Container Technology applied to HEP in High Performance Computing

Isabel Campos -- CSIC (Spain)
Mario David, Jorge Gomes and Jorge Sevilla -- LIP (Portugal)

INDIGO - DataCloud

RIA-653549

CSIC
Consejo Superior de Investigaciones Científicas

LIP
Laboratório de Instrumentação e Física Experimental de Particulas

INDIGO-DataCloud is co-founded by the Horizon 2020Framework Programme

Imagine that Computing & Storage resources are made available as a *"big pool"* in which you don't know which machine is being used to run your code

Would such resource provisioning model be suitable/viable for Scientific Computing ?

Motivation

INDIGO - DataCloud

There is no simple answer to that question

- From the **“software dependencies”** point of view, we will see that much progress has been achieved:

  Perhaps there is no fundamental obstacle anymore

- From the **hardware provisioning point of view**, it depends

  What is more efficient for the Computing Center ? as resource provider
  What is more convenient for the scientific users.

A **technical inspection** into current technologies is needed to **propose the right t*rade-off***

Outline of this presentation

INDIGO - DataCloud

- Give an **overview of what is currently developed/tested/available** to enhance access to computing facilities using this technology.
- Review **basics of container technologies**
- **How does it work in practice**: running complex HEP software on Linux clusters (and in general on multi-user environments) using container technology

What implies that “big-pool” for each actor?

- For the Computing Center

  ✓ Deploy the resources using system software that allows such service provision model: Cloud Middleware Frameworks (**CMF**): OpenStack, OpenNebula, etc…

  ✓ Providing the needed interfaces for users to access the resources in a simple way

- For the researcher using the infrastructure

  ✓ Being able to exploit such infrastructures, smoothly and efficiently, implies additional effort on software encapsulation

You will upload your application software to a storage system, from where it will be executed in a transparent environment: the software must be “self-contained” in terms of dependencies.

  ✓ Success at this level has an impact: seamless access to resources

At Computing Centers it is technically straightforward

INDIGO - DataCloud

Deploying computing resources using Cloud Middleware Frameworks:

- Using **open source interfaces** (OCCI) (i.e. no propietary interfaces like EC2)
- Provide Large amounts of Storage: always a bit more complex due to the unclear standardization.

Currently **efforts on providing an open source interface (CDMI) ongoing**
as the “de facto standard” is S3 (Amazon closed source interface)

It allows providing such resources in very elastic ways

Manage the resources using system software like ***Infrastructure Manager (IM)****, **that allows users asking for the deployment of a private sub-cluster** within a global facility:

- **Run interactively on that sub-cluster serial and MPI jobs**

(*) See https://github.com/indigo-dc/im

At Computing Centers it is technically straightforward

INDIGO - DataCloud

- **Deploying computing resources using Cloud Middleware Frameworks:**
  - Using **open source interfaces** (OCCI) (i.e. no propietary interfaces like EC2)
  - Provide Large amounts of Storage: always a bit more complex due to the unclear standardization.

✓ For this flexibility to translate into a benefit for researchers, in terms of facilitating access to resources, **Application Software developers need to give also a step forward**

Manage the resources using system software like ***Infrastructure Manager (IM)****, **that allows users asking for the deployment of a private sub-cluster** within a global facility:

- **Run interactively on that sub-cluster serial and MPI jobs**

(*) See https://github.com/indigo-dc/im

Containers Technology

The future of software provisioning both in industry and academia

https://access.redhat.com/articles/1353593

The Linux Kernel & containers support

INDIGO - DataCloud

- "Linux Containers" is a technology provided by the Linux kernel, to **"encapsulate" a group of processes in an independent execution environment**: "container"

- It offers an environment as close to possible from a Virtual Machine, but **without the overhead that comes with running a separate kernel** and simulating all the hardware.

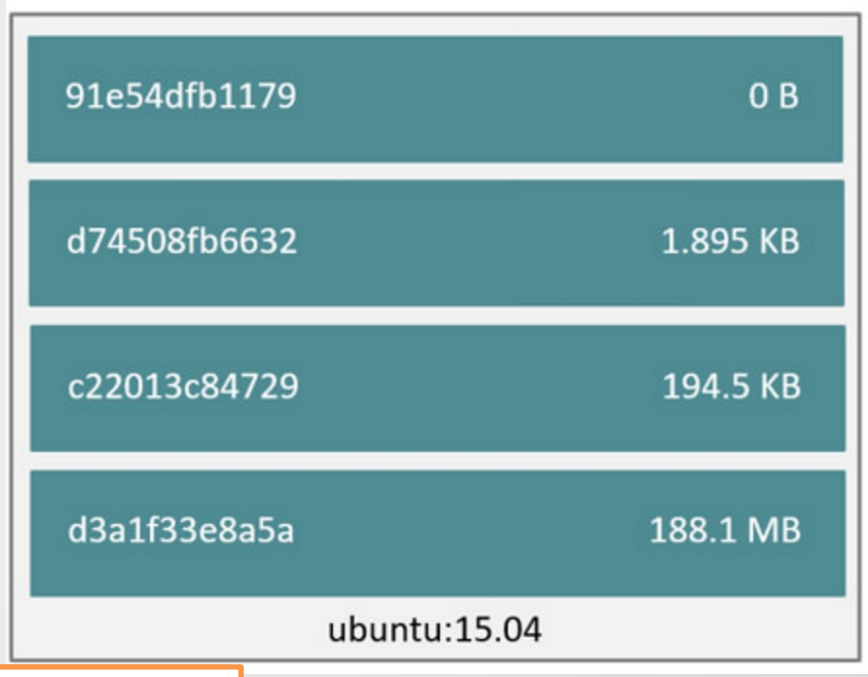
- It relies in **built-in kernel features** (**full functionality** if **kernel >= 3.12**):
  - ***cgroups*** for assignation, limitation and prioritization of resources to "groups" of processes
  - ***namespaces*** to isolate "groups" regarding process trees, networking, user IDs and mounted filesystems: each "group" has the illusion of being the only process running in the system.

The Linux Kernel & containers support

INDIGO - DataCloud

To use containers one needs:

- **An image**: a copy of the entire state system, stored in a file (the image file)
  - They are Snapshot of the OS modified to run as a “container”
  - In particular it has no kernel.
- **A “container engine”** to run that image as a container in the kernel host



One typically downloads an existing image from an image repository, made available by the Container Engine we are using

The Linux Kernel & containers support

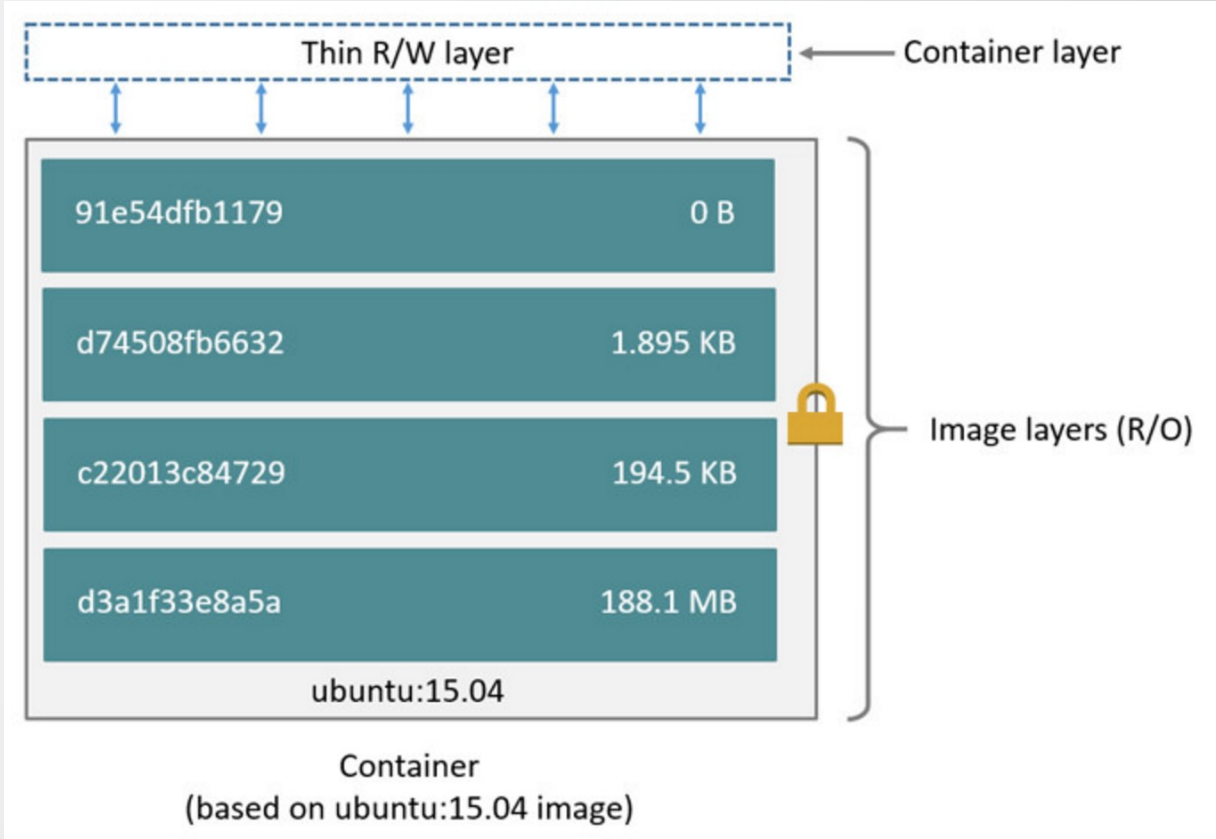
INDIGO - DataCloud

- **The “container engine runs the image**
- **It creates a R/W layer the user can modify:**

Copying there your application software i.e. the environment you to have a fully encapsulated application

- When you are done, stop the container, and save your new image.

and you have containerized your application

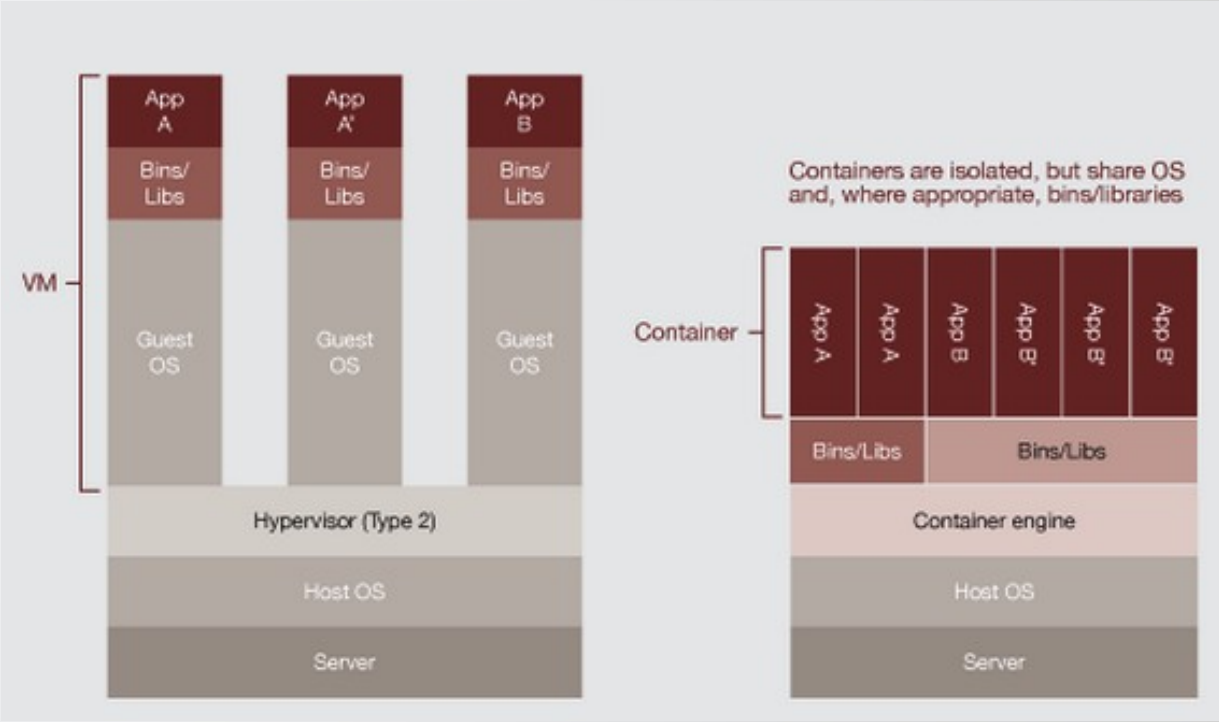


“Images” again ?

INDIGO - DataCloud

“container image” ≠ “VM image”

- In particular, when comparing with an image for a hypervisor (VM), it has no Kernel.
- Also other features have been removed (because the container image uses the kernel of the host machine, and therefore can profit from certain parts
- Still they are large, minimum 500MB (libraries…)
- Those “images” will not run when launched with a hypervisor !



The Linux Kernel & containers support

INDIGO - DataCloud

LXC as Container Engine

- LXC exposes a set of libraries and tools to work with containers
- Packages are available for all main Linux distributions:

yum install -y lxc lxc-templates debootstrap bridge-utils
apt-get install lxc

Container network connectivity is handled by a local Linux bridge by default.

See:
https://linuxcontainers.org/lxc/getting-started/

Download a container from the list of the available ones with:

lxc-create -t download -n my-container

Start it: ***lxc-start -n my-container –d***

Get a shell inside: ***lxc-attach -n my-container***

Stop it: ***lxc-stop -n my-container***

Remove it: ***lxc-destroy -n my-container***

Docker

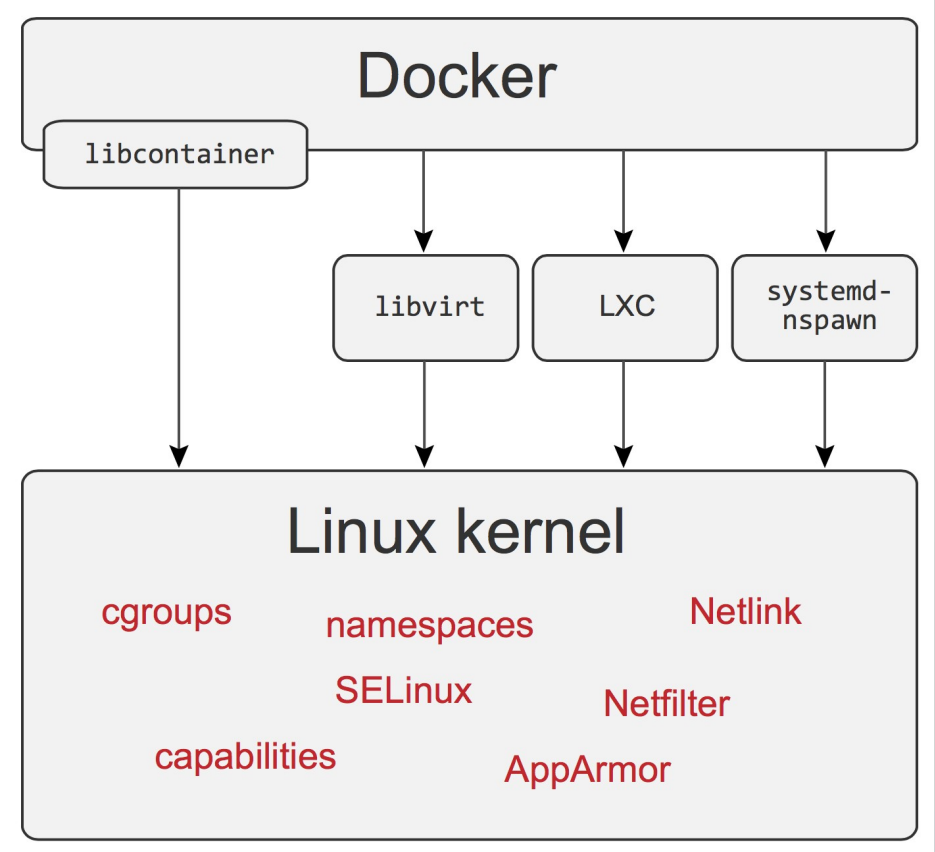
- Docker can be seen as an **advanced packaging tool for container images**

- It provides a very advanced repository of images ready
- for container running, with capabilities such as image tagging, version control, etc…

  https://hub.docker.com

- It offers *autobuild* features, which allow re-creation of images on the fly, via cloud-services offered, for free, by docker
  - You can c**onnect the dockerhub**, with your favourite repository: **github, gitlab,…**

- They introduced the **Dockerfile concept**



Docker - Dockerfile

INDIGO - DataCloud

Docker can build images automatically by reading the instructions from a Dockerfile.

A **Dockerfile is a text document** that contains all the commands to assemble an image.

Using ***docker build*** users can create an image out of a Dockerfile

```
#
# Super simple example of a Dockerfile
#
FROM centos:latest
MAINTAINER Isabel Campos "isabel.campos@csic.es"

RUN apt-get update
RUN apt-get install gcc
RUN wget https://www.open-mpi.org/software/ompi/v2.0/downloads/openmpi-2.0.0.tar.gz

RUN tar xvf openmpi-2.0.0.tar.gz
RUN cd openmpi-2.0.0
RUN ./configure & make & make install
RUN cd ..

WORKDIR /home/isabel
RUN wget http://luscher.web.cern.ch/luscher/openQCD/openQCD-1.4.tar.gz
RUN tar xvf openQCD-1.4.tar.gz
RUN cd openQCD-1.4/main
RUN make clean & make
```

Practical Example:

Mastercode

Phenomenology Tools

cern.ch/mastercode

- **Collaborative effort** between Experimentals and Theorist in High Energy Particle Physics

The code takes as input experimental data coming from Particle Accelerators and Astrophysics observations, in order to build a consistent model of Nature explaining the experimental data

✓ It focuses on the search for Supersymmetric models

It is supported by an ERC Advanced Grant:
"Exploring the Terauniverse with the LHC, Astrophysics and Cosmology" **(J. Ellis)**

See **http://johne.web.cern.ch/johne/**

Members of the MasterCode collaboration

- E. Bagnaschi, O. Buchmüller, R. Cavanaugh, M. Citron, A. De Roeck, M. Dolan, J. Ellis, H. Flächer, S. Heinemeyer, G. Isidori, J. Marrouche, D. Martinez, K. Olive and K. Sakurai

Codes involved

- **RGE running: SoftSUSY**
- **Higgs, (g-2)μ: FeynHiggs**
- **Higgs observables: HiggsBounds and HiggsSignal**
- **B-physics: SuFla**
- **B-physics: SuperIso**
- **EWPO: FeynWZ**
- **Dark matter observables: Micromegas**
- **Dark matter observables: DarkSUSY**
- **Recast of LHC searches: Atom**
- **Recast of LHC searches: Scorpion**
- **SUSY decay modes: SDECAY**

Our pilot case: Mastercode

- Mastercode is an “über-code” writen in C++ which connects all the different codes
- It does **parametric runs**: scanning through large parameter spaces, in **single core** mode
- Those original codes are treated as subroutines, or sub-codes
- The sub-codes are writen in C++ or Fortran. Many different authors, often legacy code is there….
- **Perfect candidate to be “containerized”**: problems to run the code in current computing centers because it is difficult to install (library dependences and compatibility issues with local software, etc…)
- One of the main problems they have: **finding a computing center where it works!**

Using docker to build a container for Mastercode

INDIGO - DataCloud

Description of the automated process via github + dockerhub:

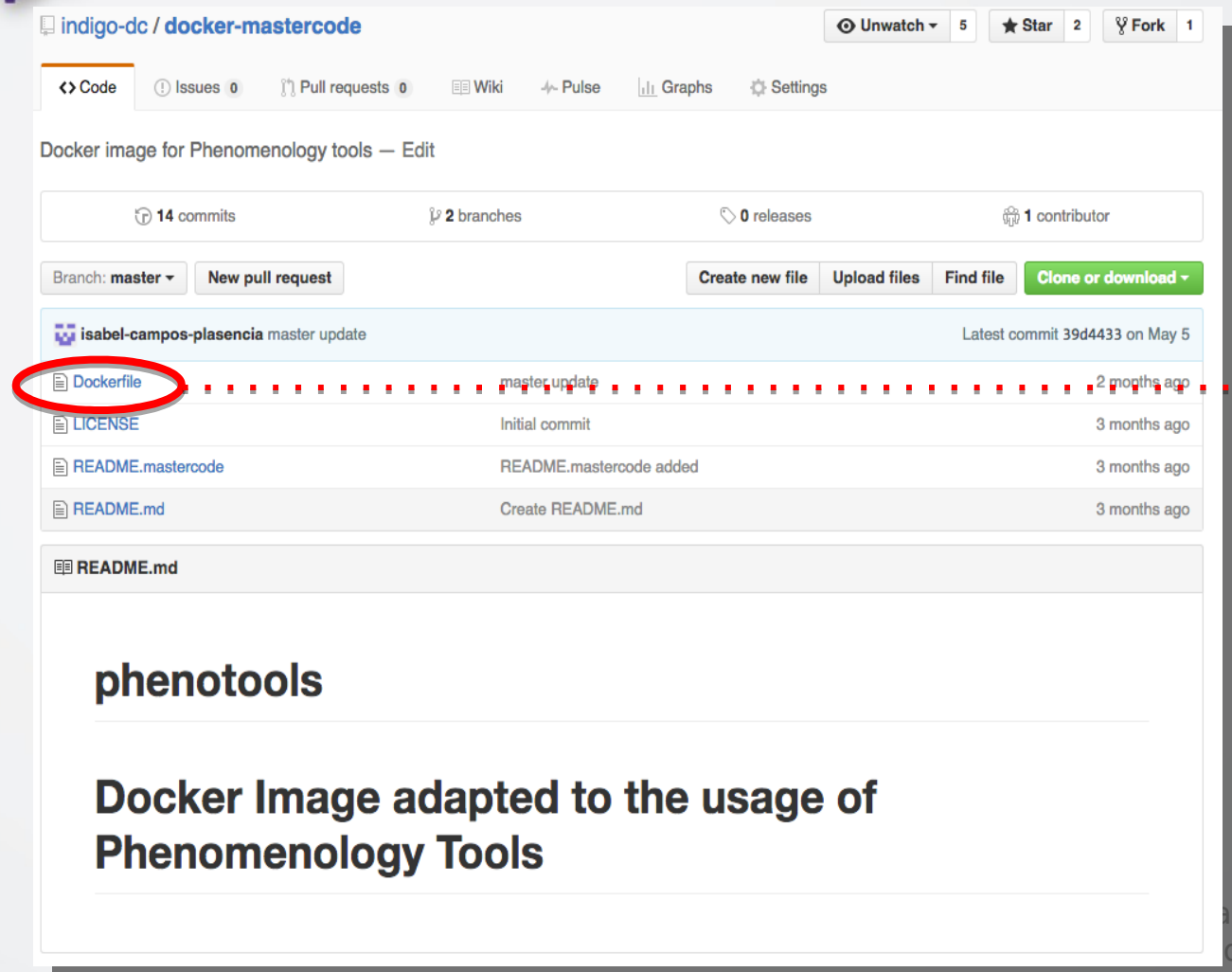
1. We have created a project in github: https://github.com/indigo-dc/docker-mastercode

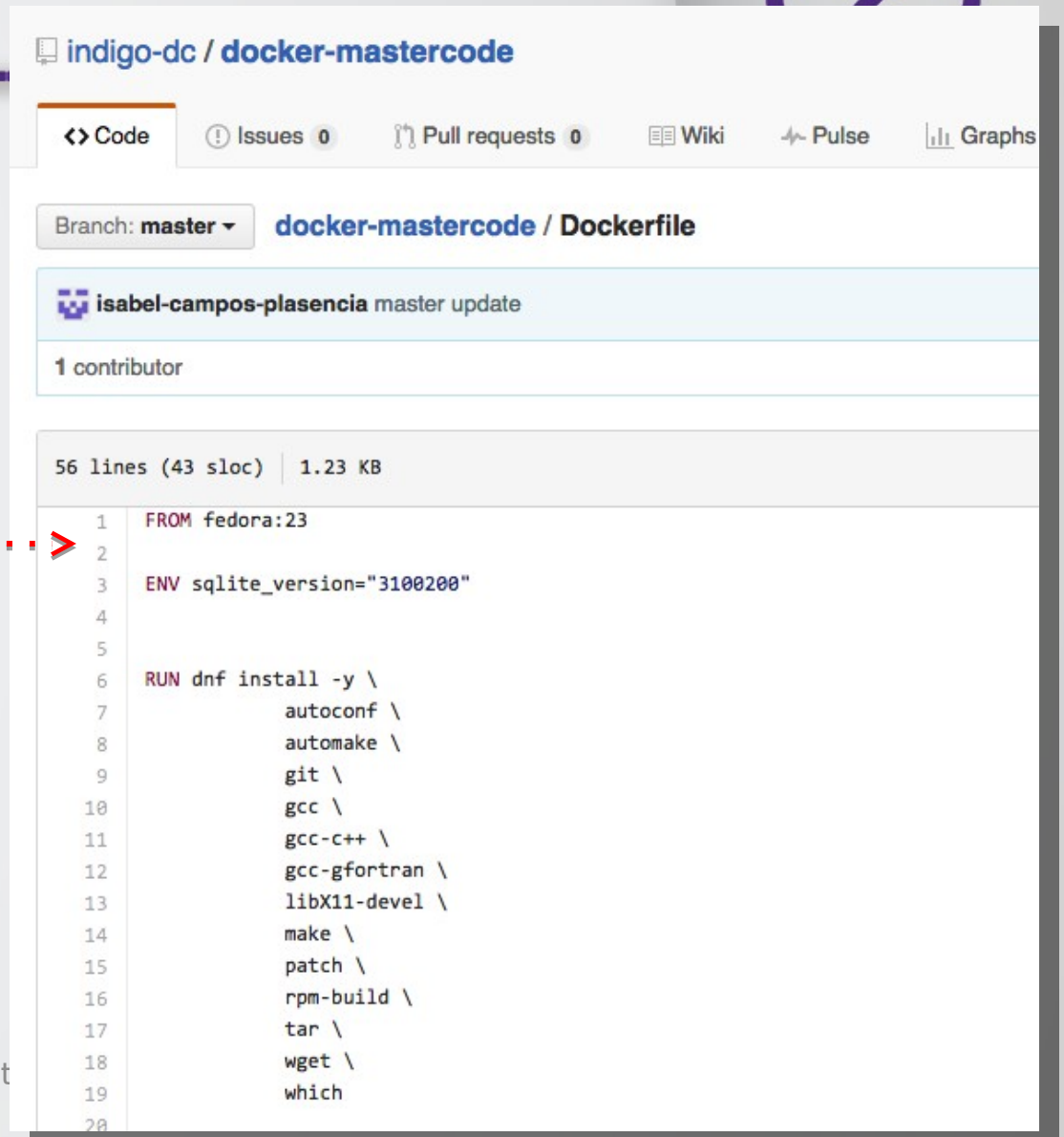
2. The project contains just a **Dockerfile** to build a **fedora-based container** that includes all the libraries and system software needed to run Mastercode

3. We created a hub in docker for the container to be built automatically in the dockerhub cloud: https://hub.docker.com/r/indigodatacloud/docker-mastercode

4. We connected github-docker to automatically build the container

Github project & Dockerfile





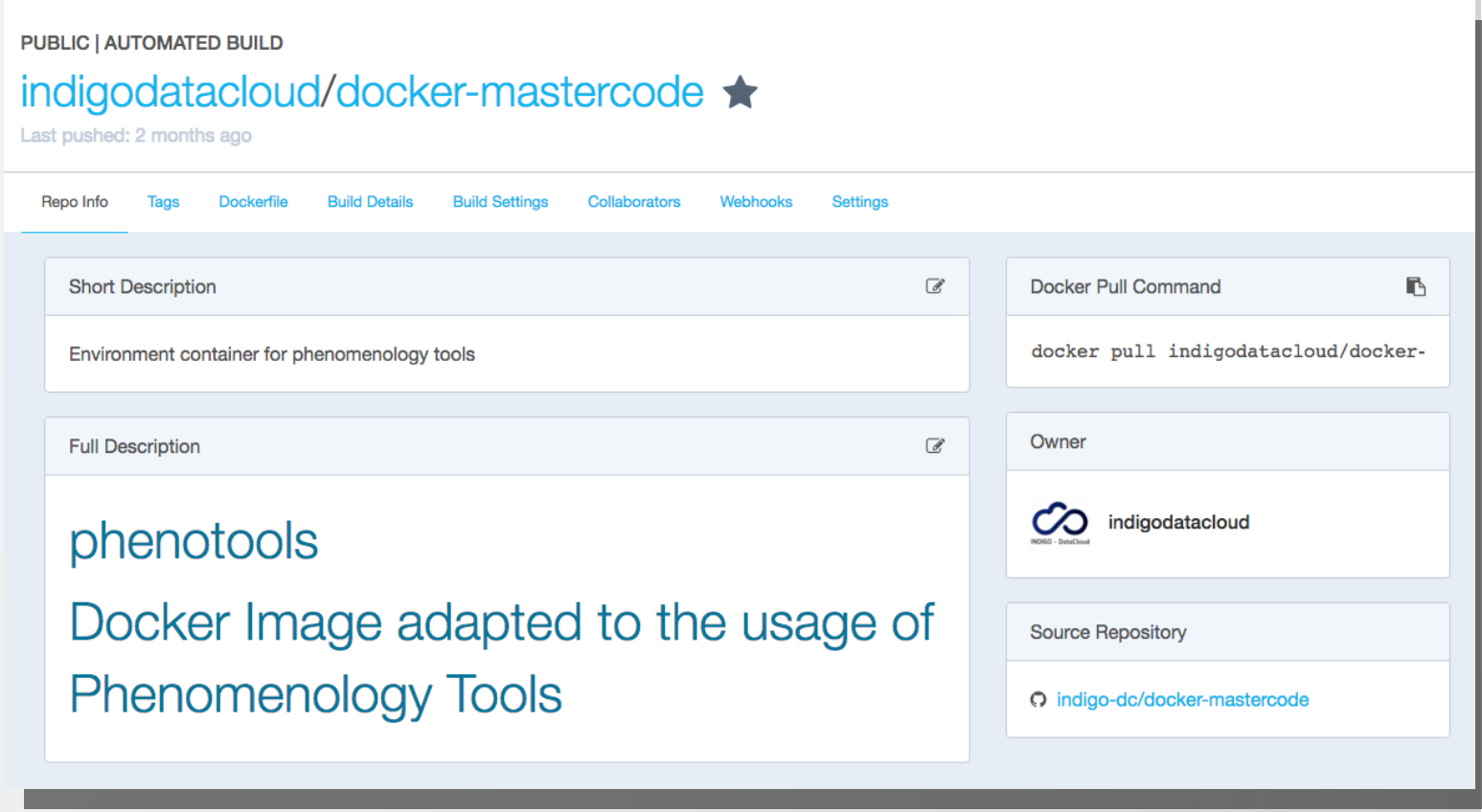
Connect Github with the Dockerhub

We created a project in **dockerhub**, which reads as **source repository** our **github Mastercode** repository.

In particular **dockerhub is aware of the changes committed to the Dockerfile**

Dockerhub provides a service to build the containers automatically on their cloud facilities (not very fast, but for free).

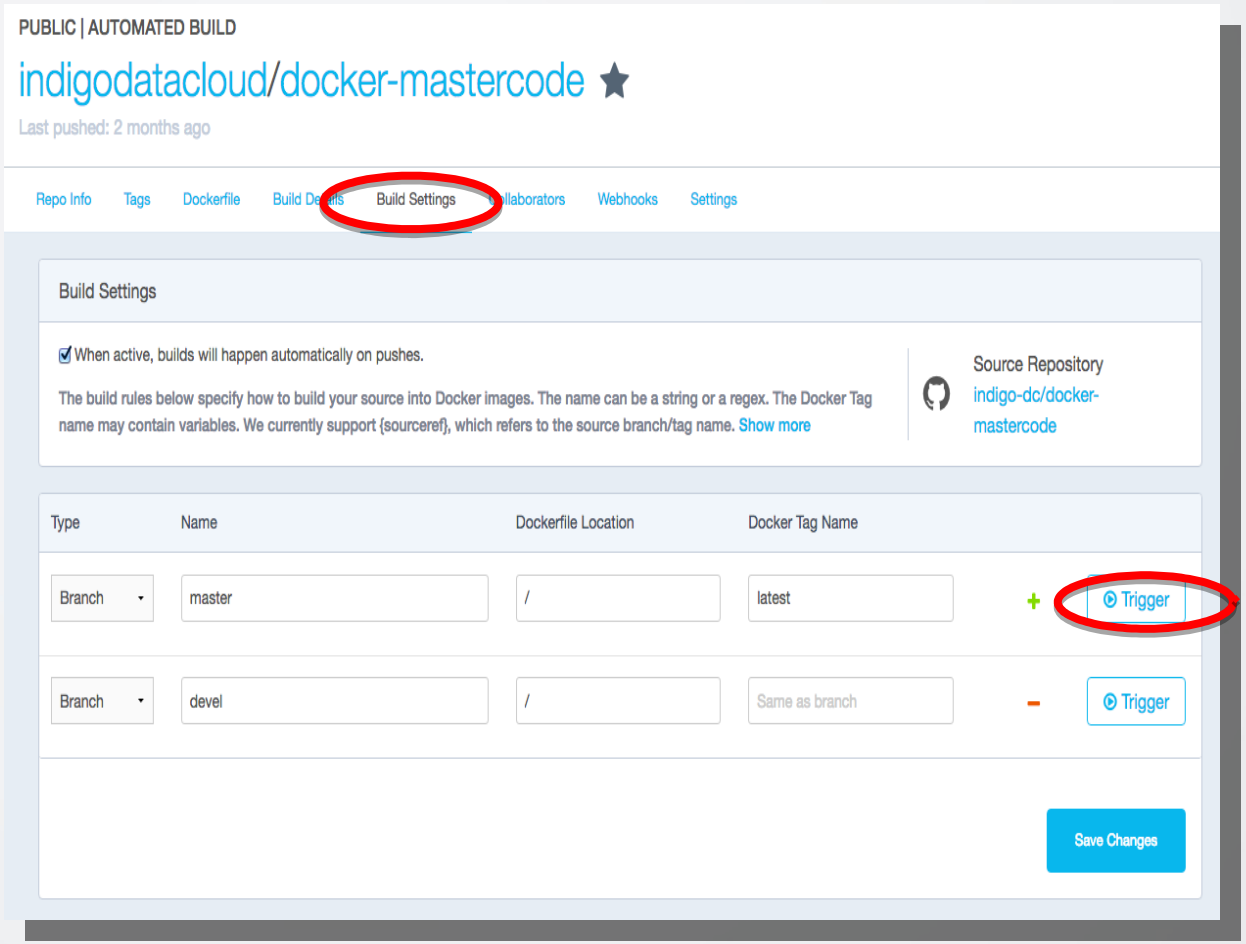
We use this service to maintain up-to-date The container for Mastercode

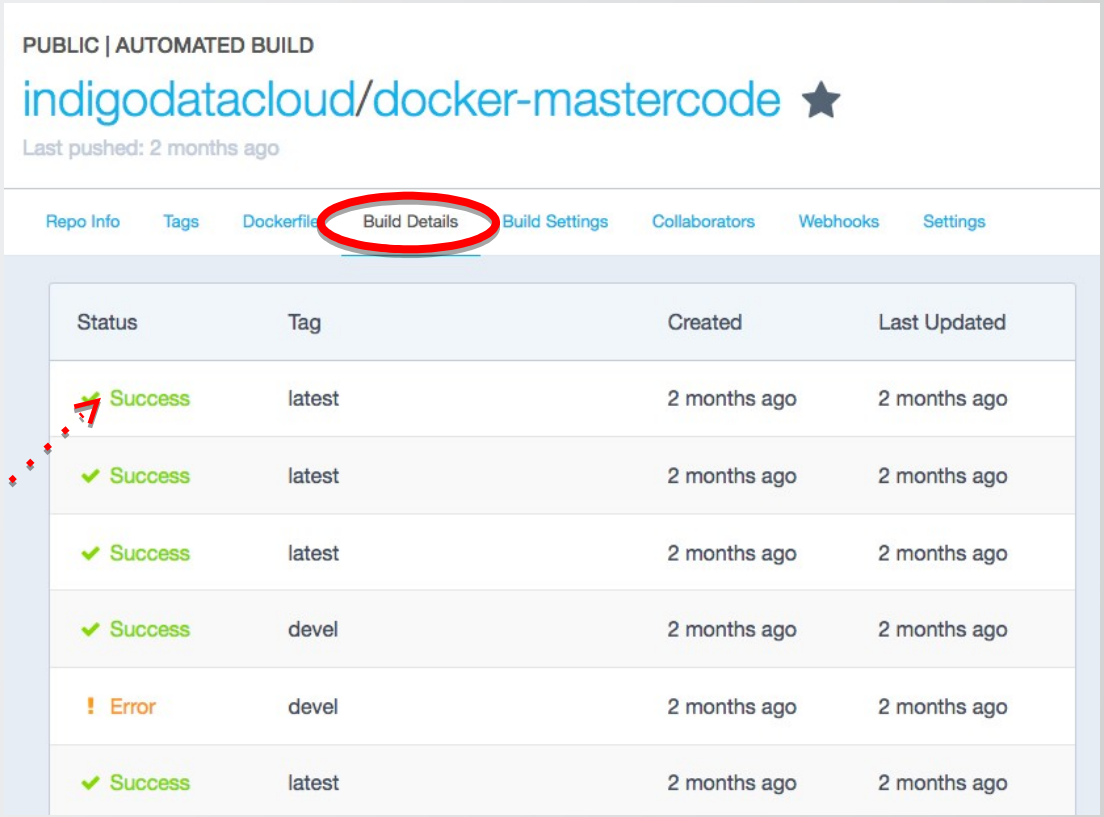


Connect Github with the Dockerhub - II

INDIGO - DataCloud

A change in the dockerfile, triggers automatically a new build of the container





Using the Mastercode container with docker

INDIGO - DataCloud

On your local Linux box, **with root user privileges**:

1. Install docker with your favourite package manager (eg. "***# yum install docker***")
2. Start the docker daemon: ***# systemctl start docker***
3. Download the container: ***# docker pull indigodatacloud/docker-mastercode***
4. *Run it*

```
# docker run -t –i  -v $HOME \                  mounts your HOME directory when running the container
-w $HOME/mcpp-master \                 Mastercode sources directory is set as working directory
indigodatacloud/docker-mastercode \    Name of the container to run
/bin/bash                              Get a bash shell inside the container
```

Notice: the container does not contain the code, only the environment necessary to run it:

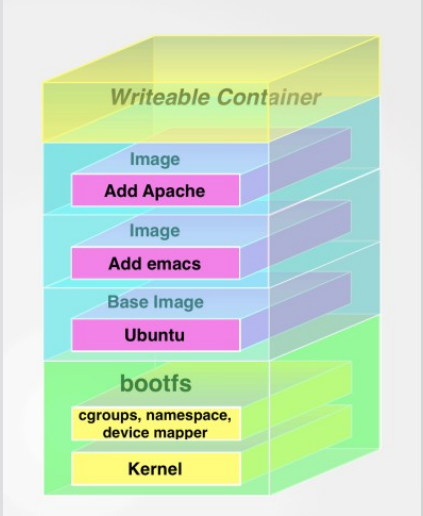
- Developpers keep doing their own modifications to Mastercode in their private copy.
- In **MacOS,** download the Docker Tool Box for Mac, and start from (3).

Docker limitations

- **The Union Filesystem** poses a **big constraint on container sizes**:
  - The size of a container, is **monotonously increasing with time**…. no matter what you delete.
  - Decreasing a container size with docker, can be done only by re-creating it

- The **docker "root"** has limited capabilities:
  - Cannot access /dev (try your audio/video) inside a container
  - Cannot mount filesystems external to the container (unless you dissable SElinux),…

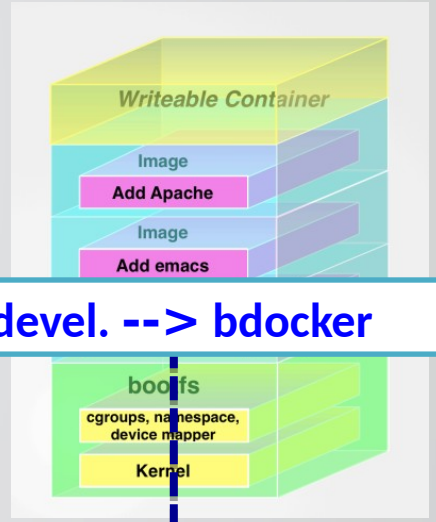
- On **multi-user environments** (eg. a Linux cluster) with a batch system:
  - Docker runs formally under root
  - There are issues with traceability of the processes (who did what?)
  - Accounting of resources is very complex/impossible
  - Could a kernel bug in the container, be exploited to attack the host machine?



Docker limitations

- **The Union Filesystem** poses a **big constraint on container sizes**:
  - The size of a container, is **monotonously increasing with time**…. no matter what you delete.
  - Decreasing a container size with docker, can be done only by re-creating it

- The **docker "root"** has limited capabilities:
  - Cannot access /dev (try your audio/video) inside a container
  - Cannot mount filesystems external to the container (unless you dissable Selinux)



INDIGO devel. --> bdocker

- On **multi-user environments** (eg. a Linux cluster) with a batch system:
  - Docker runs formally under root
  - There are issues with traceability of the processes (who did what?)
  - Accounting of resources is very complex/impossible
  - Could a kernel bug in the container, be exploited to attack the host machine?

- ✓Solves the “root” problem by a token mechanism
- ✓Provides proper accounting of resources to the scheduler

We can run docker under batch systems

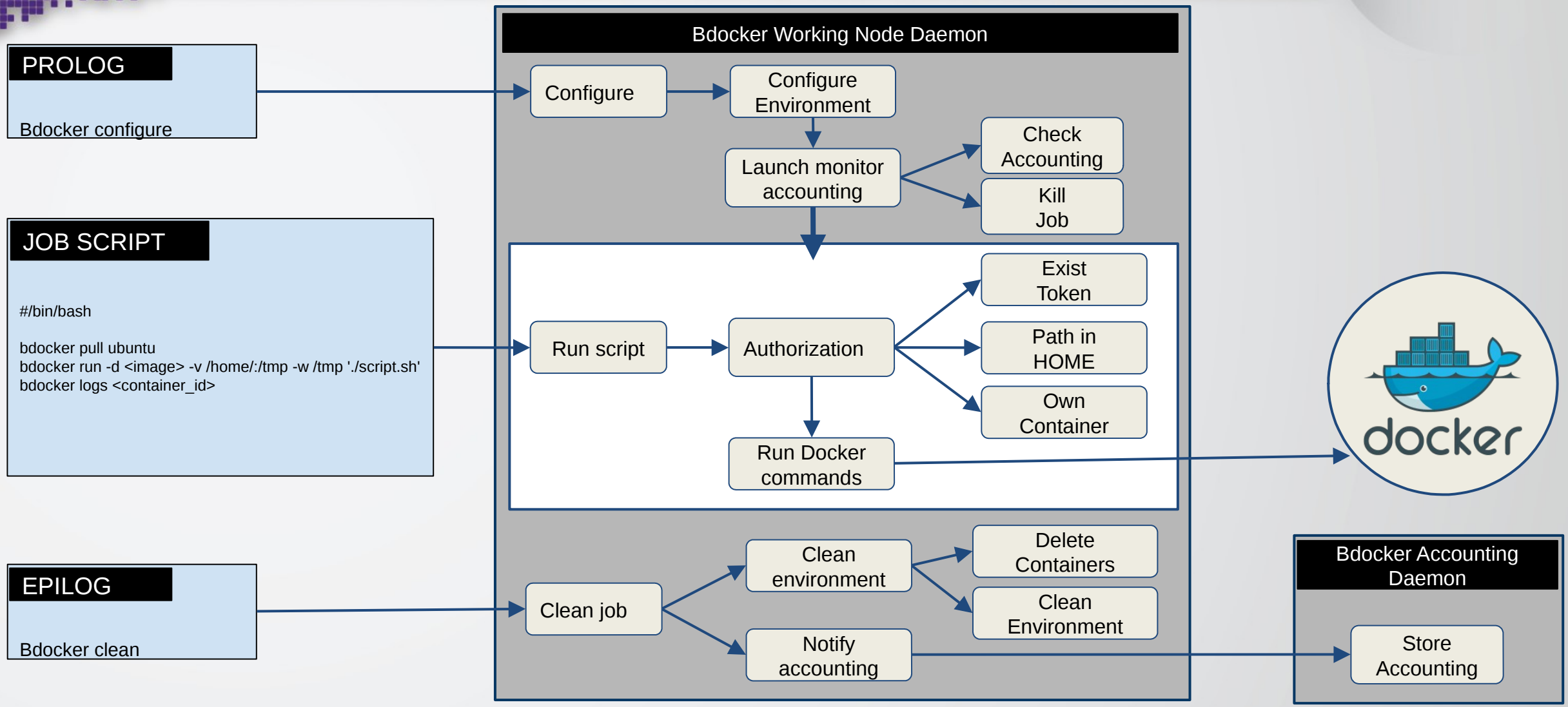
We have developed *bdocker*

- **Bdocker allows submission of batch jobs running containers to linux clusters running docker as part of their system software deployment.**
- Two daemons are working under root privileges
  - Working daemon controls the docker execution on each working node.
  - Accounting daemon listens the working nodes and stores job accounting.

- Command line tool called 'bdocker'
  - Follows the same semantic that the docker command line client
    - **./bdocker run -d my_image_name -v /home/jorge/:/tmp -w /tmp './script.sh'**

  - Two **administration commands to be executed by prolog/epilog**.
    - bdocker configure
      - Configure user credentials and batch environment.
    - bdocker clean
      - Clean user credentials and batch environment, and notify accounting.

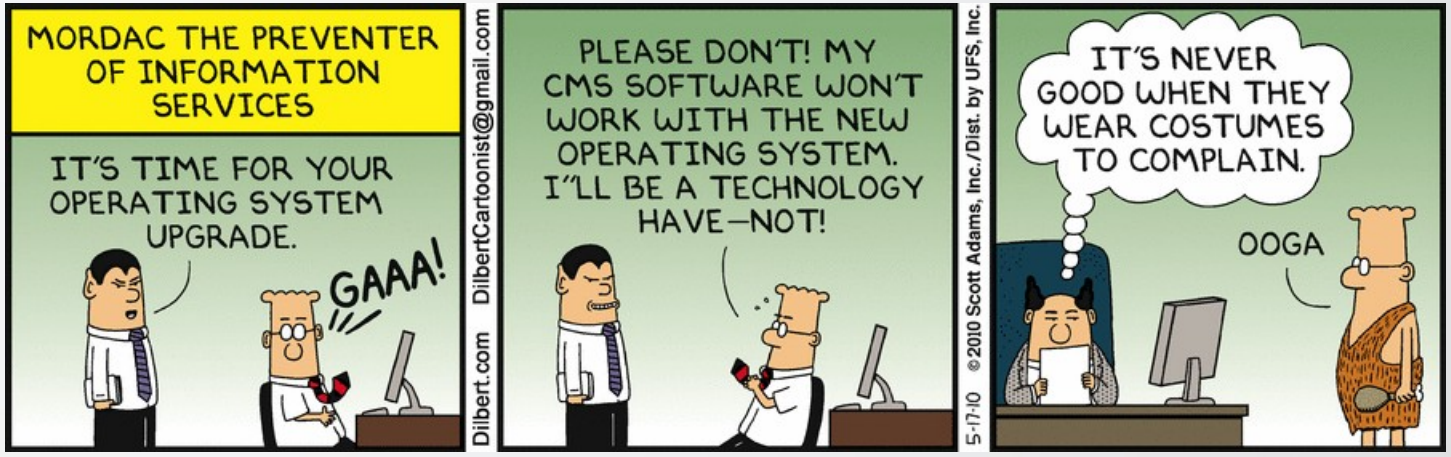
Bdocker Workflow



When docker is not available in the computing center…

In HPC centers / multi-user environment

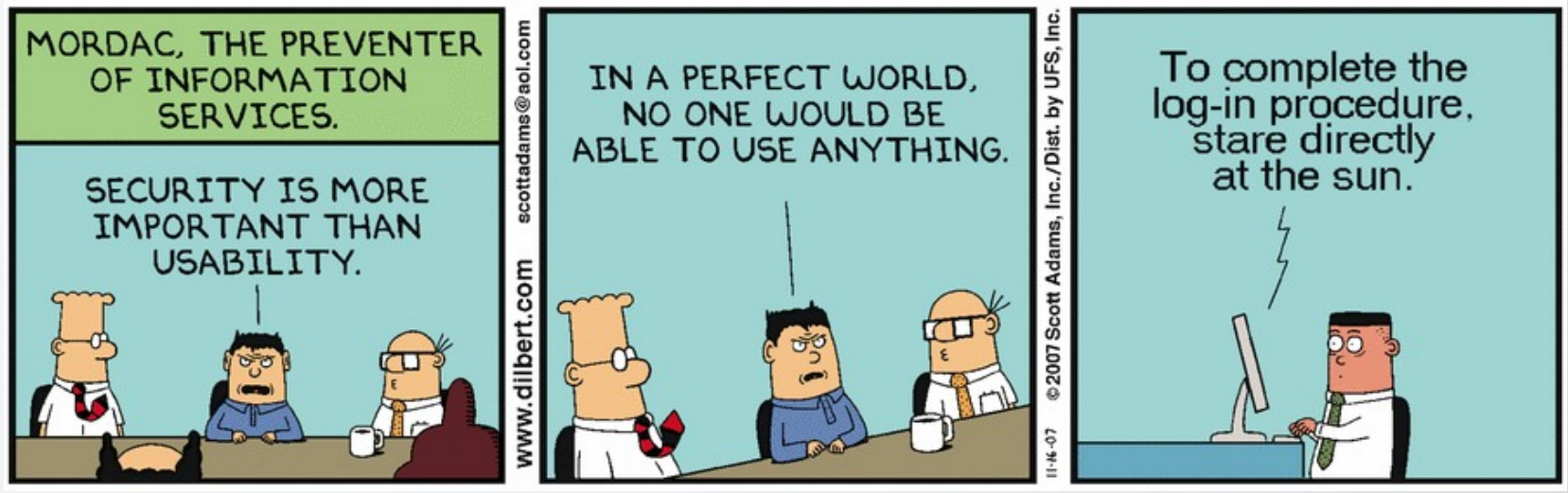
- Adoption of docker is being very slow in HPC centers
- Thus the tipical situation is that docker is not installed and one cannot run containers without some support from the system software.
- In general Docker adoption will be slow in any computing farm or interactive linux system **shared by many users** eg. **conflicting software dependencies**…



In HPC centers / multi-user environment

INDIGO - DataCloud

- Adoption of docker is being very slow in HPC centers
- Thus the tipical situation is that docker is not installed and one cannot run containers without some support from the system software.
- It **will take time for sysadmins to overcome the concerns of the security teams.**



In HPC centers / multi-user environment

- Adoption of docker is being very slow in HPC centers. Why?
- The tipical situation is that docker is not installed and one cannot run containers without some support from the system software.

…. **yet another service to maintain…**

For those not willing to deal with their particular “Mordac”, we developed *udocker*

INDIGO - DataCloud

udocker: capabilities

- **It is tool to execute content of docker containers in user space** when docker is not available
  - enables download of docker containers from dockerhub
  - enables execution of docker containers by non-privileged users
- **It can be used to execute the content of docker containers in Linux batch systems and interactive clusters managed by others**
- **Acts as a wrapper around other tools to mimic docker capabilities**
  - current version uses **proo**t to provide a **chroot like environment** without privileges (runs on CentOS 6, CentOS 7, Fedora, Ubuntu)

- **More info and downloads at:**
  - https://www.gitbook.com/book/indigo-dc/udocker/details
  - https://indigo-dc.gitbooks.io/udocker/content/doc/user_manual.html

udocker: basic description

- Everything is stored in the $HOME or some other directory belonging to the user (tunable parameter).
- Container layers are download to the above specified directory
- Directory trees can be created/extracted from these container layers
- **proot uses the debugger ptrace mechanism to change pathnames and execute transparently inside a directory tree**
- No impact on read/write or execution, only impact on system calls using pathnames (ex. open, chdir, etc)
- **Does not require installation of software** in the host system:
  - ***udocker*** is a python script
  - ***proot*** is statically compiled

Running Mastercode with *udocker*

INDIGO - DataCloud

1. Download udocker from: https://github.com/indigo-dc/udocker

2. Download the container: $ ./udocker.py pull **indigodatacloud/docker-mastercode**

3. Make sure you have enough space to uncompress it by pointing to a proper directory (default is $HOME): export UDOCKER_DIR=/MY_LARGE_FILESYSTEM/userabc/.udocker

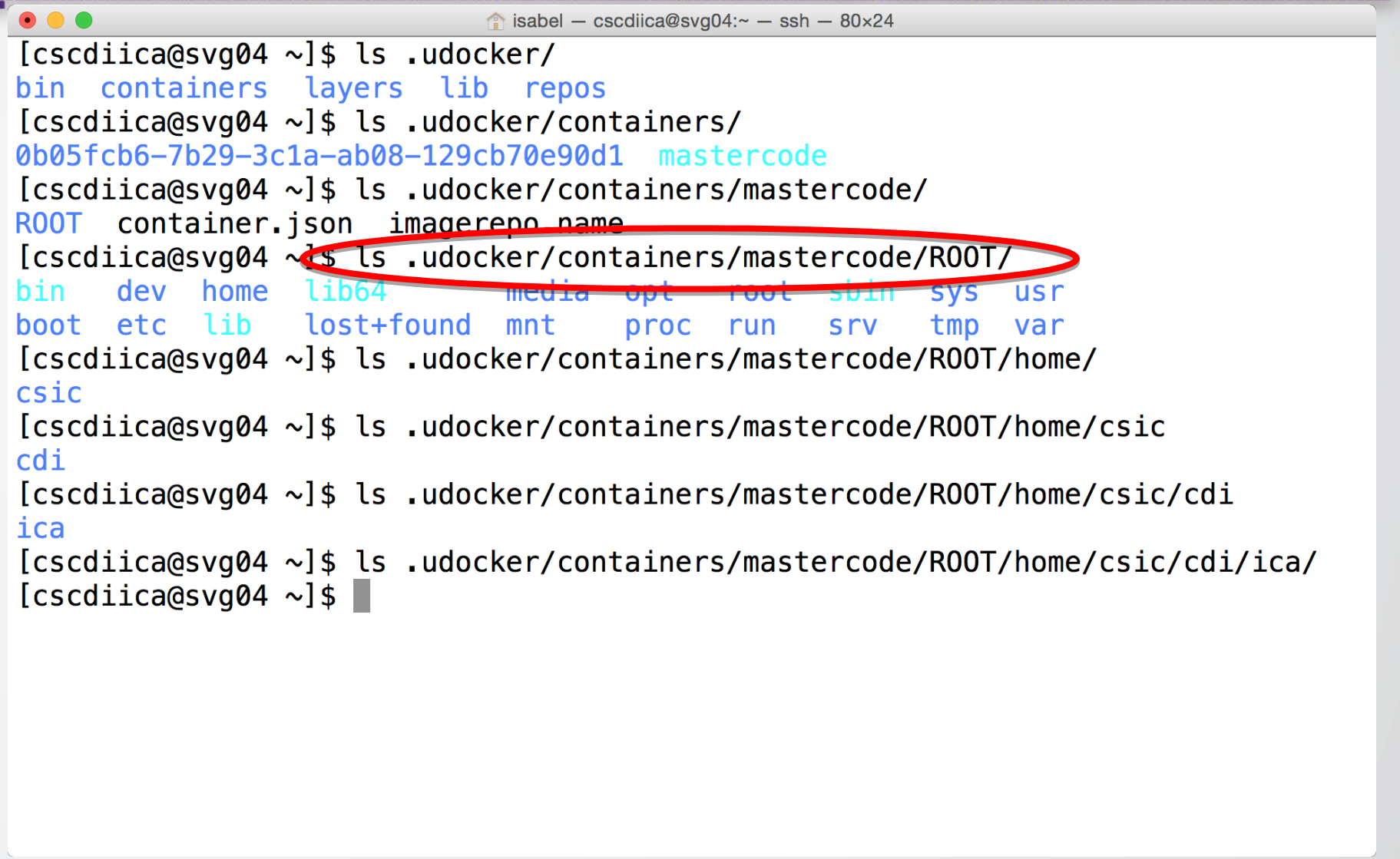
4. Create the container directory tree on your user space

$ ./udocker.py create indigodatacloud/docker-mastercode
bb889c79-2872-37f3-adad-cd9e937dd6f0

5. You probably want to give it a nicer name:

$./udocker.py name bb889c79-2872-37f3-adad-cd9e937dd mymastercode

Running Mastercode with udocker: directory tree



Running Mastercode with udocker: Linux Cluster: batch script


```
export MASTERDIR=/gpfs/csic_users/userabc/mastercode
export UDOCKER_DIR=$MASTERDIR/.udocker

 ../udocker-master/udocker.py run --hostauth \
 -v /home/csic/cdi/ica/mcpp-master \
 -v /home/csic/cdi/ica \
 -user=userabc \
 -w /home/csic/cdi/ica/mcpp-master mastercode \
 '/bin/bash -c "pwd; ./udocker-mastercode.sh"'
```

-hostauth : to use the /etc/passwd of the host machine

-v makes directories available inside the container

-user: your userid

-w working directory from where the commands will be issued

Where **udocker-mastercode.sh** is the (usual) command line sequency to execute Mastercode

```
./mc_point.py --run-mode mc-cmssm --predictors all --inputs 500 600 0 10 --print-mc-spectrum > output.txt
```

Support to MPI execution (under dev.)

INDIGO - DataCloud

- Seems it **will work as well**
- We noticed that:
  - We are able to modify the agents so that each MPI process is launched as a container
  - Under such conditions individual containers can be instructed to communicate via MPI libraries, as any regular MPI execution does.
  - In our preliminary tests, it works over Ethernet and yes, under Infiniband too.
- Will be **part of our 2nd software release** in February 2017:
  - If you want to help us testing, let us know (beta-testers very much wanted)

Conclusions

- Thanks to recent system software developments it became much easier exploiting computing resources in a transparent way
  - It needs to be kept in mind when developing new application software packages

- Cloud-like resources are optimized exclusively for this type of usage: many resources,
  - But one must get smarter to exploit them efficiently.
  - Today it is possible accessing resources in this mode in commercial environments

- As of today Computing Resources are offered still as classic Linux Clusters
  - It will change in time, despite the HPC centers are forced to be very conservative in this respect (production infrastructure)
  - We need to be ready for a smooth transition, and the good news is that the user has new possibilities to be independent from the underlying system software.

Technical References. If you wish to…

Move your infrastructure to a Cloud Middleware Framework:

- Install your infrastructure under openStack and run containers there:
- Provide your cluster users the capability of deploying private sub-clusters on demand for interactive access:

https://www.indigo-datacloud.eu/indigo-support-and-technical-services

Use containers technology, but without moving to a CMF:

- Install support to docker execution via your batch system
  - https://github.com/indigo-dc/bdocker
- Use udocker
  - https://github.com/indigo-dc/udocker

Questions?

INDIGO - DataCloud

"Using udocker we can run the container as an unprivileged user
Without the need of any additional system software"