Docker & rkt: Linux containerization and applications in astro/HEP

Sebastien Binet
CNRS/IN2P3/LPC

May 31, 2016

Docker origins

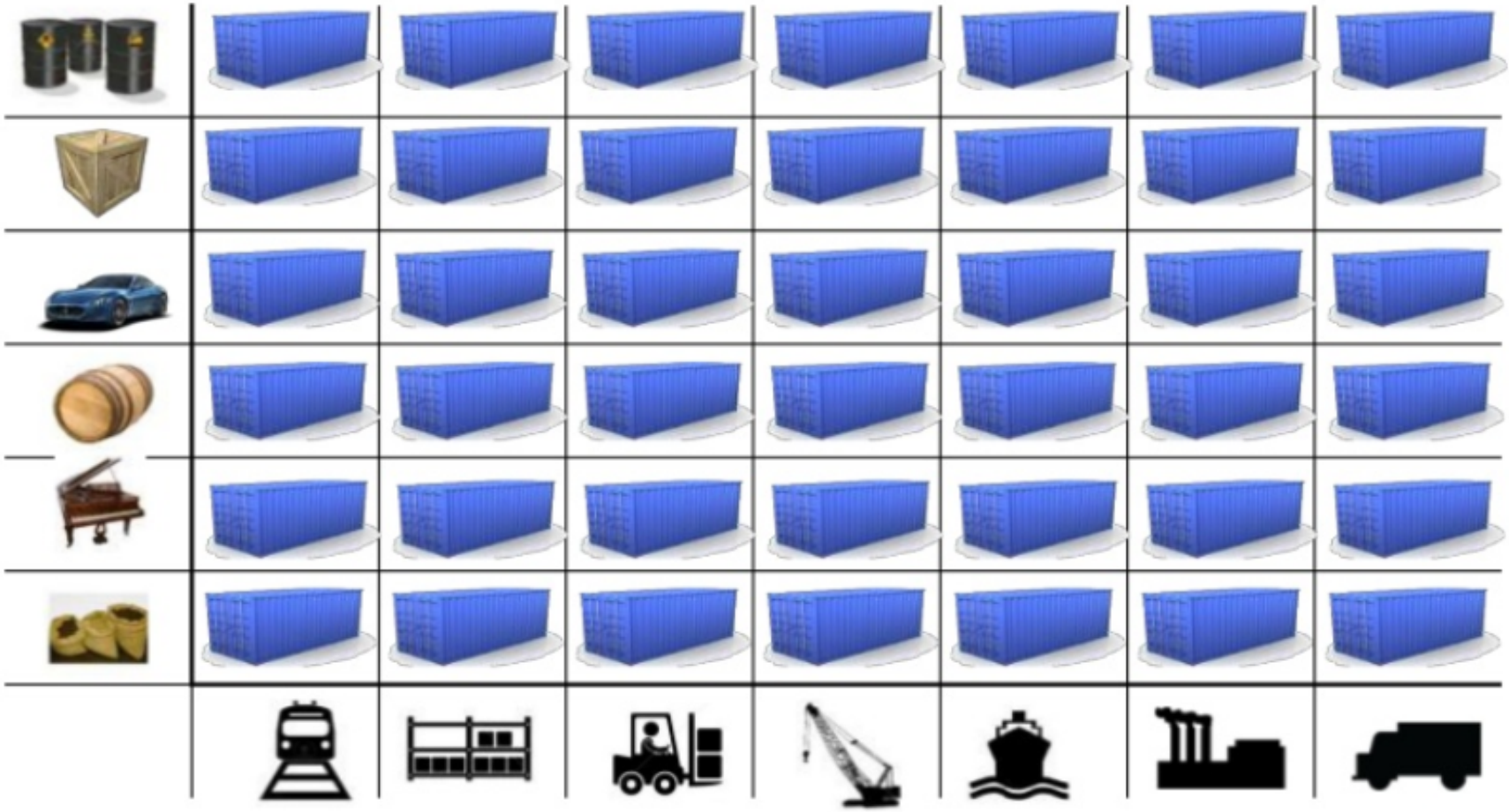
The container revolution

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)

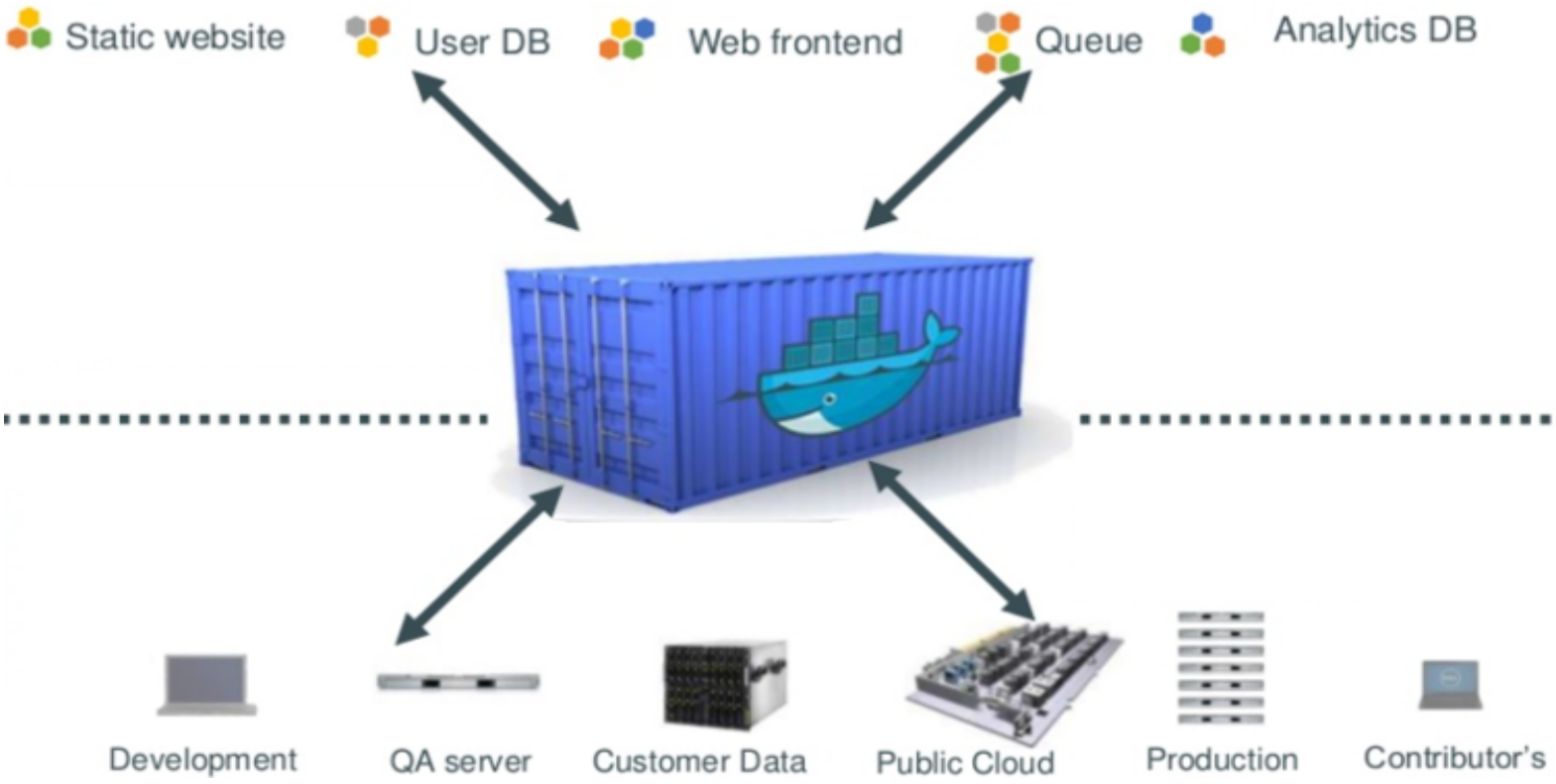
What is Docker?

Application deployment

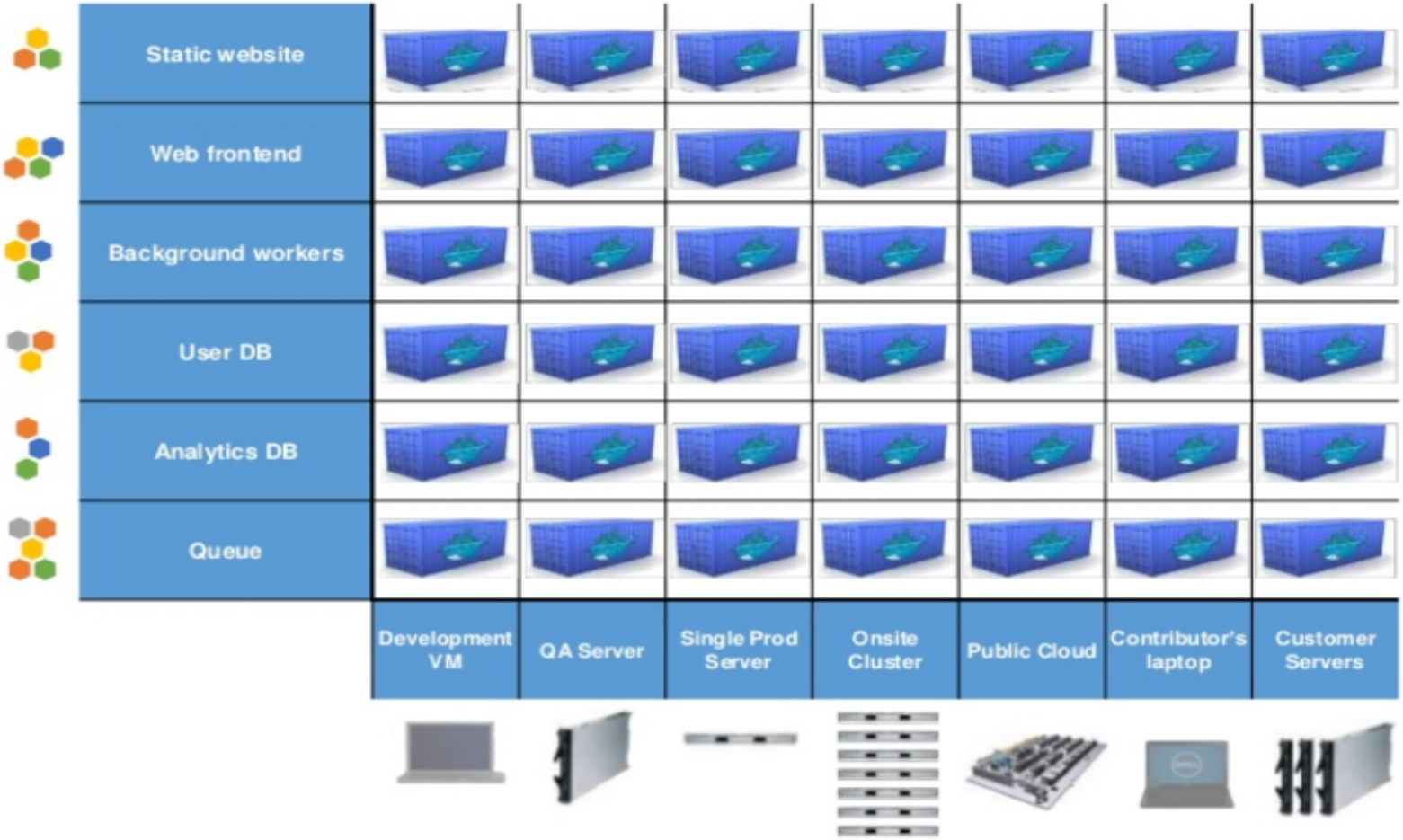
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

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`
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`

Docker

`Docker` is an open source project to pack ship and run any application as a lightweight container: `docker.io`
High-level description:

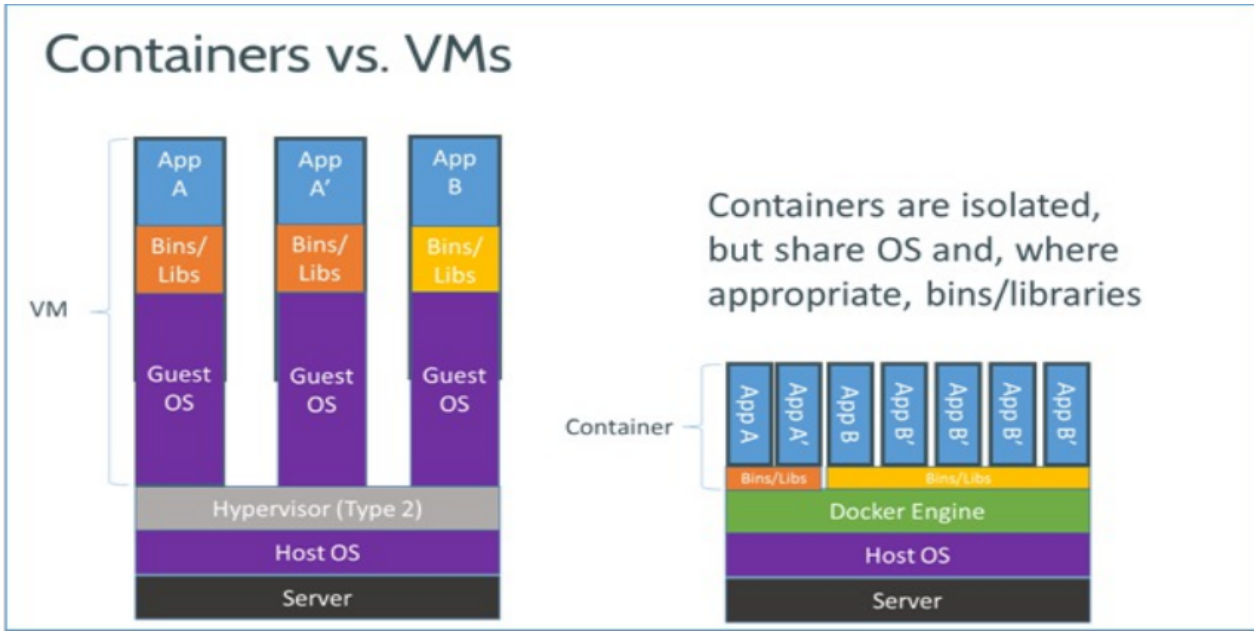
- kind of like a lightweight VM
- runs in its own process space
- has its own network interface
- can run stuff as `root`

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

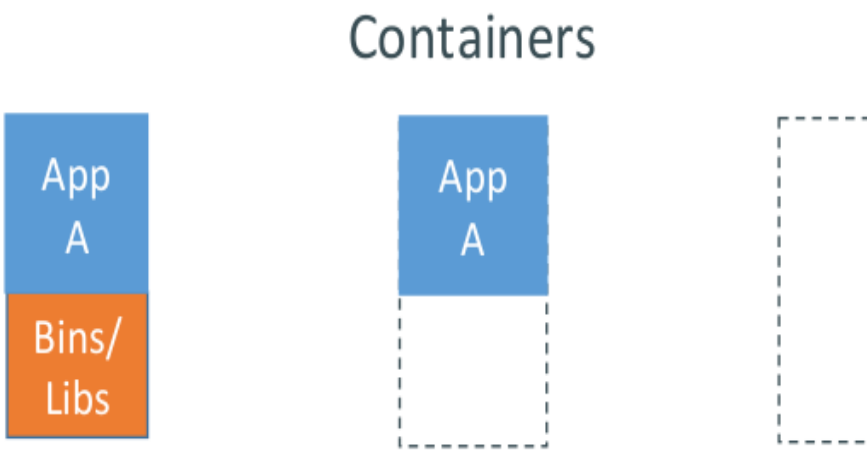
Docker: why?

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

Why are Docker containers lightweight?

VMs

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



Original App

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

Copy of App

No OS. Can Share bins/libs

Modifie

Copy on w capabilitie us to only Between c and contai 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

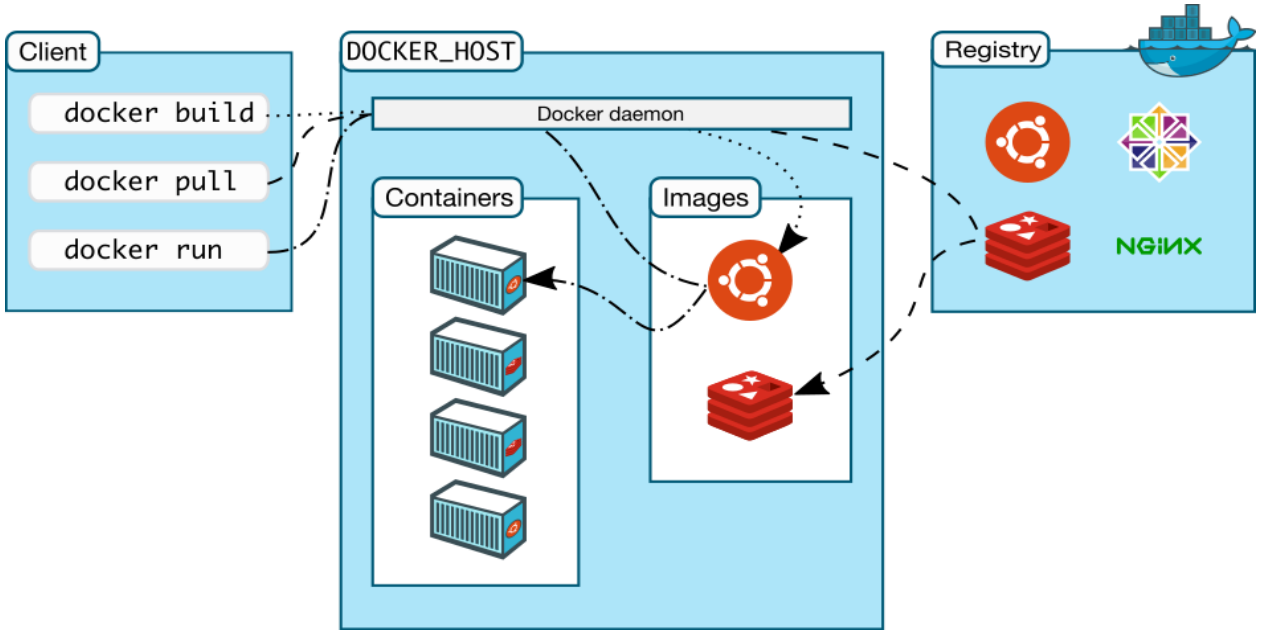
Don't care what's running in it.

Docker: blueprint

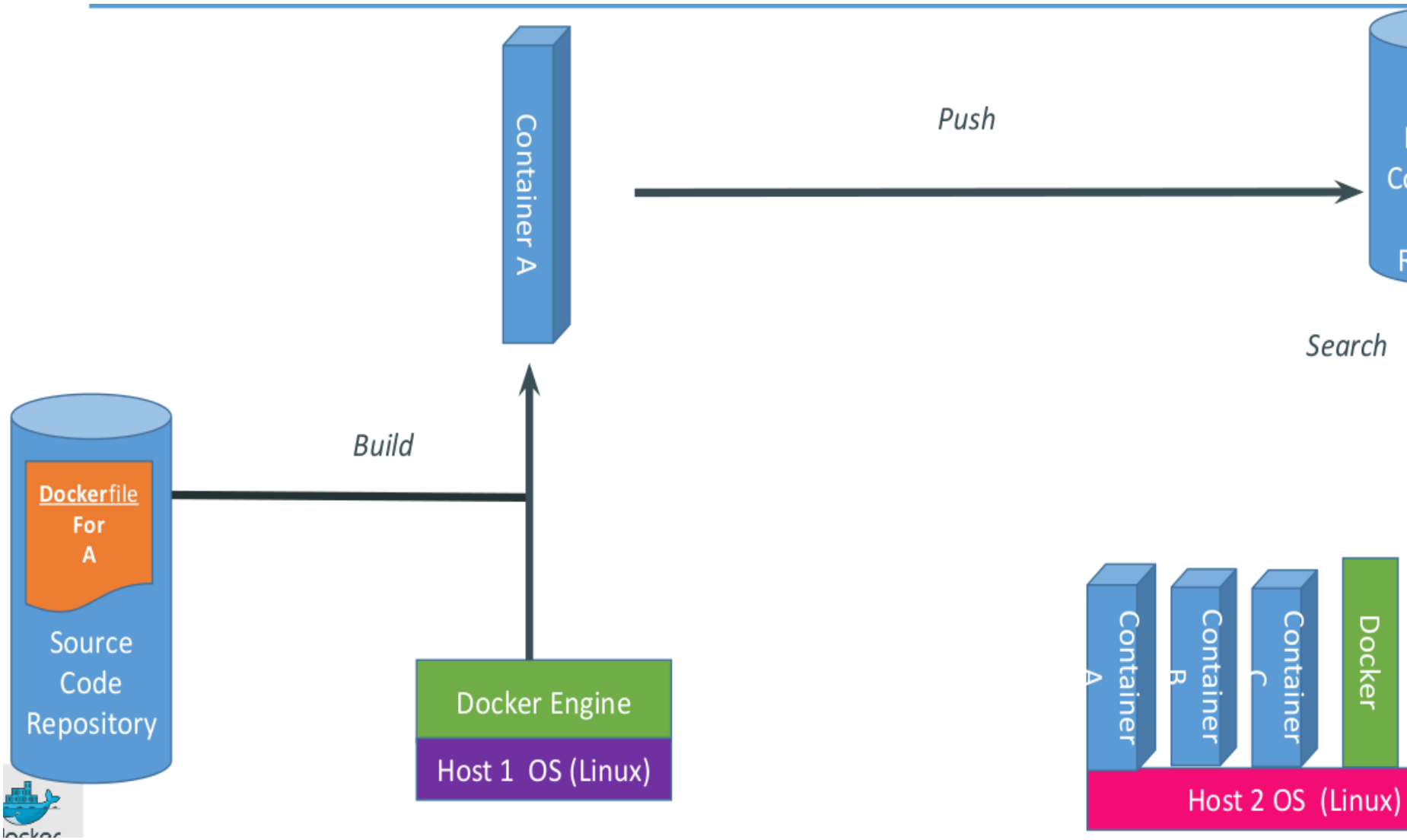
Docker: blueprint

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 ST
    images    List images
    import    Import the contents from a tarball to create a filesystem
    info      Display system-wide information
[...]
```

Docker: the CLI

```
$ docker version
Client:
 Version:      1.11.1
 API version:  1.23
 Go version:   go1.6.2
 Git commit:   5604cbe
 Built:        Mon May  2 00:06:51 2016
 OS/Arch:      linux/amd64

Server:
 Version:      1.11.1
 API version:  1.23
 Go version:   go1.6.2
 Git commit:   5604cbe
 Built:        Mon May  2 00:06:51 2016
 OS/Arch:      linux/amd64
```

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. Im
Digest: sha256:38a203e1986cf79639cfb9b2e1d6e773de84002feea2d4eb006b5200
Status: Downloaded newer image for busybox:latest

$ docker images
REPOSITORY                TAG                 IMAGE ID            CREATED
busybox                   latest              8c2e06607696        4 month
```

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                  CREATE
321c1aa5bcd4        busybox             "sh -c 'while true; d"   3 seco
```

- 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.
```

The registry is also available from the browser:

- 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"             0.0.0:3
```

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
```

or:

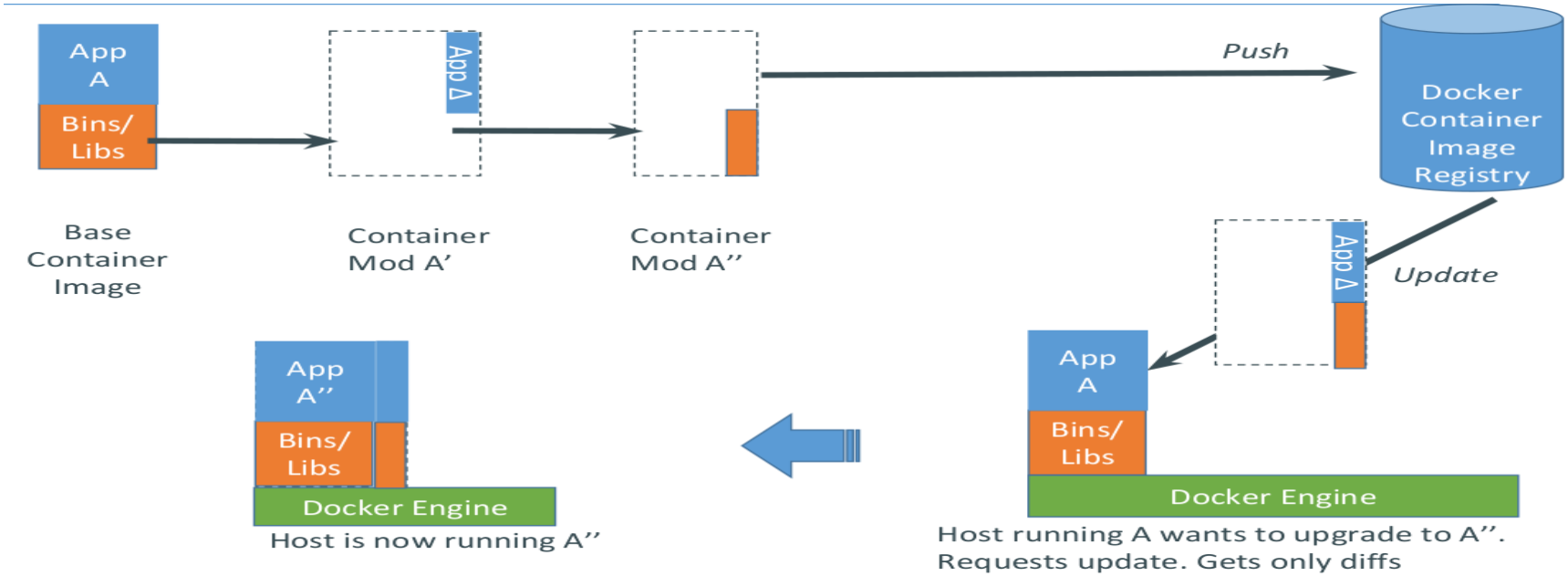
```
$ docker inspect -f '{{.NetworkSettings.Ports}}' 34dc03cdbae8
```

and then:

```
$ curl localhost: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@github.com:sbinet/my-project.git
$ vim my-project/some-file.cpp &
$ docker run -it \
   -v `pwd`/my-project:/src \
   -v `pwd`/build:/build \
  <my-name>/my-project-base-dev bash
```

- Edit code
- Mount code inside a `build` container with all dependencies pre-installed
- Build+test inside that container
- Retrieve the build artifacts under `/build`

Can be automatized via a `Makefile`:

- ideally, the `my-project-base-dev` image definition is provided by the `git` repository

rkt

rkt: introduction

rkt is another Go-based application to run containers.
The main differences *wrt* docker are:

- an improved process model
- an improved security support
- a somewhat more UNIX -y philosophy (one tool per job)

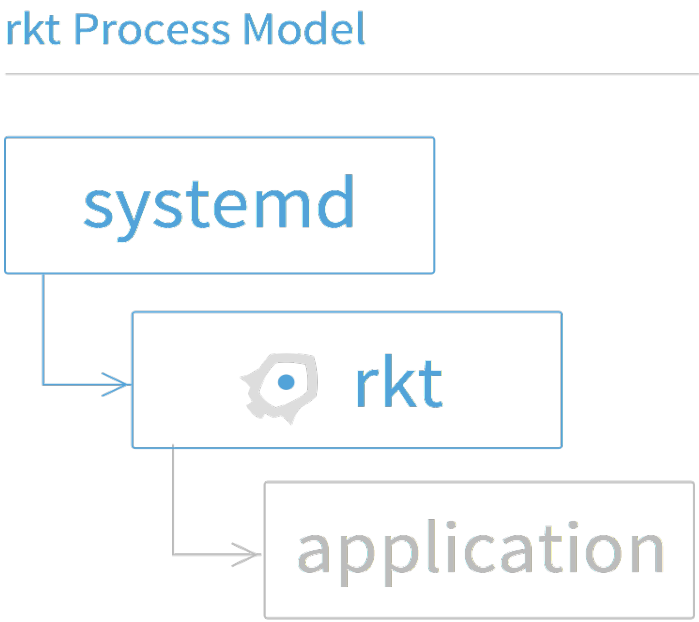
rkt implements the ACI (App Container Images) format to ensure portability and prevent **lock-in**.
rkt is also a partner of the OCI (Open Container Initiative) project.

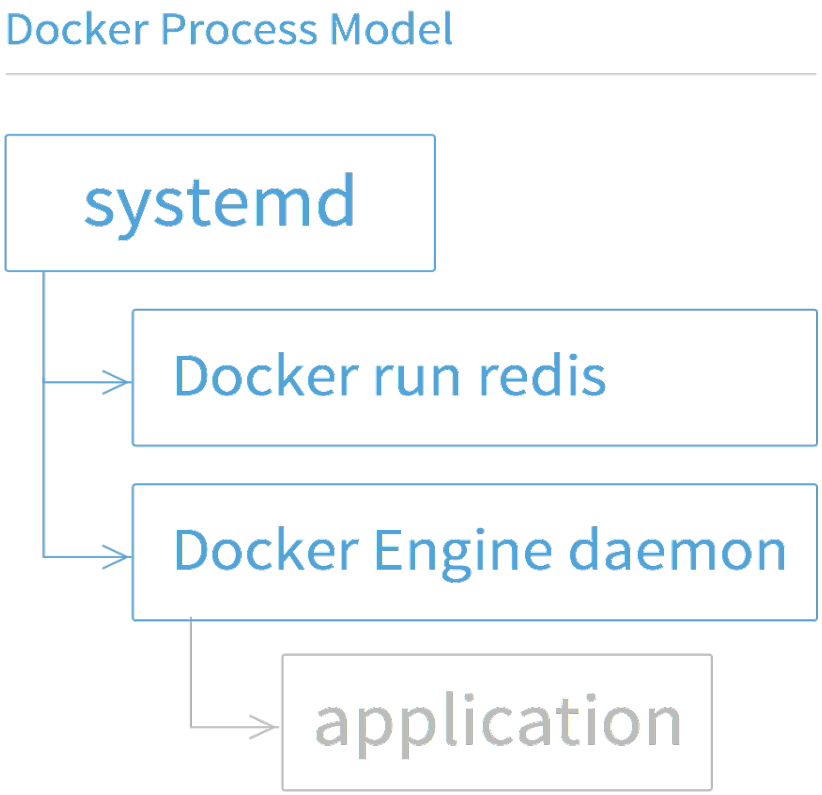
rkt: support for standards

ACI/appc and OCI try to standardize a few components of the container/image ecosystem:

- container image format (appc)
- image distribution (appc)
- runtime (appc, OCI)
- on-disk image format (OCI)

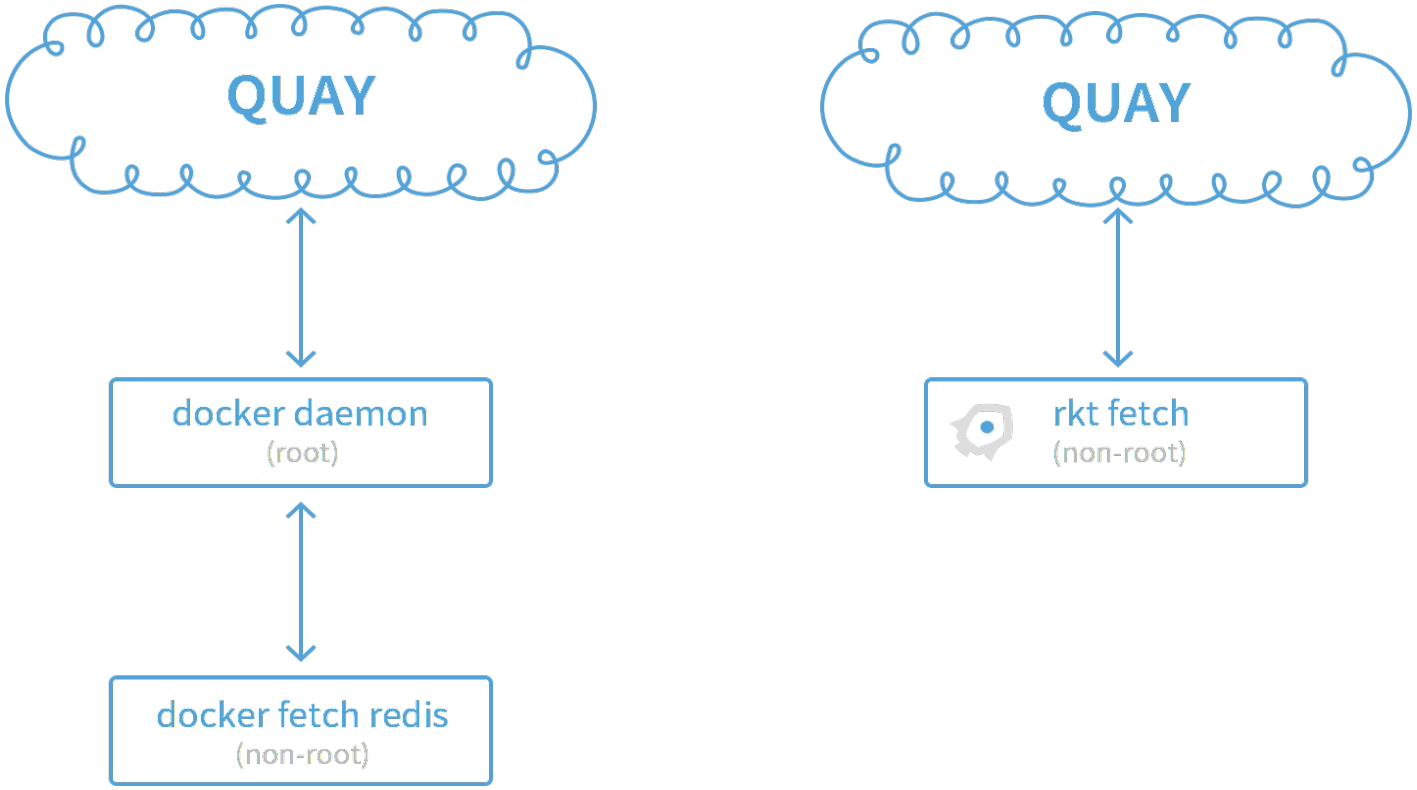
rkt: process model





- `rkt` has no centralized `"init"` daemon
- `rkt` launches containers directly from client commands
- `rkt` is thus compatible with `init` systems (`systemd`, `upstart`, ...)

rkt: privilege separation



- standard `UNIX` group permissions
- signature verification of downloaded images (with simple user privileges)

rkt: hello world

Create a statically linked Go web server:

```
package main

import (
        "log"
        "net/http"
)

func main() {
        http.HandleFunc("/", func(w http.ResponseWriter, r *http.Reques
                log.Printf("request from %v\n", r.RemoteAddr)
                w.Write([]byte("hello\n"))
        })
        log.Fatal(http.ListenAndServe(":5000", nil))
}
```

Build with:

```
$ CGO_ENABLED=0 go build -ldflags '-extldflags "-static"' -o hello ./he
$ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically
$ ldd hello
not a dynamic executable
```

rkt: create the image

To create the image, use the `acbuild` tool:

```
$ acbuild begin
$ acbuild set-name example.com/hello
$ acbuild copy hello /bin/hello
$ acbuild set-exec /bin/hello
$ acbuild port add www tcp 5000
$ acbuild label add version 0.0.1
$ acbuild label add arch amd64
$ acbuild label add os linux
$ acbuild annotation add authors "Carly Container <carly@example.com>"
$ acbuild write hello-0.0.1-linux-amd64.aci
$ acbuild end
```

This creates an `ACI` containing the application code and the needed metadata.

- advantage *wrt* a `Dockerfile`: this can **easily** and **seamlessly** be integrated into `Makefile` -based workflows

rkt: run the container

```
$ rkt --insecure-options=image run hello-0.0.1-linux-amd64.aci
image: using image from local store for image name coreos.com/rkt/stage
image: using image from file hello-0.0.1-linux-amd64.aci
networking: loading networks from /etc/rkt/net.d
networking: loading network default with type ptp
```

in another terminal:

```
$ rkt list
UUID        APP        IMAGE NAME                    STATE    NETWORKS
37e2ee52    hello    example.com/hello:0.0.1        running    default:

$ curl http://172.16.28.4:5000
hello
```

Astro/HEP application

Application to Astrophysics/Cosmology

Imagine a simulation program that is actually an assembly of many multiple programs:

- a C++ library (compilable with a specific version of `g++`)
- a `FORTRAN` library (compilable with a specific version of `gfortran`)
- `python2` bindings
- `LAPACK`, `BLAS` & `Cython` dependencies

After a while you realize, this was only tested on a specific `Ubuntu` version. And people want to run or at least develop on their laptops (`Linux` and/or `MacOSX`)...

Application to Astrophysics/Cosmology

```
from ubuntu:14.04

run apt-get update -y && apt-get install -y gcc g++ gfortran \
    python python-numpy cython libblas-dev liblapack-dev \
    make curl git

run mkdir -p /build/jla /build/salt2
run git clone https://github.com/lesgourg/class_public /build/class &&
    cd /build/class && make
run cd /build/jla && \
 curl -O -L http://supernovae.in2p3.fr/sdss_snls_jla/jla_likelihood_v6.
 tar zxf jla_likelihood_v6.tgz
run cd /build/jla/jla_likelihood_v6 && \
 make && make test_jla
run cd /build/salt2 && \
        curl -L http://supernovae.in2p3.fr/salt/lib/exe/fetch.php?media
run cd /build/salt2 && \
        tar zxf snfit-2.4.2.tar.gz && \
        cd snfit-2.4.2 && \
        ls && \
        ./configure && \
        make && make install
run git clone https://github.com/cmbant/CosmoMC /build/cosmomc && \
```

Application to Astrophysics/Cosmology

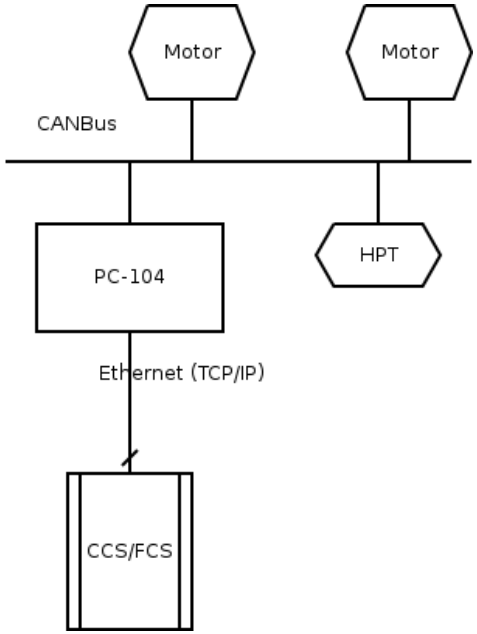
People can then fetch it from the registry (and share their modifications):

```
$ docker pull binet/cosmodev
$ docker run -it -v `pwd`/cosmodev-work:/work binet/cosmodev bash
[cosmodev] run-cosmomc
```

Application to Astrophysics/Cosmology

Imagine a control command application, developed in Java, with many Java specific requirements (version, integrated editor, toolchain, ...)

- you want to closely monitor and control your dependencies
- you want to be able to quickly distribute the development environment



Application to Astrophysics/Cosmology

```
## lsst-ccs/fcs
## A container where all dependencies for FCS are installed.
FROM lsst-ccs/base
MAINTAINER Sebastien Binet "binet@cern.ch"

USER root
ENV GOPATH /go
ENV PATH $GOPATH/bin:$PATH
## install fcs deps
RUN pacman -S --noconfirm awk bash-completion jdk8-openjdk maven mysql
    openssh sed subversion which xorg-server xorg-xclock xorg-xhost lib
## create lsst user
RUN useradd -m -g users -G wheel -s /bin/bash lsst
USER lsst
ENV HOME /home/lsst
## CANOpen will need this port
EXPOSE 50000
## JGroups will need this port
EXPOSE 45566

WORKDIR /opt/lsst
## make the whole container seamlessly executable
CMD ["/bin/bash"]
```

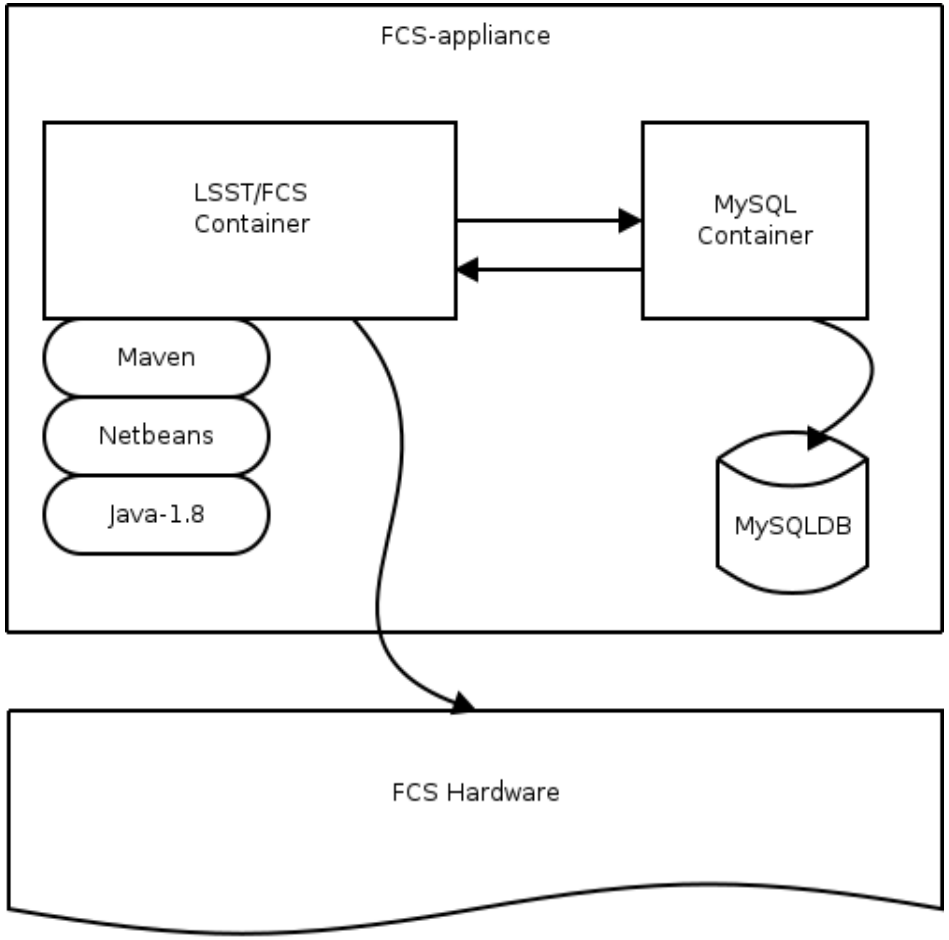
Application to Astrophysics/Cosmology

Actually, this application also needs a database to log commands and monitor data:

- embed the MySQL db into a container
- connect that container with the main container

```
FROM debian:jessie
RUN groupadd -r mysql && useradd -r -g mysql mysql
RUN mkdir /docker-entrypoint-initdb.d
RUN apt-get update && apt-get install -y perl --no-install-recommends &
    rm -rf /var/lib/apt/lists/*
# gpg: key 5072E1F5: public key "MySQL Release Engineering <mysql-build
RUN apt-key adv --keyserver ha.pool.sks-keyservers.net --recv-keys A4A9
ENV MYSQL_MAJOR 5.7
ENV MYSQL_VERSION 5.7.8-rc
RUN echo "deb http://repo.mysql.com/apt/debian/ jessie mysql-${MYSQL_MA
[...]
# share mysql socket
VOLUME /var/lib/mysql
COPY docker-entrypoint.sh /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]
EXPOSE 3306
CMD ["mysqld"]
```

Application to Astrophysics/Cosmology



- Deployable by `Makefile`
- Managed by `git`

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` (`rocket`) from `CoreOS`
- `systemd-nspawn`, now part of `systemd`

References

www.slideshare.net/jpetazzo/introduction-to-docker-december-2014-tour-d
www.slideshare.net/dotCloud/docker-intro-november
sif.info-ufr.univ-montp2.fr/docker-talk
docs.docker.com/introduction/understanding-docker/
wiki.jenkins-ci.org/display/JENKINS/Docker+Plugin
kubernetes.io/
mesos.apache.org/
coreos.com/rkt/docs