

Docker - CI

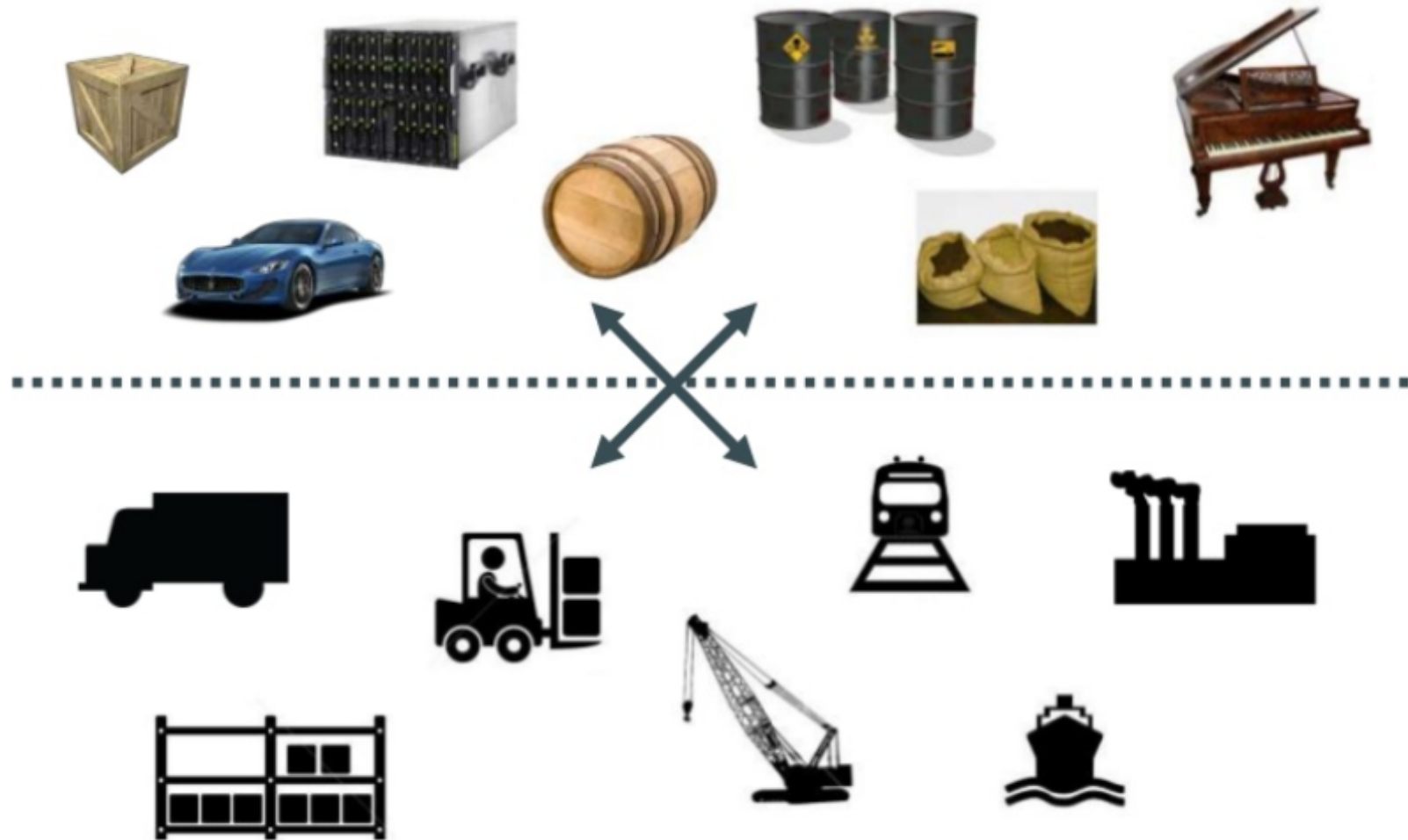
Ecole IN2P3, 2015/10/01

Sebastien Binet
CNRS/IN2P3








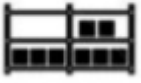





Docker origins

The container revolution

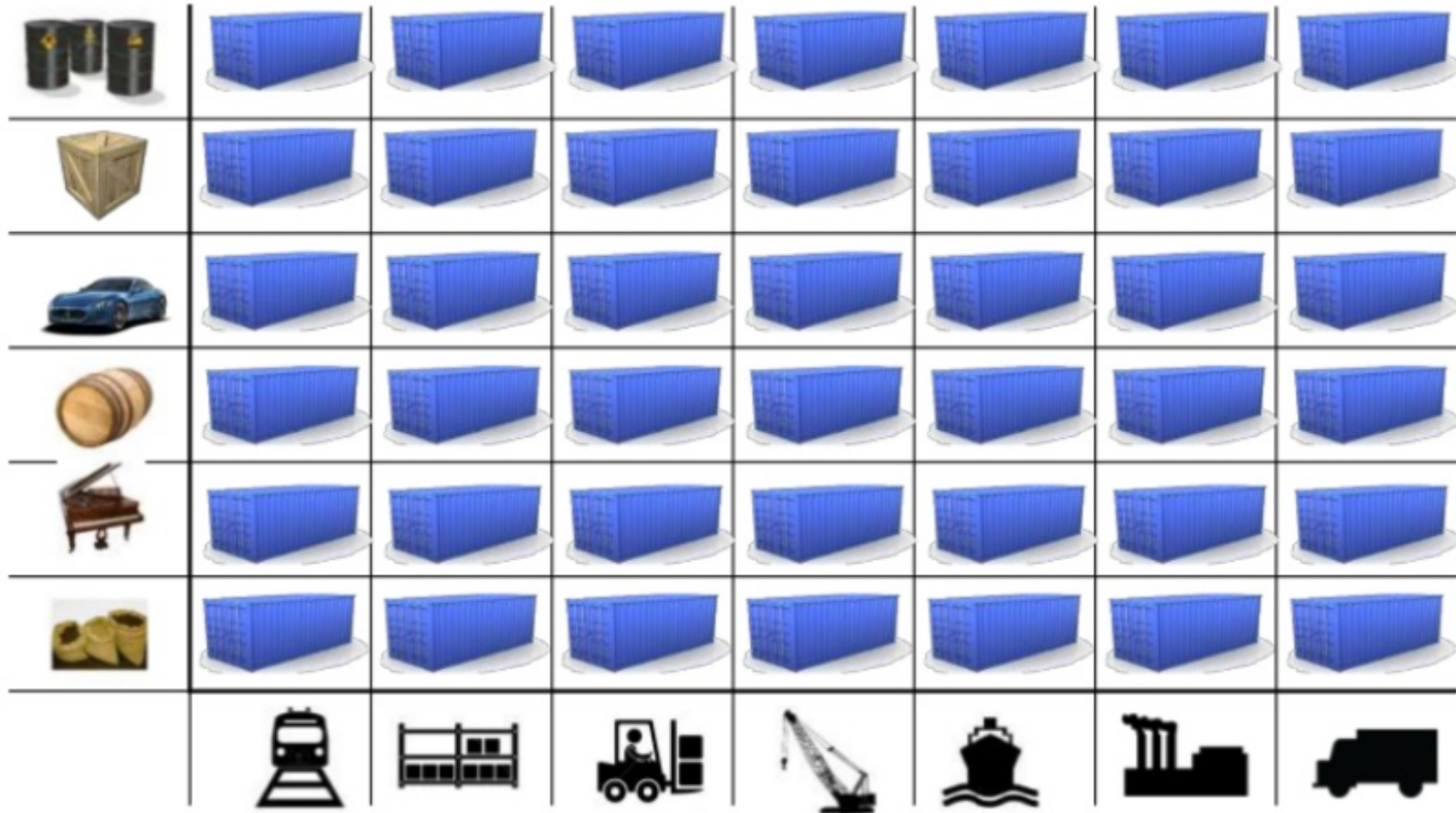
Before 1960, cargo transport looked like:



MxN combinatorics: matrix from Hell

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

Solution: Intermodal shipping container
















Containers - analysis

- enables seamless shipping on roads, railways and sea (intermodal)
- standardized dimensions
- opaque box convenient for all types of goods (privacy)



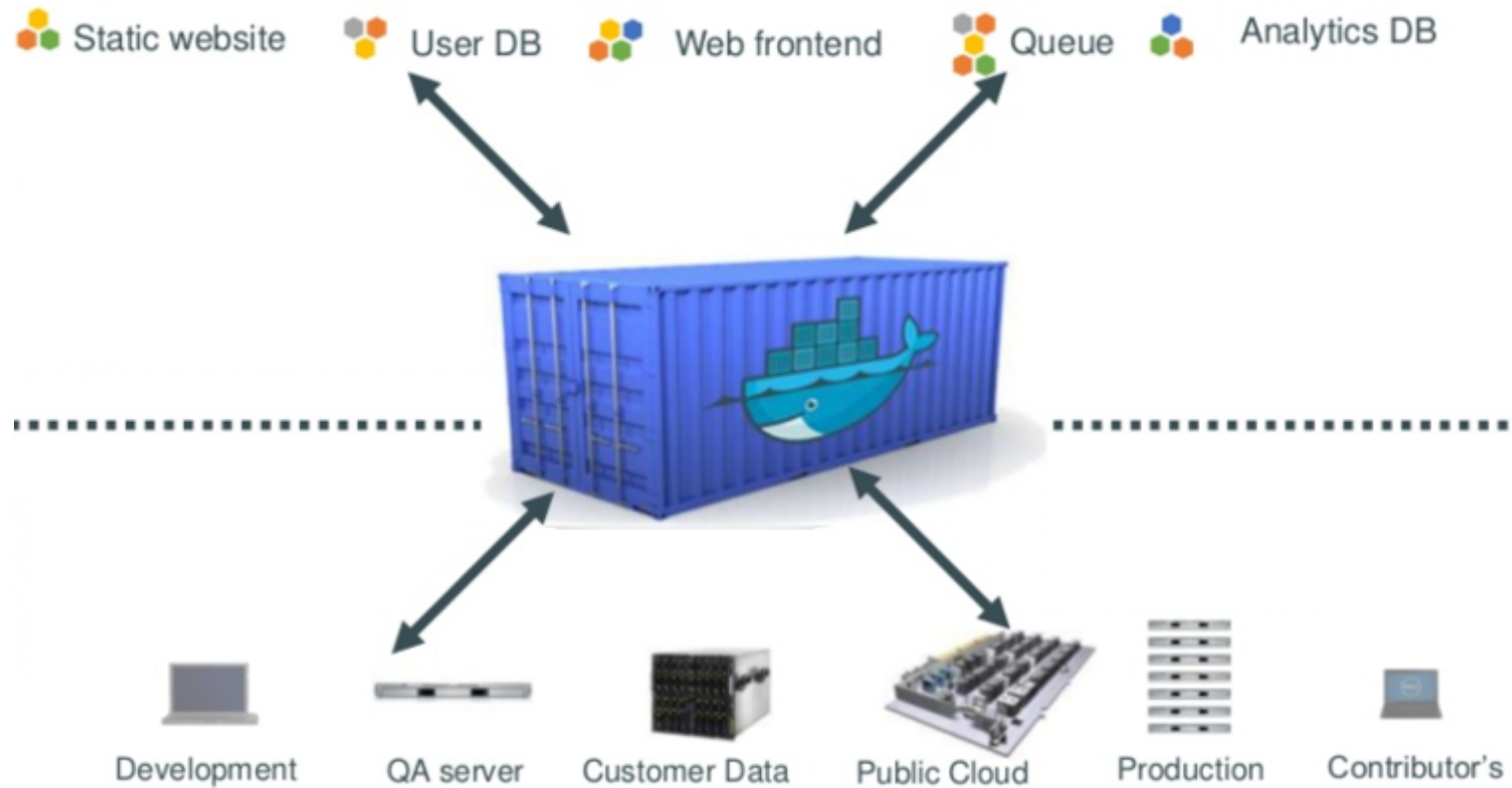
What is Docker?

Application deployment

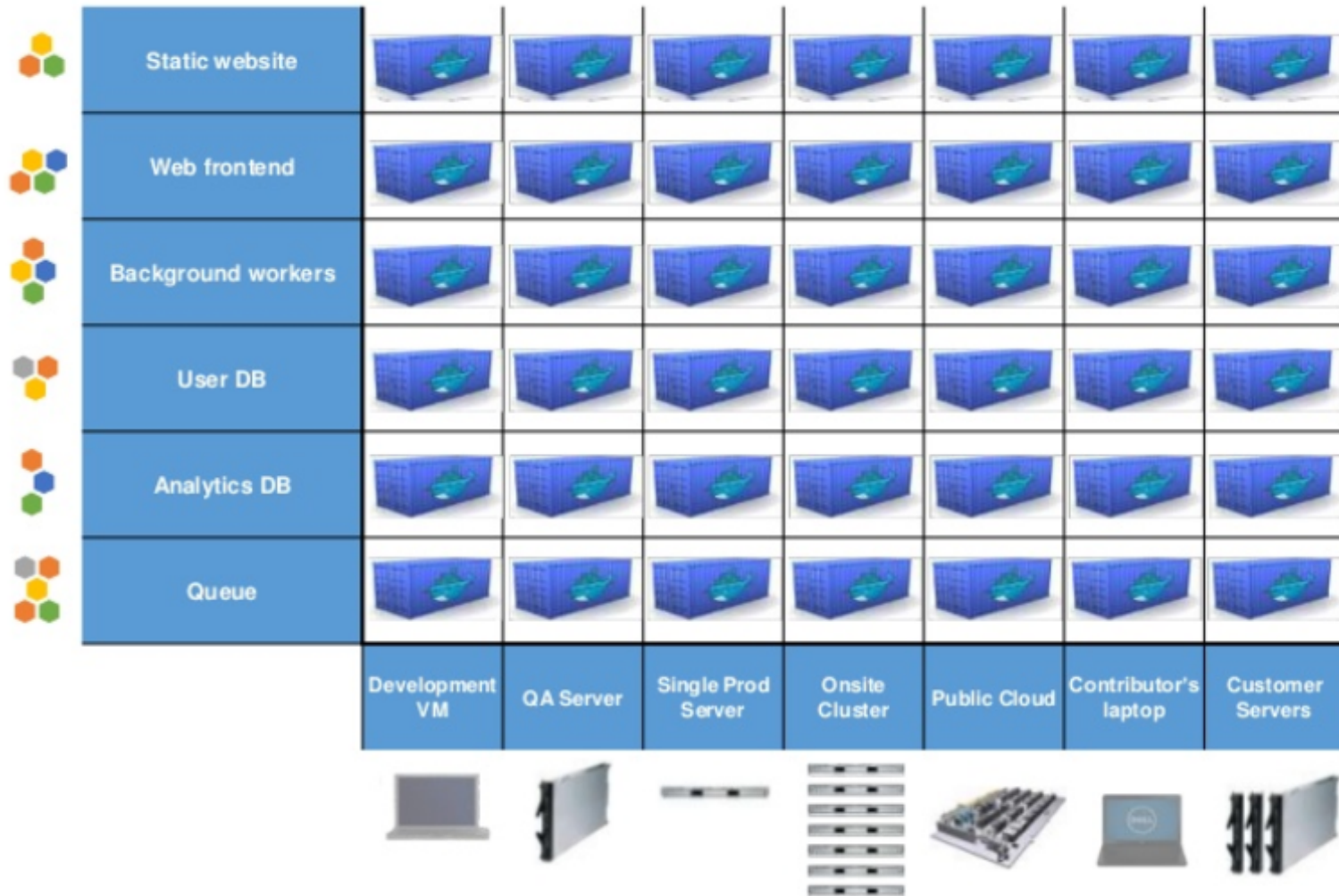
	Static website	?	?	?	?	?	?	?
	Web frontend	?	?	?	?	?	?	?
	Background workers	?	?	?	?	?	?	?
	User DB	?	?	?	?	?	?	?
	Analytics DB	?	?	?	?	?	?	?
	Queue	?	?	?	?	?	?	?
		Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
								

Note: a 3rd dimension (OS/platform) could be considered

Docker: an application container



Docker: no combinatorics no more



Docker

Docker is an open source project to pack ship and run any application as a lightweight container: [docker.io](http://www.docker.io) (<http://www.docker.io>)

Note: Although docker is primarily (ATM) Linux-oriented, it supports other OSes (Windows+MacOSX) at the price of a thin Linux VM which is automatically installed (and managed) on these systems.

See [docker installation](https://docs.docker.com/installation/) (<https://docs.docker.com/installation/>)

Docker

Docker is an open source project to pack ship and run any application as a lightweight container: [docker.io](http://www.docker.io) (<http://www.docker.io>)

High-level description:

- kind of like a lightweight VM
- runs in its own process space
- has its own network interface
- can run stuff as root

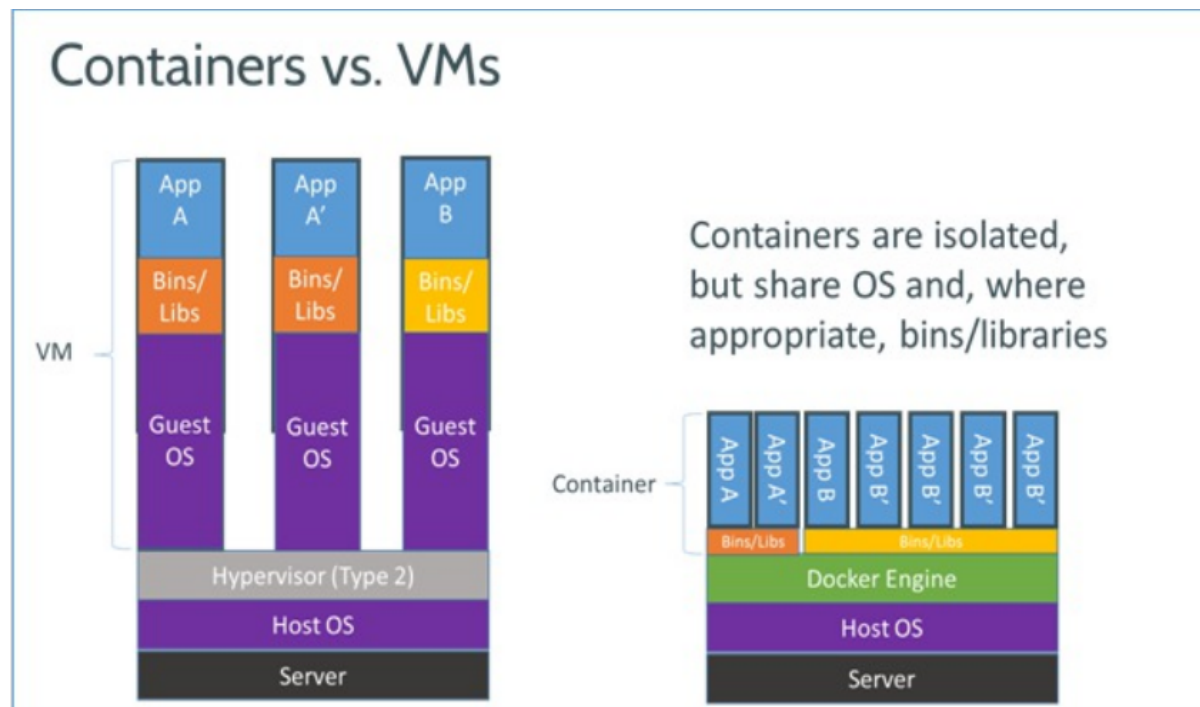
Low-level description:

- chroot on steroids
- container == isolated process(es)
- share kernel with host

- no device emulation

Docker: why?

- same use cases than for VMs (for Linux centric workloads)
- **speed:** boots in (milli)seconds
- **footprint:** 100-1000 containers on a single machine/laptop, small disk requirements



Docker: why?

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

Docker: why?

Efficiency: storage friendly

- unioning filesystems
- snapshotting filesystems
- copy-on-write

Docker: why?

- provisioning takes a few milliseconds
- ... and a few kilobytes
- creating a new container/base-image takes a few seconds

Why are Docker containers lightweight?

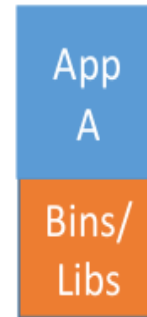
VMs



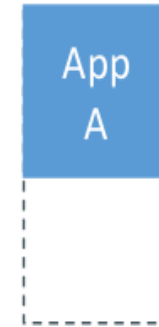
VMs

Every app, every copy of an app, and every slight modification of the app requires a new virtual server

Containers



Original App
(No OS to take up space, resources, or require restart)



Copy of App
No OS. Can Share bins/libs



Modified App
Copy on write capabilities allow us to only save the Between container and container A'

Separation of concerns

Tailored for the dev team:

- my code
- my framework
- my libraries
- my system dependencies
- my packaging system
- my distro
- my data

Don't care where it's running or how.

Separation of concerns

Tailored for the ops team:

- logs
- backups
- remote access
- monitoring
- uptime

Don't care what's running in it.

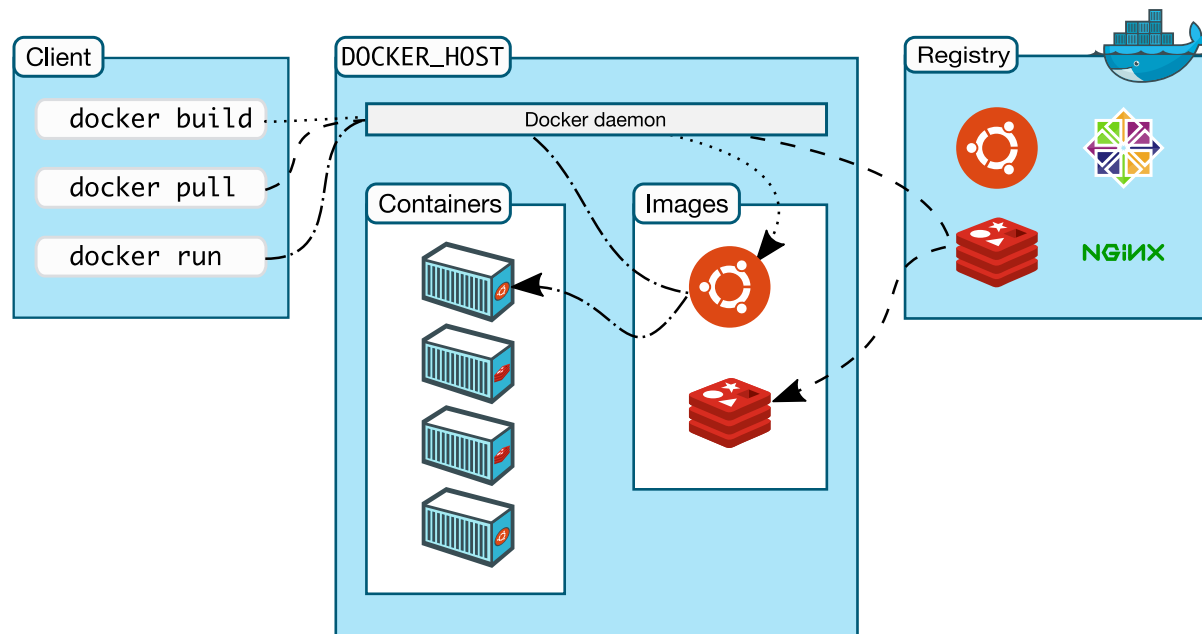
Docker: blueprint

Docker: blueprint

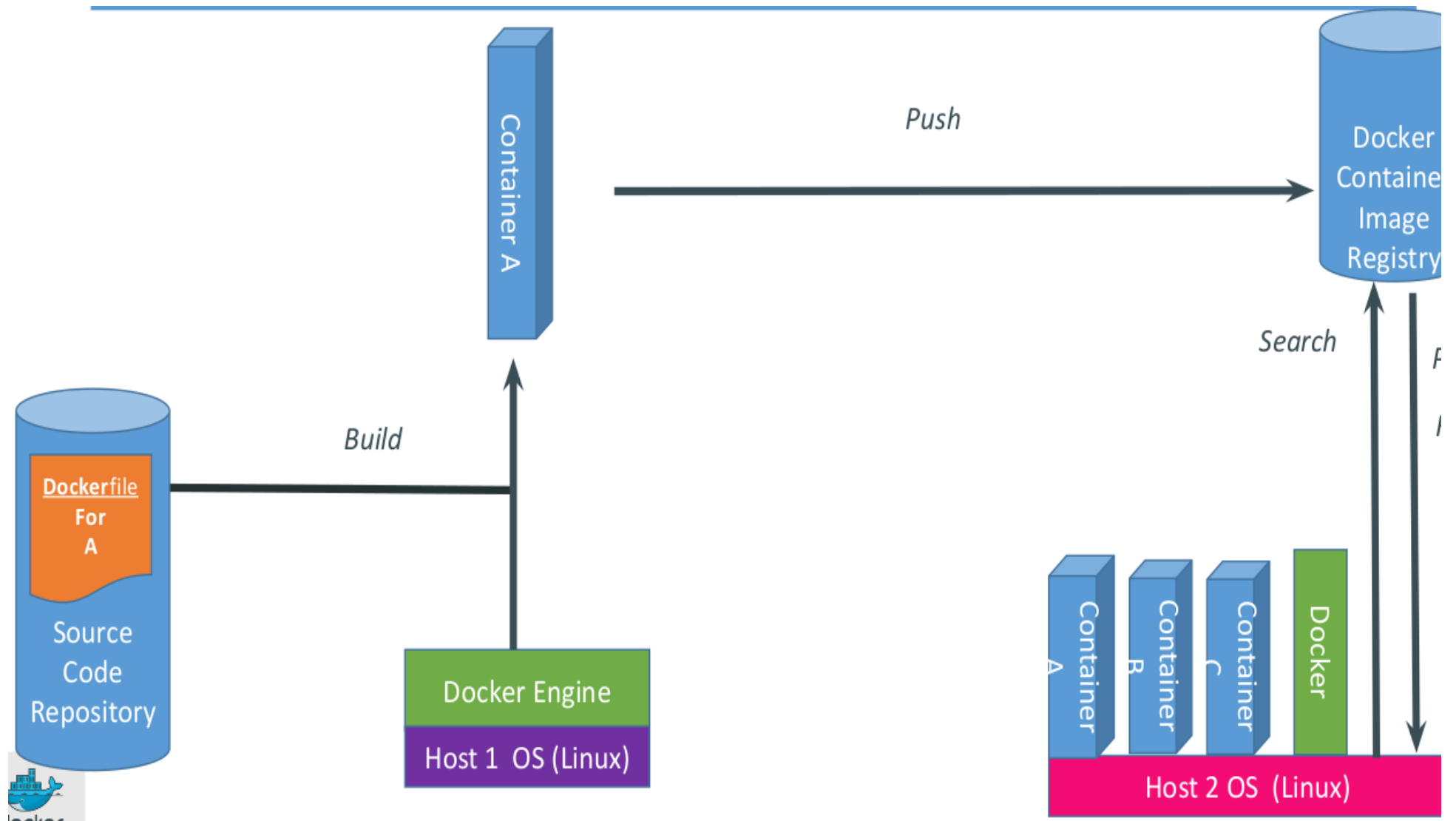
Build, ship and run any application, anywhere.

Docker uses a client/server architecture:

- the docker *client* talks to
- a docker *daemon* via sockets or a RESTful API.



Docker: basics of the system



Docker: the CLI

The docker client ships with many a subcommand:

```
$ docker help
Usage: docker [OPTIONS] COMMAND [arg...]
       docker daemon [ --help | ... ]
       docker [ -h | --help | -v | --version ]
```

A self-sufficient runtime for containers.

[...]

Commands:

attach	Attach to a running container
build	Build an image from a Dockerfile
commit	Create a new image from a container's changes
cp	Copy files/folders from a container to a HOSTDIR or to STDOUT
images	List images
import	Import the contents from a tarball to create a filesystem image
info	Display system-wide information

[...]

Docker: the CLI

```
$ docker version
```

```
Client:
```

```
Version:      1.8.1  
API version:  1.20  
Go version:   go1.4.2  
Git commit:   d12ea79  
Built:        Sat Aug 15 17:29:10 UTC 2015  
OS/Arch:      linux/amd64
```

```
Server:
```

```
Version:      1.8.1  
API version:  1.20  
Go version:   go1.4.2  
Git commit:   d12ea79  
Built:        Sat Aug 15 17:29:10 UTC 2015  
OS/Arch:      linux/amd64
```

Interlude: docker configuration @CC.in2p3

Linux

Just use the default config.

MacOSX/Windows

Same thing, but on MacOSX and Windows, let's create a new Linux VM and call it vm-ecole:

```
$ docker-machine create -d virtualbox \
  vm-ecole

$ eval $(docker-machine env vm-ecole) ## MacOSX
$ eval $(./docker-machine.exe env --shell=bash vm-ecole) ## Win
```

Hello World

Fetch a docker image from the docker registry:

```
$ docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
cf2616975b4a: Pull complete
6ce2e90b0bc7: Pull complete
8c2e06607696: Already exists
library/busybox:latest: The image you are pulling has been verified. Important: image verification
Digest: sha256:38a203e1986cf79639cfb9b2e1d6e773de84002feea2d4eb006b52004ee8502d
Status: Downloaded newer image for busybox:latest
```



```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
busybox	latest	8c2e06607696	4 months ago	2.43 MB

Now, run a command inside the image:

```
$ docker run busybox echo "Hello World"
Hello World
```

Docker basics

- Run a container in detached mode:

```
$ docker run -d busybox sh -c \  
'while true; do echo "hello"; sleep 1; done;'
```

- Retrieve the container id:

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
321c1aa5bcd4	busybox	"sh -c 'while true; d"	3 seconds ago	Up 2 seconds

- Attach to the running container:

```
$ docker attach 321c1aa5bcd4  
hello  
hello  
[...]
```

- Start/stop/restart container

```
$ docker stop 321c1aa5bcd4  
$ docker restart 321c1aa5bcd4
```


Docker: public index (aka registry, aka the Hub)

Docker containers may be published and shared on a public registry, the Hub.

- It is searchable:

```
$ docker search apache2
```

NAME	STARS	OFFICIAL	AUTOMATED
rootlogin/apache2-symfony2	7		[OK]
reinblau/php-apache2	6		[OK]
tianon/apache2	4		[OK]
[...]			

```
$ docker pull tianon/apache2
```

- Run the image and check the ports

```
$ docker run -d -p 8080:80 tianon/apache2
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	PORTS
49614161f5b7	tianon/apache2	"apache2 -DFOREGROUND"	0.0.0.0:8080->80/tcp

The registry is also available from the browser:

- hub.docker.com (<https://hub.docker.com>)

Docker: creating a customized image

- run docker interactively:

```
$ docker run -it ubuntu bash
root@524ef6c2e4ce:/# apt-get install -y memcached
[...]
root@524ef6c2e4ce:/# exit

$ docker commit `docker ps -q -l` binet/memcached
4242210aba21641013b22198c7bdc00435b00850aaf9ae9cedc53ba75794891d

$ docker run -d -p 11211 -u daemon binet/memcached memcached
a84e18168f1473a338f9ea3473dd981bf5e3dc7e41511a1252f7bb216d875860

$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	PORTS
a84e18168f14	binet/memcached	"memcached"	0.0.0:32768->11211/tcp

Docker: creating a customized image

- interactive way is fine but not scalable
- enter Dockerfiles
- recipes to build an image
- start FROM a base image
- RUN commands on top of it
- easy to learn, easy to use

Docker: Dockerfile

```
FROM ubuntu:14.04

RUN apt-get update
RUN apt-get install -y nginx
ENV MSG="Hi, I am in your container!"
RUN echo "$MSG" > /usr/share/nginx/html/index.html

CMD nginx -g "daemon off;"

EXPOSE 80
```

Docker: Dockerfile-II

- run in the directory holding that Dockerfile

```
$ docker build -t <myname>/server .  
$ docker run -d -P <myname>/server
```

- retrieve the port number:

```
$ docker ps  
34dc03cdbae8          binet/server          "/bin/sh -c 'nginx -g"  0.0.0.0:32770->80/tcp
```

or:

```
$ docker inspect -f '{{.NetworkSettings.Ports}}' 34dc03cdbae8
```

and then:

```
$ curl localhost:32770  
Hi, I am in your container!
```

Docker: Dockerfile-III

NOTE: for Windows(TM) and MacOSX(TM) users, a thin Linux VM is sitting between your machine and the container.

The container is running inside that VM so you need to replace localhost with the IP of that VM:

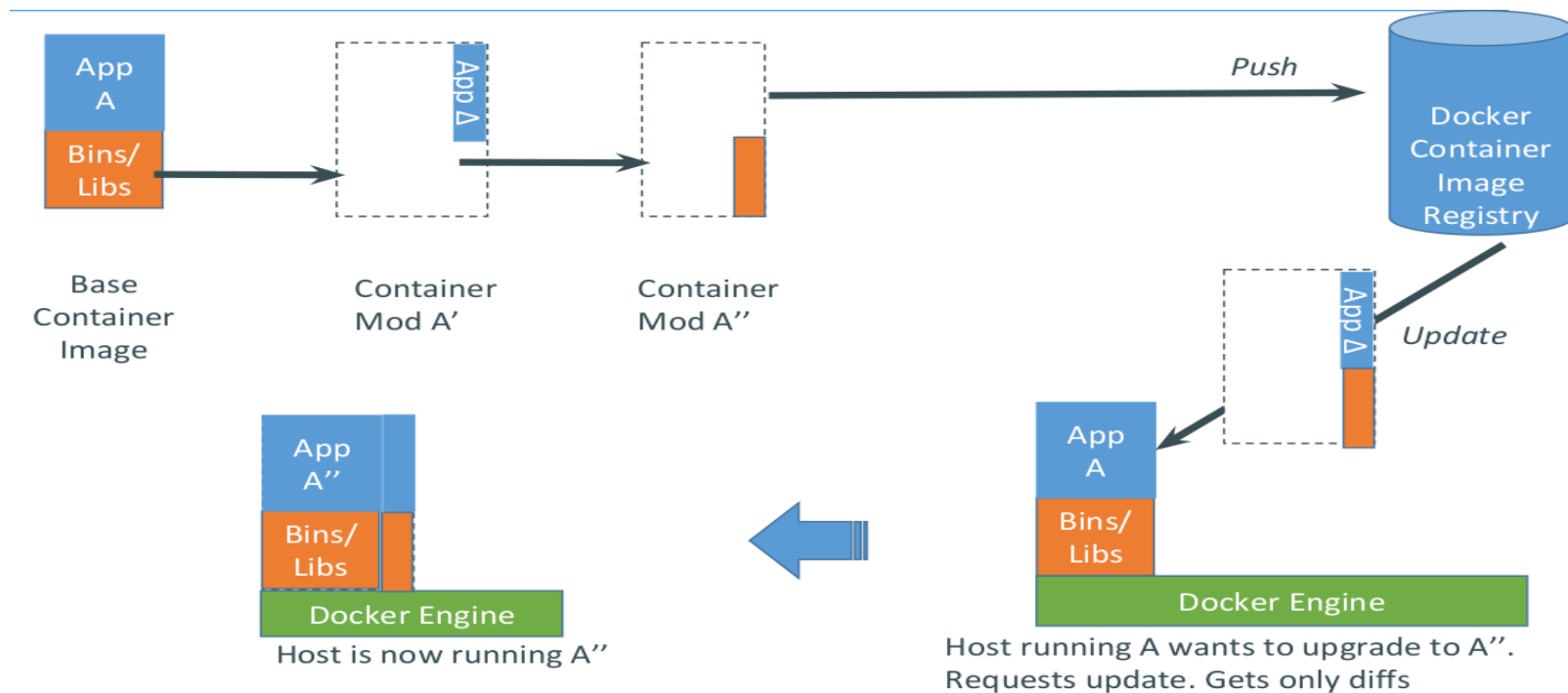
```
$ docker-machine ip vm-ecole  
192.168.59.103
```

and then:

```
$ curl 192.168.59.103:32770  
Hi, I am in your container!
```

docker build

- takes a snapshot after each step
- re-uses those snapshots in future builds
- doesn't re-run slow steps when it isn't necessary (cache system)



Docker Hub

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

This brings:

- reliable deployment
- consistency
- images are self-contained, independent from host
- if it works locally, it will work on the server
- *exact same behavior*
- regardless of versions, distros and dependencies

Docker for the developer

- manage and **control** dependencies
- if it works on my machine, it works on the cluster
- reproducibility
- small but durable recipes

Never again:

- juggle with 3 different incompatible FORTRAN compilers
- voodoo incantations to get that exotic library to link with IDL
- figure out which version of LAPACK works with that code
- ... and what obscure flag coaxed it into compiling last time

Development workflow

- Fetch code (git, mercurial, ...)

```
$ git clone git@gitlab.in2p3.fr:EcoleInfo2015/TP.git  
$ cd TP
```

- Edit code
- Mount code inside a build container
- Build+test inside that container

We'll test this workflow in the remainder of the hands-on session...

Create a base container

- create a directory `docker-web-base` to hold the `Dockerfile` for the base container
- create the `Dockerfile` and choose your favorite Linux distro (say, centos) as a base image,
- install the needed dependencies for the web-app (maven on centos)
- run:

```
$ docker build -t <myname>/web-base .
```

Create a base container - solution

(see next slide)

Create a base container - II

```
## base environment w/ deps for the web-app
FROM centos:7
MAINTAINER binet@cern.ch

RUN yum update -y && \
    yum install -y maven

RUN mkdir -p /opt/in2p3/tp

## change current directory
WORKDIR /opt/in2p3/tp

## prepare for web server
EXPOSE 8080
```

Base container for development

- One could create a new container with all the development tools (editor, completion, ...)
- But you'd need to carry over the configuration (ssh keys, editor, ...)

Probably easier to just mount the sources **inside** the base container:

```
$ docker run -it -v `pwd`: /opt/in2p3/tp -p 8080:8080 <myname>/web-base bash
[root@48b2c74a5004 tp]# ./bin/compile.sh
[root@48b2c74a5004 tp]# java -jar /opt/in2p3/tp/target/webserver-1.0-SNAPSHOT-jar-with-dependencies.jar
2015-09-10 14:13:25.467:INFO::main: Logging initialized @623ms
2015-09-10 14:13:25.719:INFO:oejs.Server:main: jetty-9.3.z-SNAPSHOT
2015-09-10 14:13:25.918:INFO:oejsh.ContextHandler:main: Started o.e.j.s.ServletContextHandler@40...
2015-09-10 14:13:25.933:INFO:oejs.ServerConnector:main: Started ServerConnector@62b60fbb{HTTP/1.1}
2015-09-10 14:13:25.934:INFO:oejs.Server:main: Started @1093ms
```

In another terminal:

```
$ curl localhost:8080
<h1>Bienvenue à l'école informatique IN2P3 2015</h1>
<a href='/analyse'>Analyse de données</a>
session=1kdgwh0cue0efokpe5t1gcvtj
```


Base container for dev - II

- On windows, the correct -v syntax is like:

```
$ docker run -it -v //c/Users/username/some/path:/opt/in2p3/tp ...
```

github.com/docker/docker/issues/12590#issuecomment-96767796

(<https://github.com/docker/docker/issues/12590#issuecomment-96767796>)

Create the final container

Now that we know the base image "works", we'll automatize the build part as yet another Dockerfile:

- create a new Dockerfile file (at the root of the git repository) based on the web-base image, with the correct build+run instructions
- make sure you can docker build it and tag it as web-app
- make sure that you can still access the web server when you run:

```
$ docker run -d -p 8080:8080 <myname>/web-app
```

Hint: ADD

Hint: CMD

docs.docker.com/reference/builder/ (<https://docs.docker.com/reference/builder/>)

Create the final container - solutions

(see next slide)

Create the final container - II

```
## image for the web-app
FROM binet/web-base

MAINTAINER binet@cern.ch

## add the whole git-repo
ADD . /opt/in2p3/tp

WORKDIR /opt/in2p3/tp
RUN ./bin/compile.sh

CMD java -jar /opt/in2p3/tp/target/webserver-1.0-SNAPSHOT-jar-with-dependencies.jar
```

Create the final container - III

- CMD describes the command to be run by default when the container is started
- ADD copies files, directories or URLs into the container's filesystem
- VOLUME creates a volume mount point inside the container which can contain data from the host or from other containers
- USER defines the user (or UID) with whom to run the various commands inside the container

Create multiple versions of an image

At times, it might be very useful to test 2 versions of an application and run them concurrently (to debug discrepancies.)

Let's do just that.

- tag the last web-app image as v1

```
$ docker tag \  
    <myname>/web-app \  
    <myname>/web-app:v1
```

- modify
src/main/java/fr/in2p3/informatique/ecole2015/web/MyServlet.java to
print a different welcome message
- containerize the new version as .../web-app:v2

Create multiple versions of an image - II

- run the container v2 on port 8082

```
$ docker run -p 8082:8080 \  
  --name=web-app-v2      \  
  <myname>/web-app:v2
```

- run the container v1 on port 8080

```
$ docker run -p 8080:8080 \  
  --name=web-app-v1      \  
  <myname>/web-app:v1
```

- make sure the servers on ports 8080 and 8082 display the correct welcome messages.

Continuous Integration

Modify the Jenkins build to:

- build the final container
- run the tests suite
- push the result as a tagged image

Sharing images

Up to now, the images you've been creating have been put on your local registry. But there is another registry instance available at:

```
cc-ecole2015-docker.in2p3.fr:5000
```

Let's try to package the previous `web-app:v2` and `web-app:v1` images and put them on that new registry:

```
$ docker tag \  
  <myname>/web-app \  
  cc-ecole2015-docker.in2p3.fr:5000/<myname>/web-app
```

Now, try to `pull` the `web-app` image of your friend and run it.

Inspecting logs

docker is nice enough to let us inspect what (running) containers are generating as logs.

For a single container, it is as simple as:

```
$ docker logs <some-container-id>  
$ docker logs <some-container-name>
```

- inspect the logs of your web-app-v2 container
- inspect the logs of the container running your local registry

Inspecting logs - II

e.g.:

```
$ docker logs web-app-v2
{
  "timeMillis" : 1443615891663,
  "thread" : "main",
  "level" : "INFO",
  "loggerName" : "org.eclipse.jetty.server.Server",
  "message" : "Started @1343ms",
  "endOfBatch" : false,
  "loggerFqcn" : "org.eclipse.jetty.util.log.Slf4jLog",
  "contextMap" : [ ]
}

$ docker logs ecole-registry
[...]
30/Sep/2015:12:20:13 +0000 DEBUG: args = {'tag': u'latest', 'namespace': u'binet', 'repository'
30/Sep/2015:12:20:13 +0000 DEBUG: [get_tag] namespace=binet; repository=web-app; tag=latest
30/Sep/2015:12:20:13 +0000 DEBUG: api_error: Tag not found
172.17.42.1 - - [30/Sep/2015:12:20:13 +0000] "GET /v1/repositories/binet/web-app/tags/latest HT
```

Inspecting logs - III

- launch a container in interactive mode
- start a bash shell
- run inside that container:

```
docker> logger -i -s plop
```

- in another terminal:

```
$ docker logs <container-id>
```

Creation of a build+target container pair

So far, we have been building containers where the intermediate results leading to the final binary (or set of binaries) are left inside the image.

This might not be completely efficient if these intermediate steps are (disk) resource heavy.

The usual solution is to have a 2-step process:

- a container in which the binaries are built
- a container in which the binaries are directly copied from the first

Let's do that.

Hint: docker export

Hint: docker import

Hint: docker cp

Creation of a build+target container pair

(solution on next slide)

Creation of a build+target container pair

Extract the root fs from the build image:

```
$ mkdir rootfs && cd rootfs
$ docker run -d -p 8080:8080 --name=web-app-v2 \
  localhost:5000/<myname>/web-app:v2
$ docker export web-app-v2 | tar xf -
$ ls opt/in2p3/tp
bin/                .git/               sonar-project.properties
doc/                .gitignore          src/
Dockerfile          pom.xml             target/
docker-web-base/    README.md
```

Another way is to use docker cp:

```
$ docker cp web-app-v2:/opt/in2p3/tp tp
```

The binaries are under target.

If they were static libraries, you could just create a very slim container with them, using docker import t.

Running GUIs

The application we have been currently "dockerizing" doesn't need any graphics per se.

Many do, though.

Let's try to run a simple graphics-enabled application from within a docker container:

```
$ docker run -it --rm centos bash
docker> yum install -y xclock
docker> xclock
```

If the network is too slow, you can try to pull:

```
$ docker pull \
    cc-ecole2015-docker.in2p3.fr:5000/centos-xclock
```

where `xclock` and its dependencies have been already installed.

Running GUIs - II

Running GUIs is a bit more involved than just running your simple "from the mill" CLI application.

There are many options to enable graphics:

- ssh into a container with X11 forwarding
- VNC
- sharing the X11 socket

fabiorehm.com/blog/2014/09/11/running-gui-apps-with-docker/

(<http://fabiorehm.com/blog/2014/09/11/running-gui-apps-with-docker/>)

blog.docker.com/2013/07/docker-desktop-your-desktop-over-ssh-running-inside-of-a-docker-container/ (<https://blog.docker.com/2013/07/docker-desktop-your-desktop-over-ssh-running-inside-of-a-docker-container/>)

wiki.ros.org/docker/Tutorials/GUI (<http://wiki.ros.org/docker/Tutorials/GUI>)

Running GUIs - III

Let's try the most direct (albeit a bit insecure) one: sharing the X11 socket.

First, allow all X11 connections (that's the insecure part):

```
$ xhost +
```

Then:

```
$ docker run -ti --rm \  
    -e DISPLAY=$DISPLAY \  
    -v /tmp/.X11-unix:/tmp/.X11-unix \  
    centos bash  
docker> yum install -y xclock && xclock
```

Don't forget to re-enable X11 access control afterwards:

```
$ xhost -
```

Conclusions

- docker is a rather good tool to deploy applications in containers
- eases the life of developers and sysadmins (devops)
- docker isn't the only game in town
- [rkt](https://coreos.com/rkt/docs)(<https://coreos.com/rkt/docs>)(rocket) from CoreOS
- [systemd-nspawn](http://0pointer.de/public/systemd-man/systemd-nspawn.html) (<http://0pointer.de/public/systemd-man/systemd-nspawn.html>), now part of systemd

References

www.slideshare.net/jpetazzo/introduction-to-docker-december-2014-tour-de-france-bordeaux-special-edition (<http://www.slideshare.net/jpetazzo/introduction-to-docker-december-2014-tour-de-france-bordeaux-special-edition>)

www.slideshare.net/dotCloud/docker-intro-november (<http://www.slideshare.net/dotCloud/docker-intro-november>)

sif.info-ufr.univ-montp2.fr/docker-talk (<https://sif.info-ufr.univ-montp2.fr/docker-talk>)

docs.docker.com/introduction/understanding-docker/ (<https://docs.docker.com/introduction/understanding-docker/>)

wiki.jenkins-ci.org/display/JENKINS/Docker+Plugin (<https://wiki.jenkins-ci.org/display/JENKINS/Docker+Plugin>)

kubernetes.io/ (<http://kubernetes.io/>)

mesos.apache.org/ (<http://mesos.apache.org/>)

coreos.com/rkt/docs (<https://coreos.com/rkt/docs>)

Thank you

Sebastien Binet
CNRS/IN2P3

