Morning

- Introduction
- First container
  - ➢ *Hands-on! My first container*
- First images
  - ➢ *Playing with images*
- Anatomy of a container
  - ➢ *Looking under the hood*

Afternoon

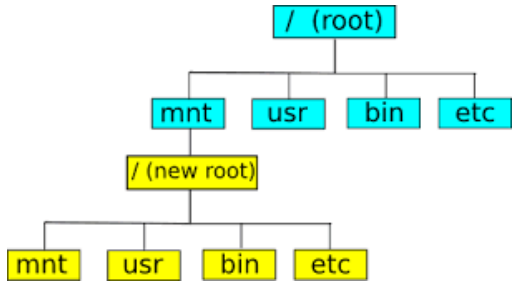
- Going into production
- Working with multiple containers
  - ➢ *Hands-on! docker-compose*
- A real application
  - ➢ *Hands-on! Building your dropbox with docker*

Docker Training Agenda

A few words on this training

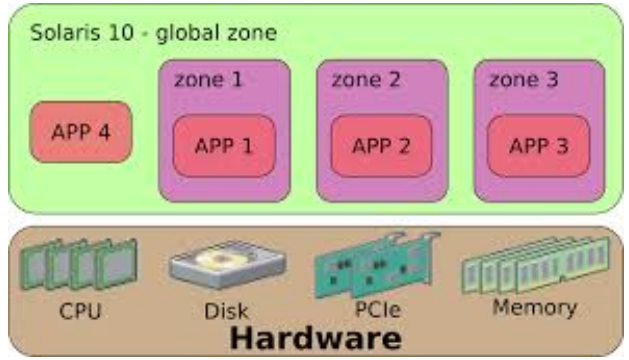
- Our experience with usual docker trainings
  - Use-case oriented
  - Focused on application deployments
- What we found out the “hard” way
  - How it works under the hood: understanding the **magic** of it
  - The minimum you need for **production**
  - The training we would have wanted as **OPS**
- **First run** of this training!
- Many thanks to Jérôme Petazzoni and Jessica Frazelle

A brief history of containers



chroot (1979, Unix v7)

Freebsd jail (2000)



Solaris zones (2004)

OpenVZ

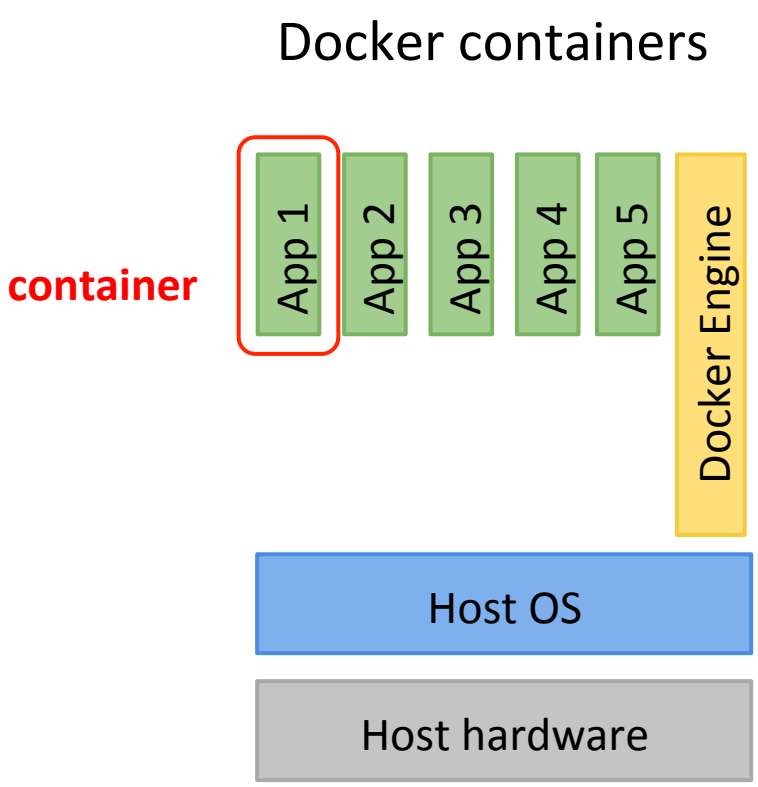
OpenVZ (2005)

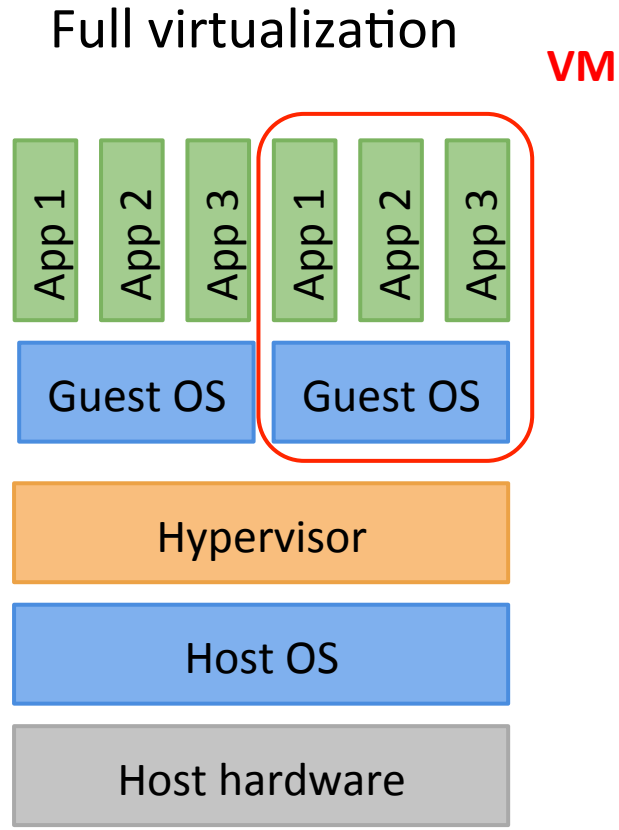
Lxc (2006)

docker

docker (2013)

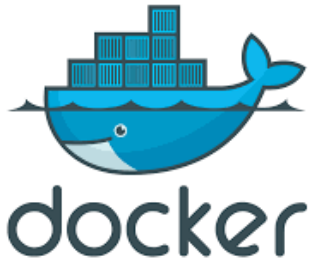
Just a small VM then?



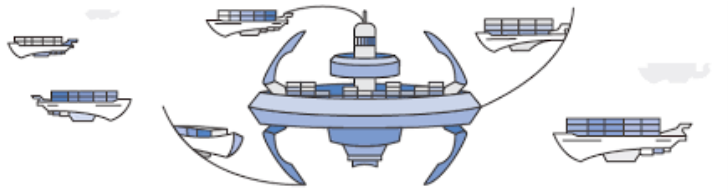


What about docker?

dotcloud (2009)



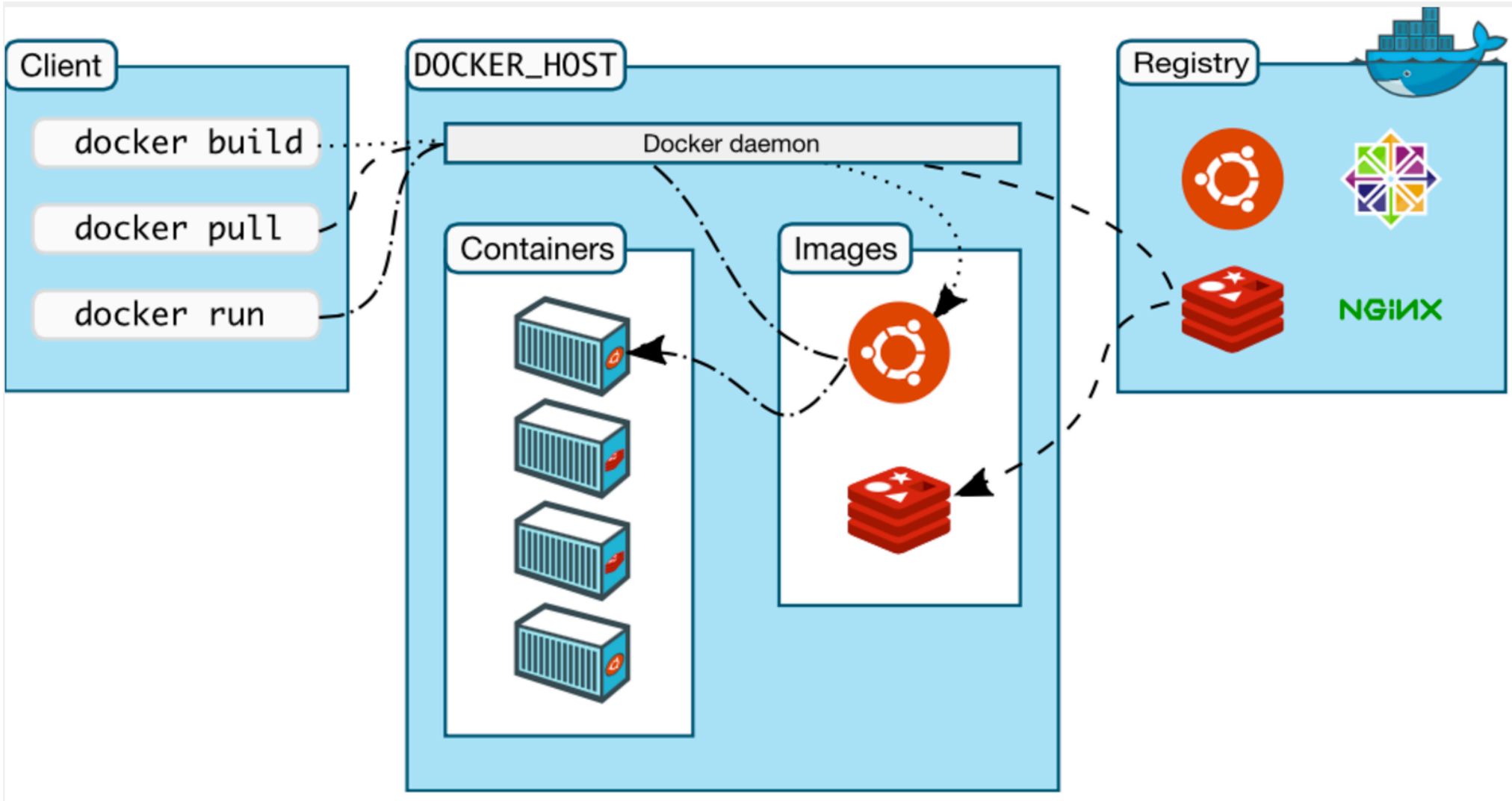
docker (2013)



Amazon ECS

Large ecosystem

Docker components



FIRST CONTAINER

Getting started: my first container

```
# docker version
Client:
 Version:      1.10.1
 API version:  1.22
 Go version:   go1.5.3
 Git commit:   9e83765
 Built:        Thu Feb 11 20:39:58 2016
 OS/Arch:      linux/amd64

Server:
 Version:      1.10.1
 API version:  1.22
 Go version:   go1.5.3
 Git commit:   9e83765
 Built:        Thu Feb 11 20:39:58 2016
 OS/Arch:      linux/amd64
```

```
# docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 1.10.1
Storage Driver: aufs
 Root Dir: /mnt/sda1/var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 0
 Dirperm1 Supported: true
Execution Driver: native-0.2
Logging Driver: json-file
```

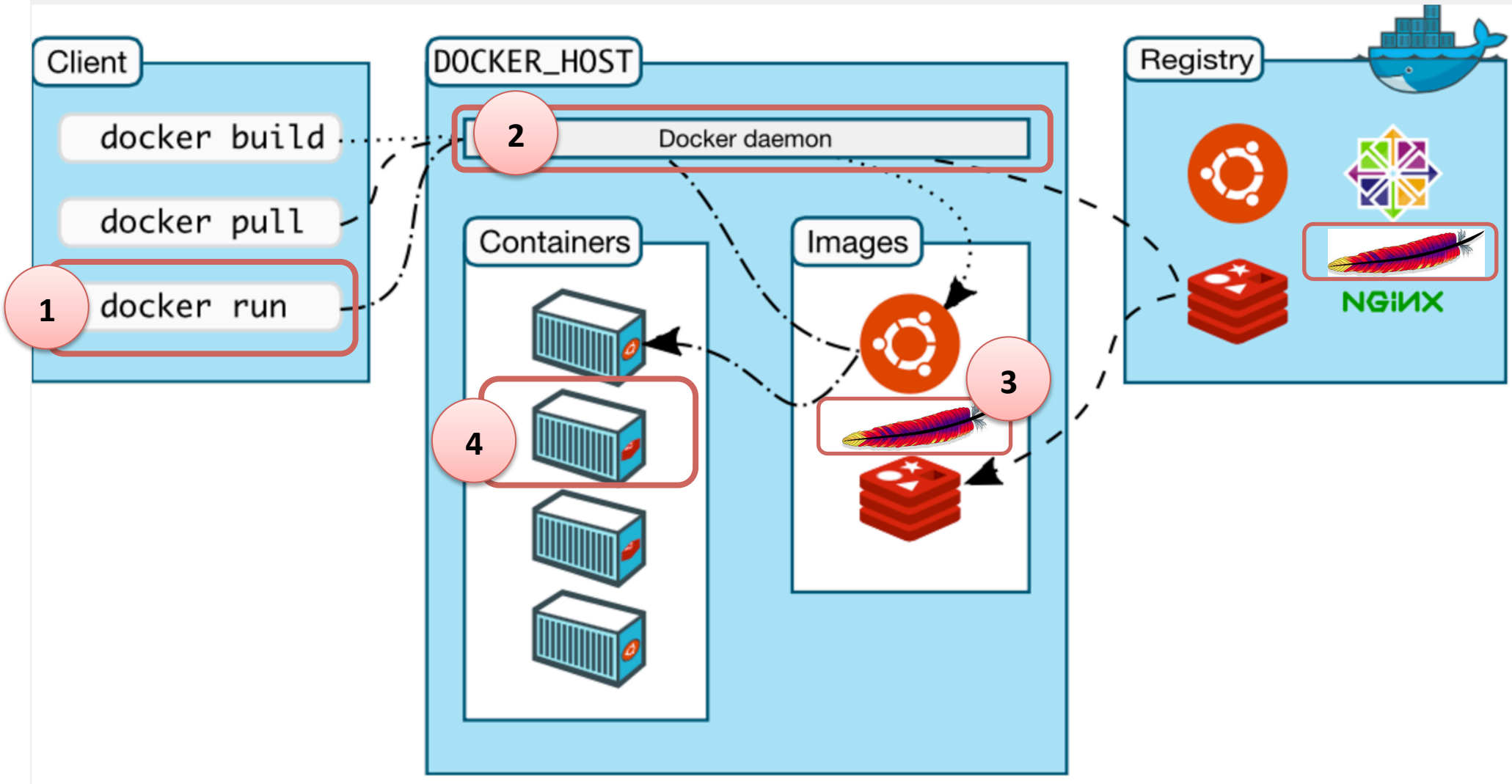
First container

```
# docker run httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
7268d8f794c4: Downloading [=====>                                             ] 5.766 MB/51.37 MB
a3ed95caeb02: Download complete
5d77cae53716: Download complete
7505df4d1bb0: Downloading [=========================>                         ]  5.98 MB/11.69 MB
24fe16d193eb: Download complete
eb1288202b50: Downloading [==============>                                    ] 1.781 MB/6.359 MB
a9172f0bee65: Waiting
```

Run in background

```
# docker  run -d httpd
028dfee1e7671b1665f174329d145c4359165261013 92d08fefc1371c7141074
```

What just happened?



With docker commands

```
# docker  images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
httpd               latest              a55bb27c1420        2 weeks ago         193.5 MB
```

```
# docker  ps
docker ps
CONTAINER ID    IMAGE     COMMAND              CREATED         STATUS         PORTS     NAMES
028dfee1e767    httpd     "httpd-foreground"   3 minutes ago   Up 3 minutes   80/tcp    determined_colden
```

```
# docker  ps -a
```

This is an inspection!

```
# docker inspect 028dfee1e767
[
    {
        "Id": "sha256:d10e4c3e7ec976f313d8f650d1c779bd5fb8783866763f12e4831510c135d8fd",
        "RepoTags": [
            "httpd:latest"
        ],
        ...
```

```
# docker inspect --format='{{.Config.Env}}' 028dfee1e767
[PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/apache2/bin
HTTPD_PREFIX=/usr/local/apache2 HTTPD_VERSION=2.4.18 HTTPD_BZ2_URL=https://www.apache.org/
dist/httpd/httpd-2.4.18.tar.bz2]
```

```
# docker inspect --format='{{.NetworkSettings.IPAddress}}' 028dfee1e767
172.17.0.2
```

Playing with the CLI

docker start	starts a container
docker stop	stops a running container
docker restart	stops and starts a container
docker pause	pauses a running container
docker unpause	unpauses a running container
docker wait	blocks until running container stops
docker kill	sends a SIGKILL to a running container

Playing with the CLI (cont'd)

docker attach	will connect to a running container
docker exec	execute a command in container
docker ps	shows running containers
docker logs	gets logs from container
docker inspect	looks at all the info on a container
docker events	gets events from containers
docker port	shows public facing port of container
docker top	shows running processes in container
docker stats	shows containers' resource usage stats

Try it with your container

```
# docker  run  --name web -d httpd
```

```
# docker  run  --name test -e var1=toto -e var2=titi ubuntu env
```

```
# docker  logs web
```

```
# docker  stop web
```

```
# docker  ps  /  docker ps -a
```

```
# docker  start web
```

```
# docker  exec web ps aux
```

```
# docker  top web
```

```
# docker  kill web
```

```
# docker  events --since 1h
```

Useful tips!

- Stop all containers
  docker stop $(docker ps -qa)

- Delete stopped containers
  docker rm $(docker ps -qa -f status=exited)
-
- Delete all containers (running or not)
  - **docker rm -f $(docker ps -qa)**

FIRST IMAGES

Let's go back to our httpd container

- Start the web server and expose port 80 (more on that later)

```
# docker  run  --name web -p 80:80 -d httpd
```

- Access the website (point your browser to the server IP, or curl)

```
# curl localhost
```

- Let's create an image with another content
  - Create a directory "web", cd into it, create an index.html file

```
<html><body><h1>Hello world!</h1></body></html>
```

  - Create a file called "Dockerfile"

```
FROM httpd
MAINTAINER laurent
ADD index.html /usr/local/apache2/htdocs
```

  - Build it, run it (stop the initial web container before), access it!

```
# docker build -t hello .
# docker  run  --name myweb -p 80:80 -d hello
```

Let’s create an image “from scratch”

- Create a dockerfile

```
FROM        ubuntu
MAINTAINER  laurent
RUN         apt-get update && apt-get install -y redis-server
RUN         sed -i "/bind/d" /etc/redis/redis.conf
EXPOSE      6379
CMD         ["/usr/bin/redis-server","/etc/redis/redis.conf","--daemonize no"]
```

- Build an image and run it

```
# docker build -t myredis .
# docker run -d --name myredis -p 6379:6379 myredis
```

- Test it

```
# sudo apt-get install -y redis-tools
# redis-cli
127.0.0.1:6379> info
127.0.0.1:6379> set hello world
127.0.0.1:6379> get hello
```

Improve this image

- Check what user is running redis

```
# docker  exec myredis ps aux
```

- Let's modify the Dockerfile

```
FROM        ubuntu
MAINTAINER  laurent
RUN         apt-get update && apt-get install -y redis-server
RUN         sed -i "/bind/d" /etc/redis/redis.conf
EXPOSE      6379
USER        redis
CMD         ["/usr/bin/redis-server","/etc/redis/redis.conf","--daemonize no"]
```

```
# docker build -t myredis.data .
# docker run -d --name myredis.data -p 6379:6379 myredis.data
```

- Run it again and see what user is running docker

CMD / ENTRYPOINT

- Let's start with a very simple image

```
# docker build -t pingctn .
# docker run -t  pingctn
```

```
FROM          ubuntu
CMD           ["ping", "8.8.8.8"]
```

- The CMD can be overriden

```
# docker run -t  pingctn ping 8.8.4.4
# docker run -it  pingctn  bash
```

- Entrypoint: create runnable images

```
# docker build -t pingctn .
# docker run -t  pingctn
# docker run -t  pingctn -c 4 8.8.4.4
# docker run -t  pingctn bash
```

```
FROM          ubuntu
ENTRYPOINT    [“ping”]
CMD           ["8.8.8.8"]
```

CMD / ENTRYPOINT: runtime configuration

- ENTRYPOINT: script.sh

```
if not configured
then
  configure (database, message broker…)
else
 run
```

- A quick example

```
# docker pull mysql
# docker run mysql
# docker run -e MYSQL_ROOT_PASSWORD=test -e MYSQL_DATABASE=testdb -p 3306:3306 mysql
```

- From another shell

```
# sudo apt-get install mysql-client -y
# mysql –u root –ptest –h 127.0.0.1 testdb
# docker inspect --format='{{.ContainerConfig.Entrypoint}}' mysql
```

A few commands for images

- List images

  docker images

- Describe image in details

  docker inspect <image>

- Remove image

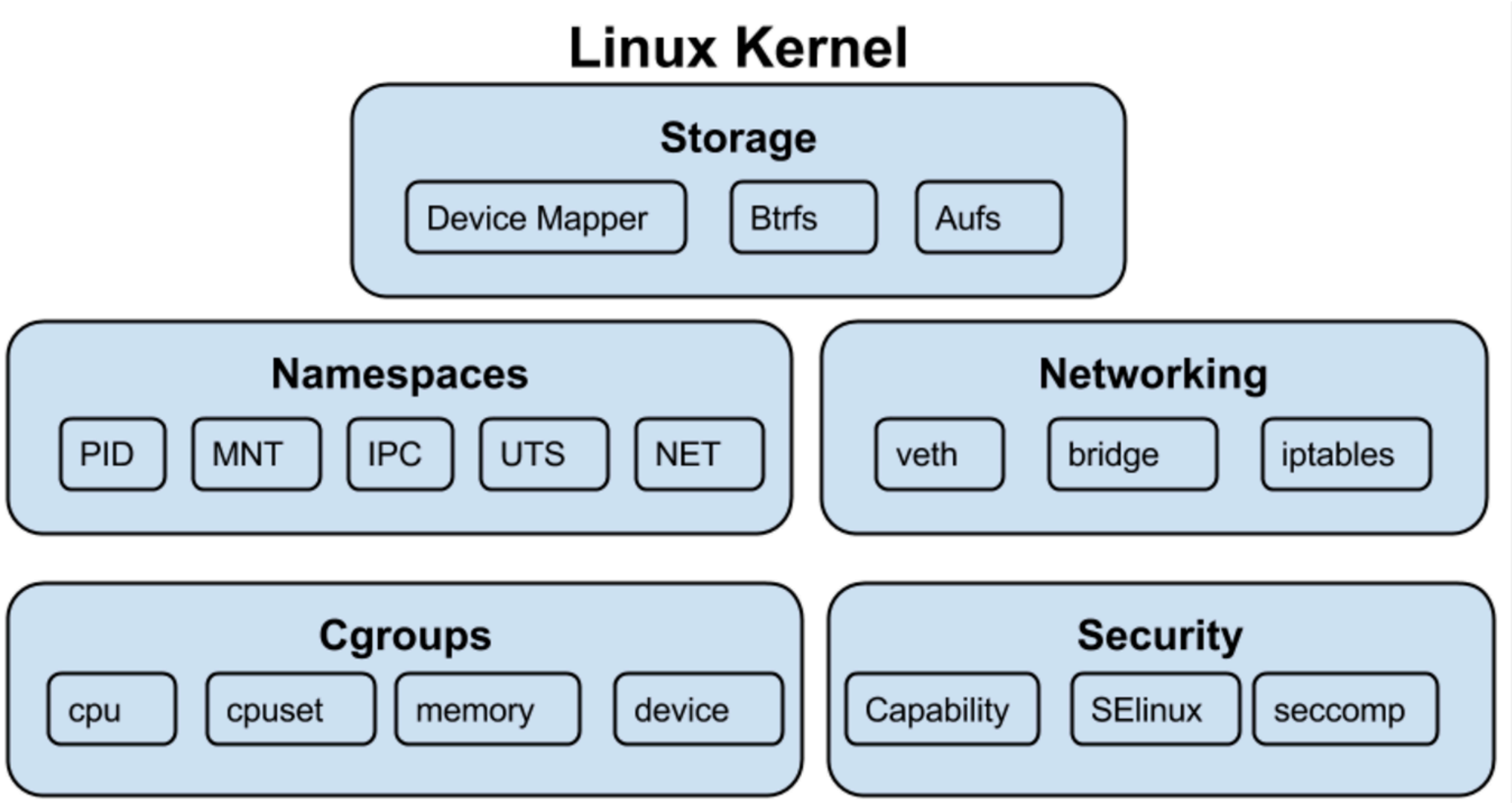
  docker rmi <image>

- Remove untagged image

  docker rmi $(docker images -f "dangling=true" -q)

ANATOMY OF A CONTAINER

Docker: bringing many kernel tools together



1- Kernel namespaces

- Isolating views of the system
- Can make the process thinks it’s the only one

mnt: mounts, filesystems

pid: processes

net: network

ipc: inter-process communications

uts: hostname

user: UIDS

Hands-on: pid namespace

- Look at processes on the host

```
pstree -a
```

- Enter the container namespace

```
docker exec -it <CONTAINER ID> /bin/bash
```

- Look at processes in the container

```
ps -edf
```

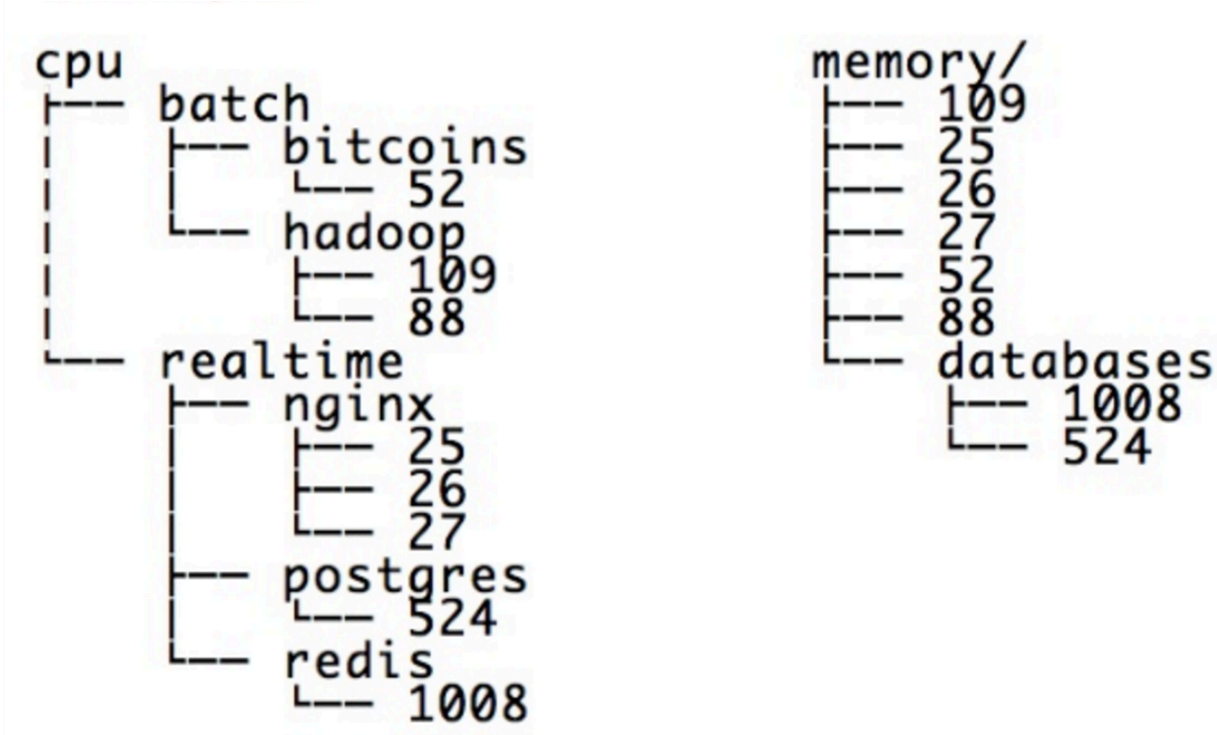
Hands-on: uts/net namespace

- UTS
  - Look at hostname in the host and in the container
- Network
  - Look at interface configuration in the host and in the container
    - ip addr
  - Look at routing configuration
    - ip route show

2- cgroups

- Built into Kernel (RHEL7/Debian/etc)
- Generically isolates resource usage (CPU, memory, disk, network)
- Guarantee resources to app/set of apps
- Can be adjusted on the fly
- Can monitor the cgroup itself

In /sys/fs/cgroup/



cgroups: hands-on

- Start an mprime container: **docker run -d pvnovarese/mprime**
- Look at container statistics: **docker stats**
- Start a second container, look at statistics
- Stop both, restart them with different **--cpu-shares** options
- Look at **docker stats**
- Look at **/sys/fs/cgroup/cpu/docker/<CID>/{cpu.shares,tasks}**
- Update cgroups limits of a live container
  - **docker update --cpu-shares xxx <CID>**

3- Security: Capabilities

- **"root"** has **all** capabilities
- a fine-grained division of "root"'s permissions for a process
- **CAP_NET_ADMIN** - modify routing tables, firewalling, NAT, etc
- **CAP_KILL** - bypass any checks for sending the kill signals
- **CAP_SYS_ADMIN** - mount, set hostname, etc
- **CAP_NET_RAW**- create a raw socket
- **CAP_SYS_TIME**- set time and data

Hands-on: Docker and capabilities

- Default ones

```
docker run --rm debian grep Cap /proc/1/status
CapInh:  00000000a80425fb
CapPrm: 00000000a80425fb
CapEff:   00000000a80425fb
CapBnd: 00000000a80425fb

capsh --decode=00000000a80425fb
```

- Privileged container (no limitation): look at the difference

```
docker run --rm --privileged debian  grep Cap /proc/1/status
```

What's happening

- Docker drops capabilities when starting the container
- Can be controlled

```
docker run -t --rm debian ping 8.8.8.8

docker run -t --rm --cap-drop=ALL debian ping 8.8.8.8

docker run -t --rm --cap-drop=ALL --cap-add=NET_RAW debian ping 8.8.8.8
```

Other security mechanisms in docker

- seccomp (by default after docker 1.10)
  - Filter system calls
  - Default profile

    https://github.com/docker/docker/blob/master/profiles/seccomp/default.json
  - Filter in addition to capabilities
    - Try to change system time from with a container (**date -s '2015-01-01'**)
    - Add capability **SYS_TIME**, try again
    - Disable seccomp filtering **--security-opt seccomp:unconfined**
    - Try again
    - You can know if seccomp is enabled for a process in /proc/self/status (Seccomp line)
- Need more control?
  - Which files can be read/written?
  - Which files can executed?
  - ➢ **Apparmor / selinux**

A (very) quick apparmor example

```
profile docker-test {
  file,
  audit deny /etc/** wl,
  audit deny /usr/bin/top x,
}
```

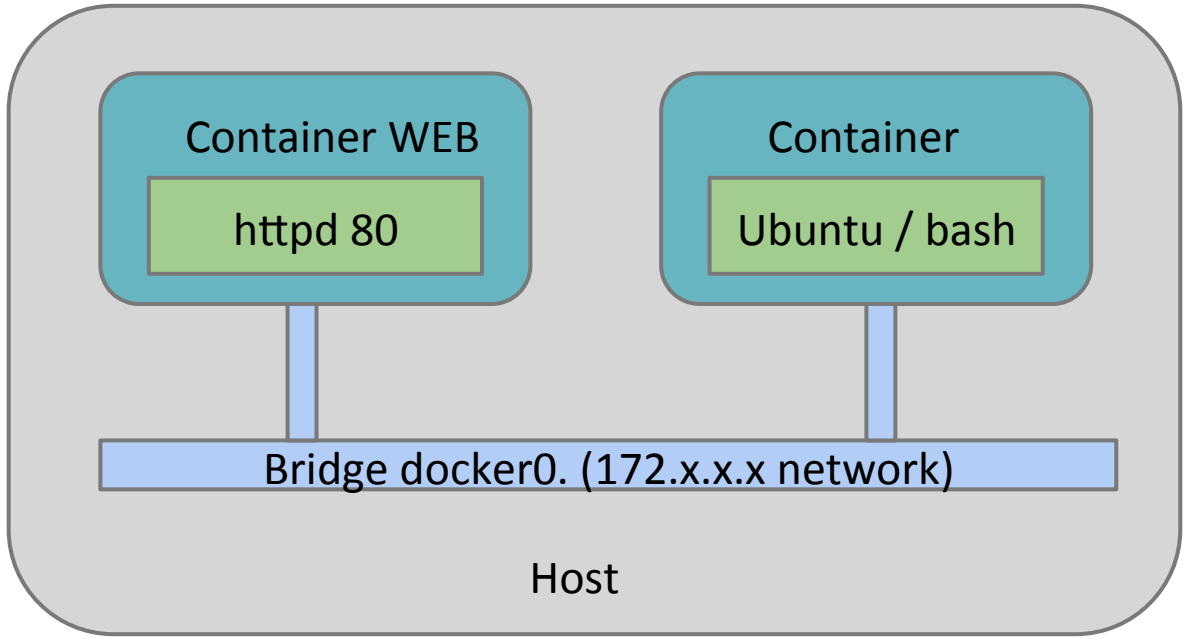
Deny writing/linking in /etc

Deny execution of top

- Let's try it!
- Load apparmor profile: **sudo apparmor_parser -r docker-test.apparmor**
- You can check what apparmor is used for with **sudo aa-status**
- Start a container with this profile:

  docker run -it --rm --security-opt="apparmor:docker-test" ubuntu /bin/bash
- Try writing in /etc or using top
- On the host, look at apparmor logs (in **/var/log/syslog**)

4- Networking



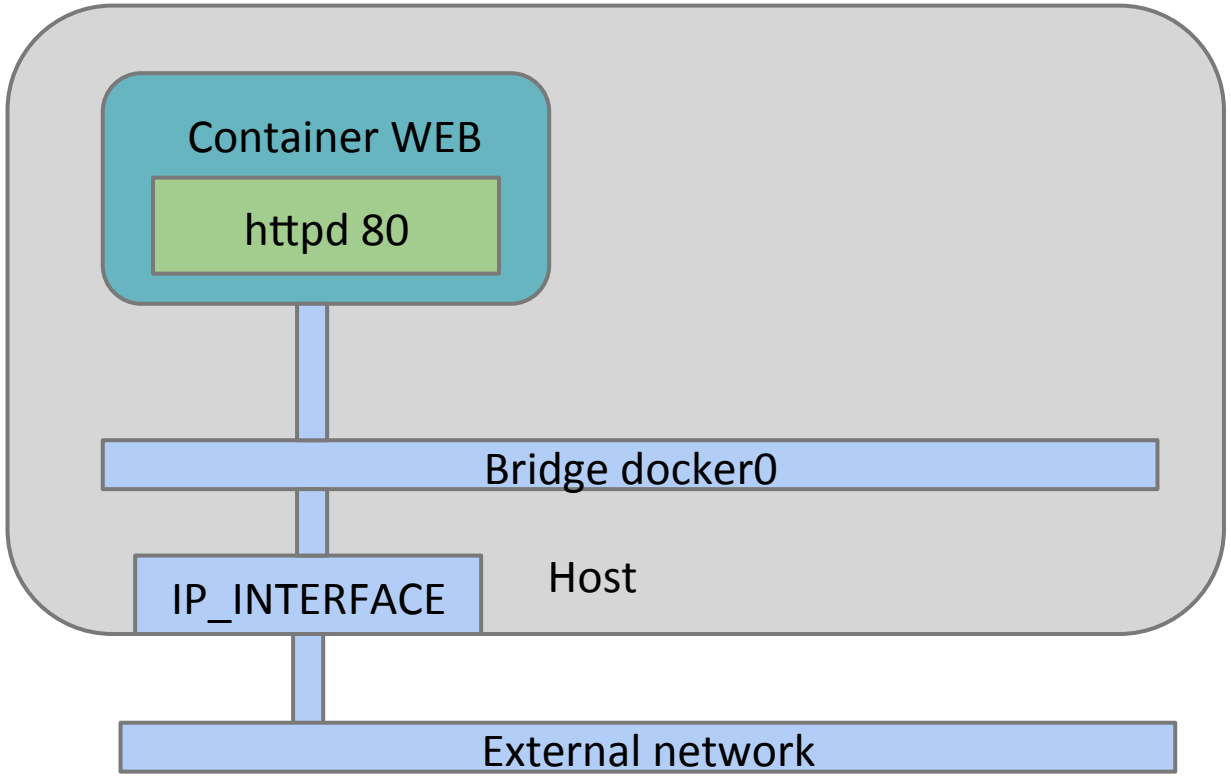
- Let's try it!

  Run an httpd container: **docker run -d --name web httpd**

  Get its IP: **docker inspect --format='{{.NetworkSettings.IPAddress}}' web**

  Start a new interactive container, ping the first one and retrieve a web page (curl/wget)

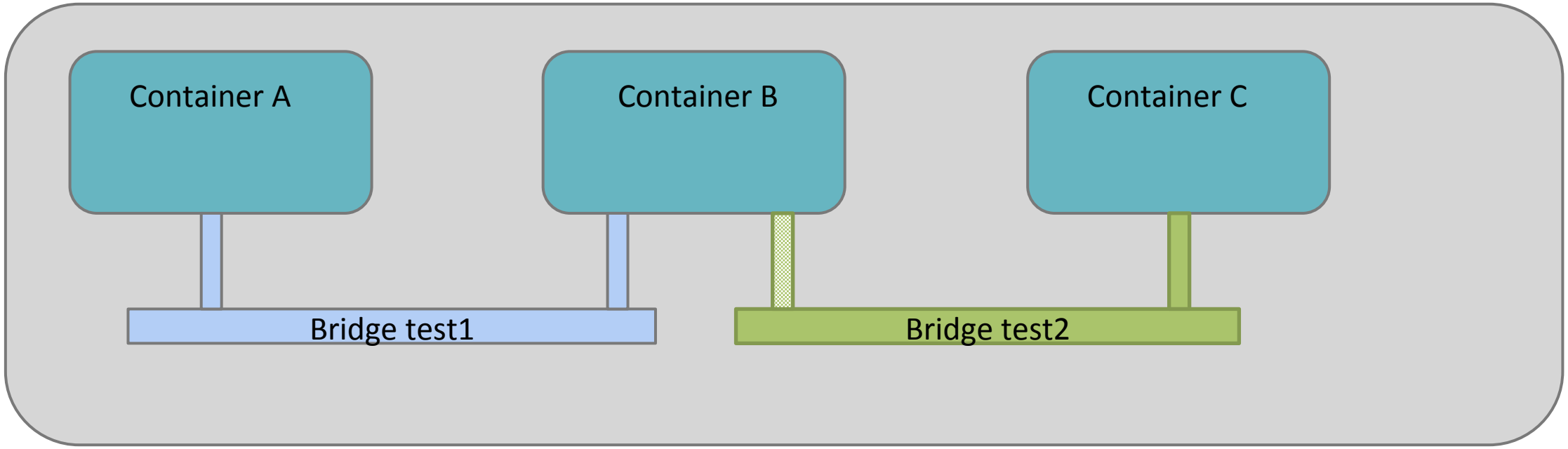
Accessing containers from the host



- Let's try it!

  Run an httpd container and forward port 80: **docker run -d --name web -p 80:80 httpd**
  Connect to your VM on port 80 (curl / web browser)
  Look at iptables NAT configuration! (**sudo iptables -t nat -nL**)
  Stop the container an look again

Slightly more complex networks



Create 2 networks: **docker network create test<X>**

Start A**: docker run --rm -it --name ca --net test1 ubuntu /bin/bash**

Start B on test1 and C on test2. Check for connectivity (ping cb from ca…)

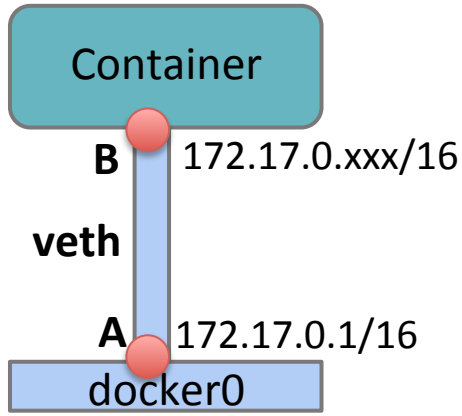
Connect B on test2: **docker network connect test2 cb**

Look at connectivity again

OK but how does this work?

- Look at **ip link** in a container and on the host
- Let's build this
  - Create a container without a network

  docker run --rm -it --net=none --privileged ubuntu /bin/bash
  - Look at interfaces (ip link, ping 8.8.8.8)



On the host

```
sudo ip link add A type veth peer name B
sudo brctl addif docker0 A
sudo ip link set A up
```

Create a "pipe" between interfaces A and B
Add one endpoint to the docker bridge

```
docker inspect -f '{{.State.Pid}}' <CID>
sudo ip link set B netns <PID>
```

Put the second endpoint in the container namespace

In the container

```
ip link set B up
ip addr 172.17.0.xxx/16 dev B
ip route add default via 172.17.0.1
```

Configure the interface in the container

How useful can this be?

- Let's create an httpd container

  docker run -d --name web -p 80:80 httpd

- Let's create a second container in the **same** netns

```
docker run -it --rm --net="container:web" ubuntu /bin/bash

In the container
apt-get update && apt-get install tcpdump –y
tcpdump –i eth0 “port 80”
```

From the (or your laptop) access the website

Quick lab on docker security & network

- Create an image
  - Based on ubuntu
  - With tcpdump installed
  - Running as a non-root user
  - Giving NET_RAW capability to tcpdump: **setcap cap_net_raw=ep /usr/sbin/tcpdump**)

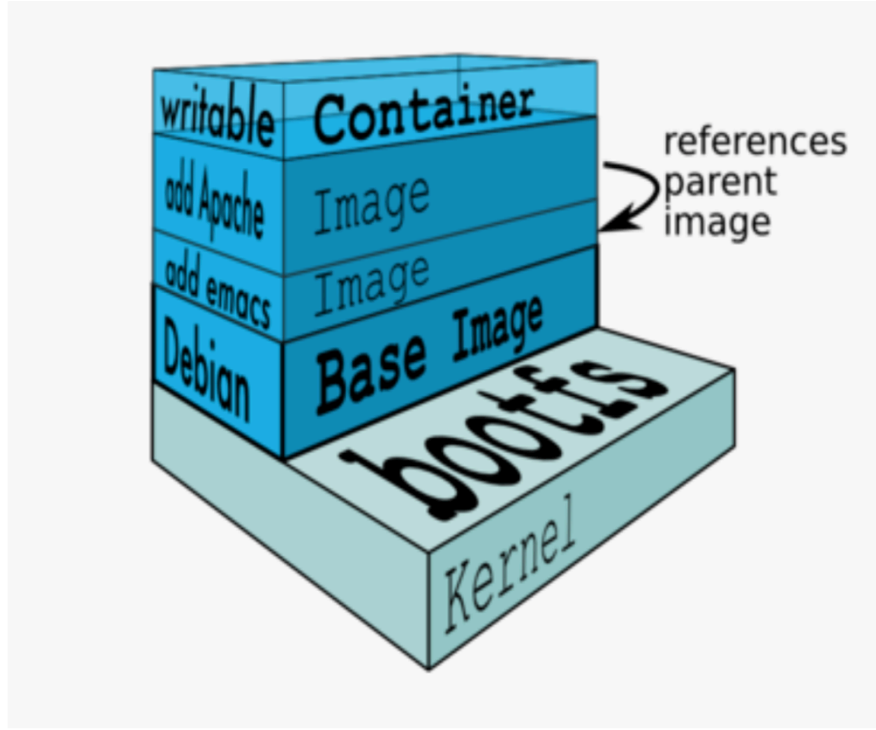
- Then run a container
  - With capabilities limited to the maximum (NET_RAW)
  - Where the filesystem is readonly
  - Sharing the network NS of a target container

```
profile docker-tcpdump {
[…]
 capability net_raw,
  network inet,
  network raw,
}
```

5- Storage

Docker images

- **Templates** from which containers are created
- **Layered** using union filesystems
- Each change to the system is a layer
- Typically created with Dockerfiles/instructions
- Stored in a docker registry (public/private)



Storage drivers

- **AUFS**
- **Device mapper**
- **OverlayFS**
- **BTRFS**

Looking at images and layers

List images

```
ubuntu@ip-172-30-0-191:~/web$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello               latest              c236574ec3e9        4 seconds ago       193.3 MB
```

Looking at image layers: **docker history**

```
ubuntu@ip-172-30-0-191:~/web$ docker history hello
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
c236574ec3e9        10 seconds ago      /bin/sh -c #(nop) ADD file:d3e5652259b7abbff0   48 B
91776fed0faa        11 seconds ago      /bin/sh -c #(nop) MAINTAINER laurent            0 B
d10e4c3e7ec9        13 days ago         /bin/sh -c #(nop) CMD ["httpd-foreground"]      0 B
<missing>           13 days ago         /bin/sh -c #(nop) EXPOSE 80/tcp                 0 B
<missing>           13 days ago         /bin/sh -c #(nop) COPY file:f465a45ed4146a281   135 B
<missing>           13 days ago         /bin/sh -c buildDeps='   ca-certificates   cu   27.16 MB
```

Understanding layers and union file system

- Build an image
  - From ubuntu
  - Which creates a 100MB files somewhere

    RUN dd if=/dev/zero of=/data bs=1 count=0 seek=100M
  - Then removes it

    RUN rm /data
- Look at image size and layer history
  - ➢ **Don’t forget it when building images**
- Layers are present as directories on disk

```
root@ip-172-30-0-191:/home/ubuntu# ls -lrt /var/lib/docker/overlay/
total 108
drwx------ 3 root root 4096 Mar 16 14:24 67b73c14720c527fef8b951168da0c5f5f96add8de5f408983a6b1d521d3d823
drwx------ 3 root root 4096 Mar 16 14:24 2b882ec6eef1c78e4ec1e4597c58819c36f8d454969e7e0b8b409b6ebbf246a7
drwx------ 3 root root 4096 Mar 16 14:24 7e045673fd45fb33f5d3ca53184d3e08e2b5fd9a4c5865bcd1ce99e48ead1a5e
drwx------ 3 root root 4096 Mar 16 14:24 d2a424c63b21cdd84c852ba92e55048ddf0f09a5b10410028aed6c6741087976
```

Data persistency

- Let's play with a redis container

```
# docker run -d --name myredis -p 6379:6379 myredis
# redis-cli
127.0.0.1:6379> set hello world
127.0.0.1:6379> get hello

# docker stop myredis && docker start myredis
# redis-cli
127.0.0.1:6379> get hello

# docker stop myredis && docker rm myredis
# docker run -d --name myredis -p 6379:6379 myredis
# redis-cli
127.0.0.1:6379> get hello
```

Create a redis container, add data

Restart the container, check data

Delete the container
Recreate "it", check data

Docker volumes

- *Directories on the host mounted in the container*
- *Not union file systems (faster)*
- *Persisted when container is deleted*

- Let's create a volume and start a container using this volume

```
# docker volume create --name redisdata
# docker run -d --name myredis -v redisdata:/var/lib/redis -p 6379:6379 myredis
```

- Use redis-cli to put data in redis, stop, rm and recreate the container.
- Look at the data. Where is it?

```
# docker volume inspect redisdata
# sudo ls -l /var/lib/docker/volumes/redisdata/_data
```

Docker volumes

- Info on volumes

```
# docker volume ls
```

- Another option: bind mount a chosen directory

```
# mkdir /tmp/data
# chmod 777 /tmp/data
# docker run -d --name myredis -v /tmp/data:/var/lib/redis -p 6379:6379 redis
```

- Create data, stop the container, look at /tmp/data

GOING INTO PRODUCTION

Docker logs

- By default logs remain within the daemon

```
# docker run -d --name web httpd
# docker logs web
```

- The log driver can be reconfigured on the daemon
  - Edit **/etc/systemd/system/docker.service**
  - Add the log-driver option: **--log-driver=syslog**
  - Reload systemd unit files and restart docker

```
# sudo systemctl daemon-reload
# sudo systemctl restart docker
```

  - Restart the container and look at syslog

```
# docker start web
# grep docker/<CID> /var/log/syslog
```

- Several log drivers (cloudwatch logs, splunk,fluentd)

Control container restart

- By default containers are never restarted

```
# docker run -d --name web httpd
# sudo systemctl restart docker
# docker ps
```

- This can be controlled when starting the container

```
# docker run -d --restart always --name web httpd
# sudo systemctl restart docker
# docker ps
# docker exec web kill 1
```

- Another option: on-failure. Try the same commands

```
# docker run -d --restart on-failure--name web httpd
```

```
# docker run -d --restart on-failure:5  ubuntu /bin/bash -c "echo hello ; exit 1”
# docker ps –a
# grep docker /var/log/syslog
```

Monitor containers

- Simple docker commands
  - Start a few containers

```
# docker run -d --name webX httpd
```

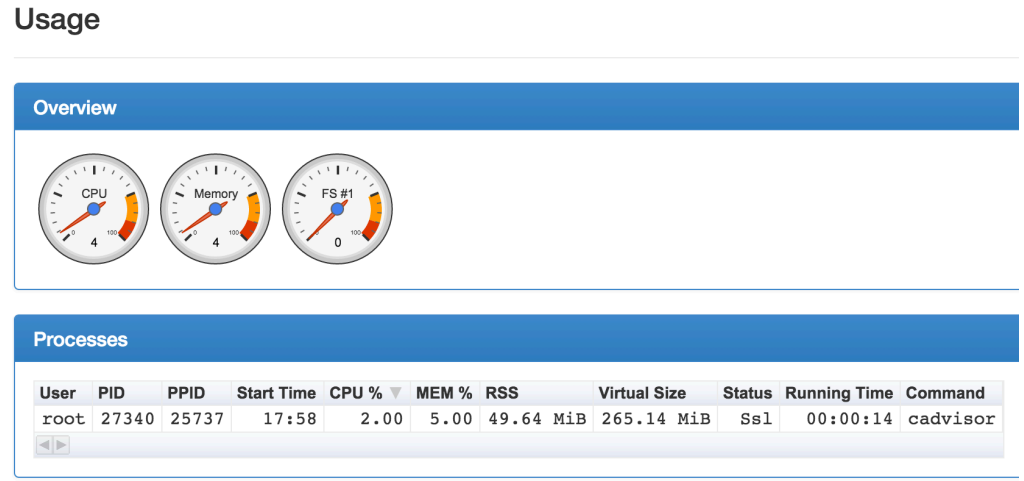
  - docker stats
  - docker top web1
  - docker events --since 1h
  - ➢**Very limited**
- For production, you need monitoring
  - Everything can be accessed with docker API

cAdvisor (from google)

- https://github.com/google/cadvisor

```
# docker run -v /:/rootfs:ro -v /var/run:/var/run:rw -v /sys:/sys:ro
  -v /var/lib/docker/:/var/lib/docker:ro -p 8080:8080
  -d --name=cadvisor google/cadvisor:latest
```

- Go to
  - http://yourserver:8080/
  - http://yourserver:8080/docker
  - http://yourserver:8080/api/v1.3/containers

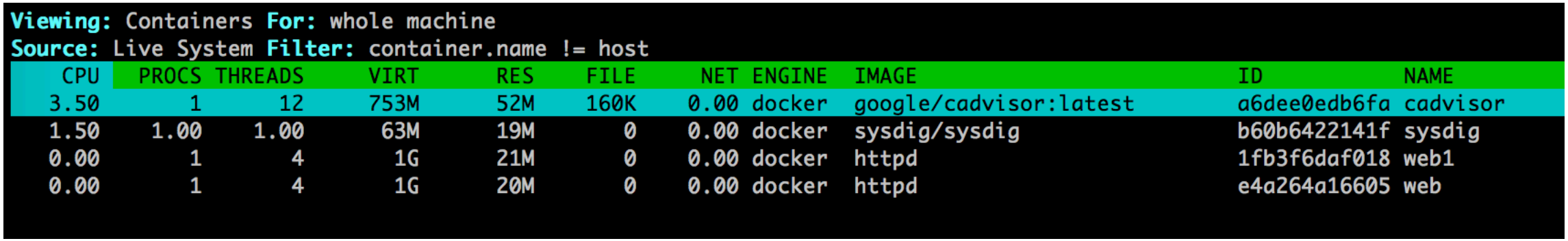


- Not very powerful on its own but can aggregated

sysdig

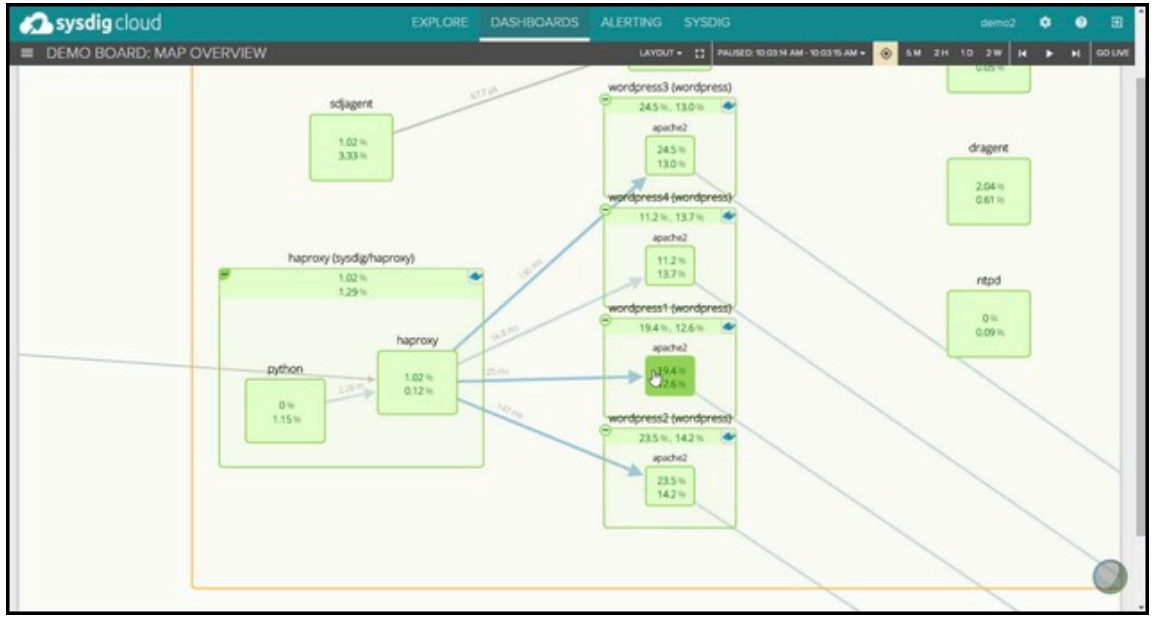
- open source, system-level exploration
- ~ strace + tcpdump + htop + iftop + lsof + …
- Great visibility for containers
- Let's try it
  - Start a sysdig container
  - Go to view/containers
  - Dig into a container

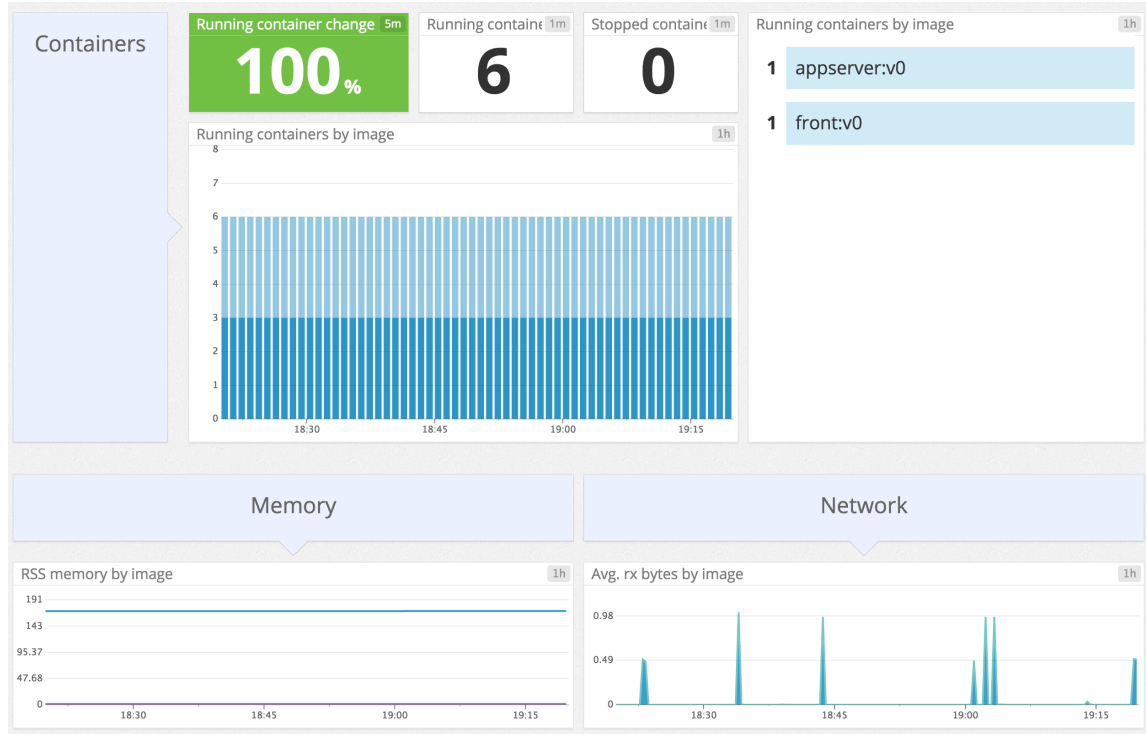
```
# docker run -i -t --name sysdig --privileged          \
   -v /var/run/docker.sock:/host/var/run/docker.sock \
   -v /dev:/host/dev -v /proc:/host/proc:ro          \
   -v /boot:/host/boot:ro                            \
   -v /lib/modules:/host/lib/modules:ro              \
   -v /usr:/host/usr:ro sysdig/sysdig csysdig
```



Monitoring centralization

- You also need to collect metric centrally
  - heapster (google, used for kubernetes)
  - InfluxDB + graphana (or other)
  - sysdig cloud (SAAS)
  - datadog (SAAS)





Go to production now?

- Single image in DEV and production
  - Docker is part of the CI build
  - Doing the same without docker doubles the work
- How?
  - Scheduling is not ready yet
  - Static assignment is OK
    - Ansible
    - User-data
    - Local docker-compose

CONTAINER ORCHESTRATION

docker-compose

- YAML file describing set of containers

```yaml
version: '2'
services:
  web:
    image: httpd
    ports:
    - "80:80"
  db:
    image: mysql
    environment:
    - MYSQL_ROOT_PASSWORD=password
    - MYSQL_DATABASE=mydb
    ports:
    - 3306:3306
```

Container with its parameters

- Create a docker-compose.yml file with this content

```
# docker-compose up -d
# docker-compose ps
# docker-compose stop web
# docker ps -a
```

A “richer” example

- Build a php+mysql image

```
FROM php:5.6-apache
RUN docker-php-ext-install mysqli
```

```
# docker build –t phpmysql .
```

- Write some php code
  - Create a code directory
  - Create an index.php file
- Modify your compose file

```
web:
  image: phpmysql
  ports:
  - 80
  volumes:
  - ./code:/var/www/html
```

- Put data in the mydb database

```
mysql> create table mytable (hello varchar(20));
mysql> insert into mytable  values ("world");
```

- Test

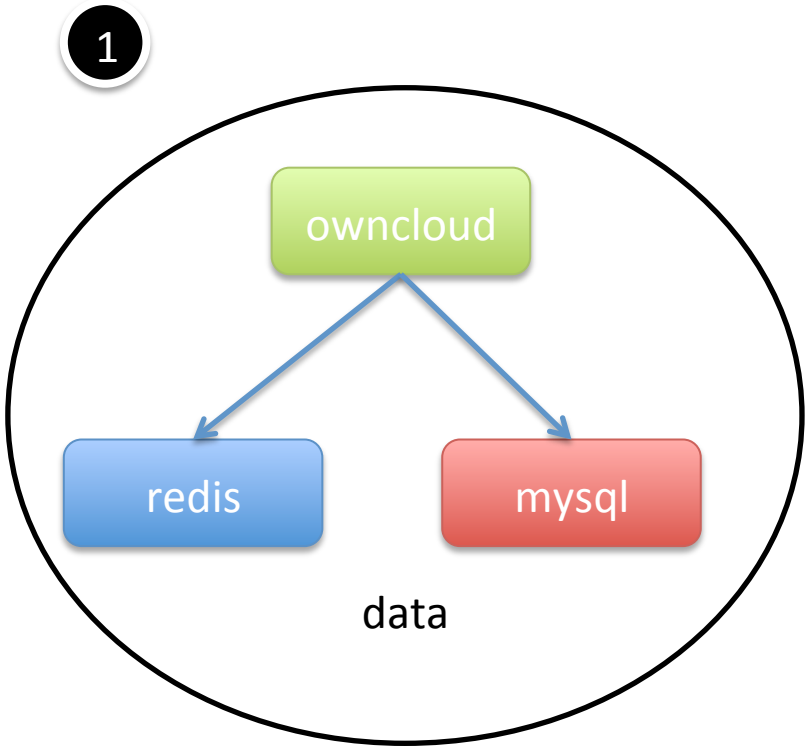
```
# curl localhost:xxxx
```

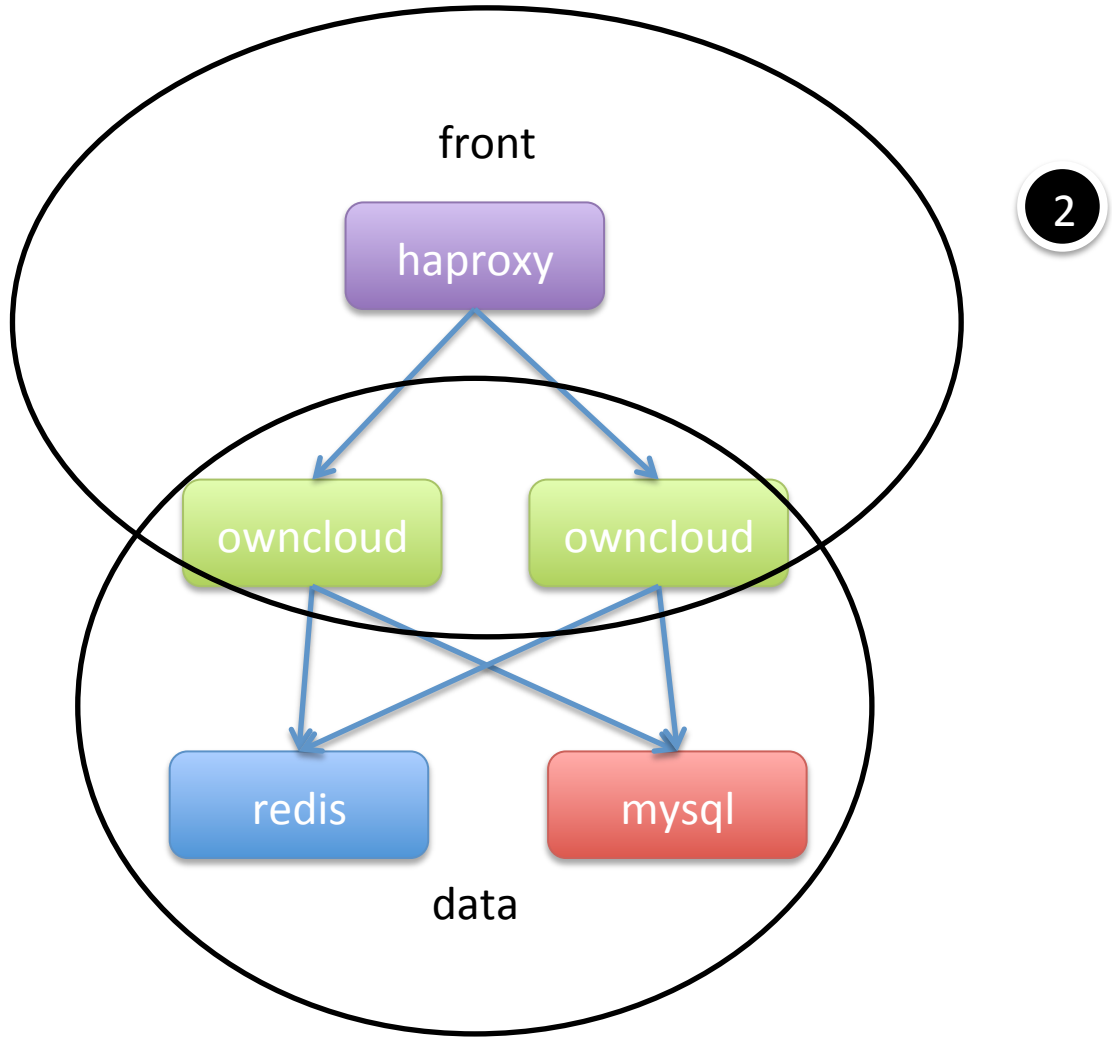
```
# docker-compose scale web=2
```

```php
<?php

$sql="select hello from mytable";

$conn = new mysqli("db","root", "password","mydb");
// check connection
if ($conn->connect_error) {
    echo 'Unable to connect to DB. Error: '   . $conn->connect_error;
    exit();
}

$rs=$conn->query($sql);

if($rs === false) {
    echo "Unable to retrieve data: ".$conn->error;
} else {
    $rs->data_seek(0);
    $row = $rs->fetch_row();
}

?>

<html>
<body>
<h2>
</h2>
<p>Content of the first table row: <?php echo $row[0]; ?></p>
</body>
</html>
```

LAB OWNCLOUD

Build your owncloud





CONCLUSION AND PERSPECTIVES

What we did not see today

- docker registry
  - Pushing
  - Creating a registry
- docker-machine
- docker-swarm
- docker universal control plane
- Many other things (ecosytem)

Conclusion and perspectives

- Containers will play a major role in the coming year
- Containers are starting to be production ready
  - ecosystem is getting richer
  - Instrumentation and security have come a long way
- Still young
  - docker network / volume appeared in 1.9, 1.10
  - docker-compose format changed in 1.6
- Some problems are not really solved yet
  - Multi-host orchestration (Kubernetes, ECS, Swarm, Mesos)
  - Multi-tenancy