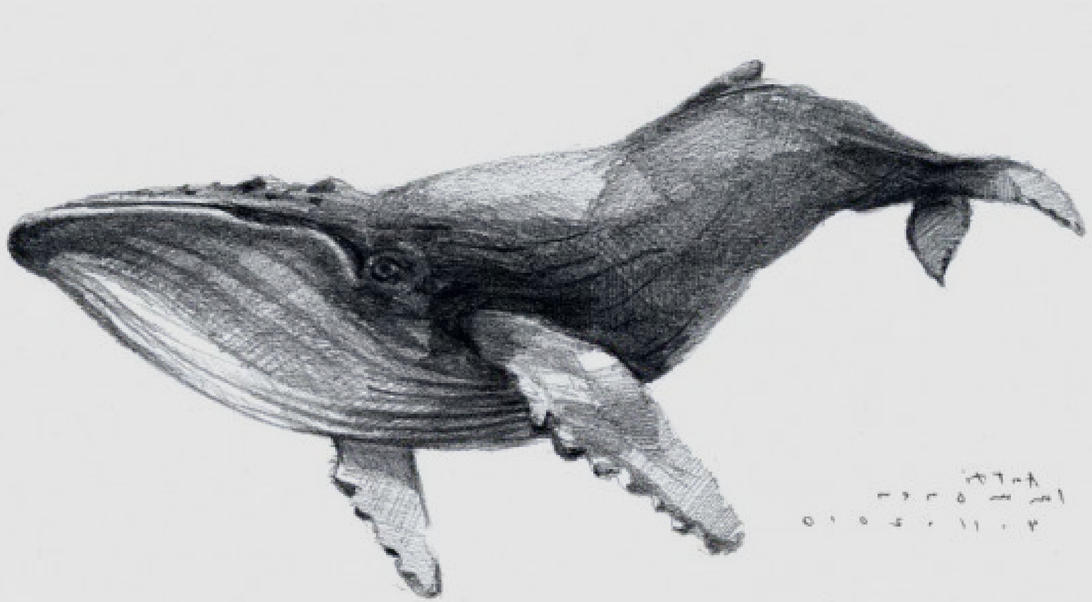
```
docker run --name tomcat dockerized_tomcat

docker logs tomcat

docker run -v /var/run/docker.sock:/var/run/docker.sock \
logshipper docker logs tomcat | pipestash ...
```

- Caveat: logs are not rotated (but PR is on the way)

Remote access

- If you own the host: SSH to host + nsenter
  https://github.com/jpetazzo/nsenter
- If you don't own the host: SSH in the container
  https://github.com/phusion/baseimage-docker
- More on that topic (“do I need SSHD in containers?”):
  http://blog.docker.com/2014/06/why-you-dont-need-to-run-sshd-in-docker/
- In the future:
  - run separate SSH container
  - log into that
  - “hop” onto the target container

Not an actual book (yet)



Docker in production

Containers, containers everywhere!

Thank you! Questions?

http://www.docker.com/
@docker
@jpetazzo

Come talk about Docker tomorrow:
- 10:40am: office hours (expo hall table A)
- evening: meet-up at New Relic

#oscon

O'REILLY® OSCON