



# User space containers with udocker



Infraestrutura  
Nacional de  
Computação  
Distribuída



LABORATÓRIO DE INSTRUMENTAÇÃO  
E FÍSICA EXPERIMENTAL DE PARTÍCULAS

Jorge Gomes / Mário David  
udocker@lip.pt



Cofinanciado por:



Fundação  
para a Ciência  
e a Tecnologia

# Help users run in heterogeneous environments

Run applications across Linux systems

Avoiding dependencies on software and sysadmins

A tool to execute containers that is easy to use and deploy

Providing privilege-less methods to execute containers

Empower end-users to leverage containers

# Advantages

## Installation:

- is deployable directly by the end-user
- does not require privileges for installation
- self contained does not require installation of additional software
- does not require compilation just deploy and use
- does not require system administrators intervention to setup
- can be installed from source, release tarball or from PyPI

## Execution:

- as a normal end-user
- runs entirely in user space
- execution regardless of OS functionalities
- respecting normal process controls and accounting
- supports multiple execution methods
- in Linux interactive or batch systems

Ideal to execute applications across digital research infrastructures

# udocker in 4 steps

## 1) Installation:

- **get** the udocker python code
- **untar** the python code into your home directory
- **udocker install** to copy and unpack the execution engines

## 2) Get a container image:

- **udocker pull** to get containers from docker compatible repositories
- **udocker load** to load images in docker and OCI formats
- **udocker import** to import images from tarballs

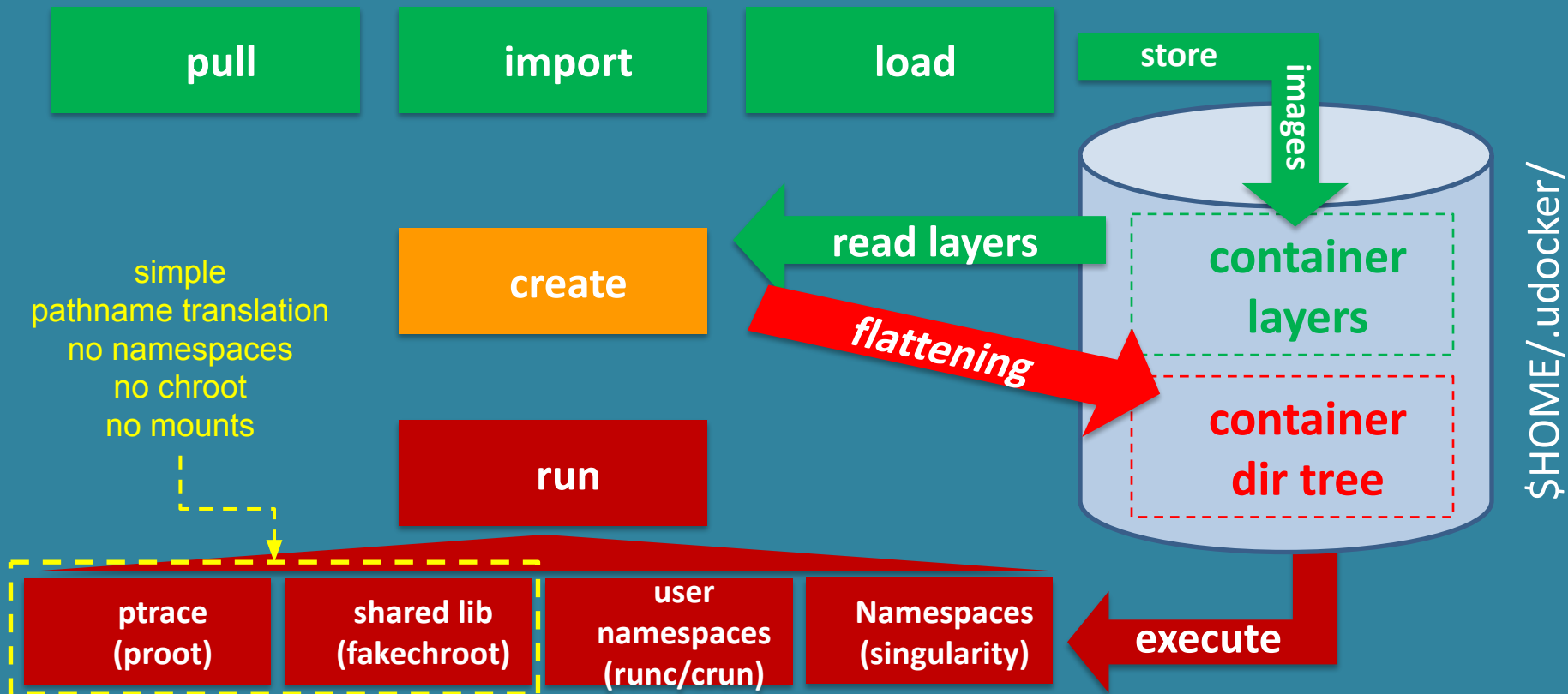
## 3) Extract the image content:

- **udocker create** to extract the container directory tree from the image

## 4) Execute applications from the image:

- **udocker run** to execute using one of the supported methods

# udocker in a nutshell



# Implementation

- Front-end
  - Provides the a command line interface similar to docker and other tools
  - Provides handling of container images (pull, import ,export, load save)
  - Manages a local image and extracted containers repository
  - Provides the interface with the several execution engines
  - Written in Python supports Python 2.6, 2.7 and Python >= 3.5
- Backend
  - Includes external binary tools modified and packaged by the udocker team
  - Both executables and libraries implementing the several engines
  - Compiled statically to enable execution across Linux systems

master 9 branches 21 tags

Go to file

Add file

Code

## About



A basic user tool to execute simple docker containers in batch or interactive systems without root privileges.

[indigo-dc.github.io/udocker/](https://indigo-dc.github.io/udocker/)

docker grid hpc containers  
emulation batch user chroot

<https://github.com/indigo-dc/udocker>

mariojmdavid Merge pull request #371 from indigo-dc/devel3 fd2e6c3 on Aug 26 1,696 commits

.sqa Remove sqa configuration block not required, since we only need trigg... 16 months ago

docs update version last month

etc update variables in udocker.conf 16 months ago

p

t

u

utils tests run containers improved reporting last month

.gitignore add to gitignore, remove link 17 months ago

.mailmap add mailmap 6 years ago

.travis.yml prepare for test and travis 4 years ago

AUTHORS.md update several documents, markdown style check 16 months ago

CHANGELOG.md update version last month

CITING.md update several documents, markdown style check 16 months ago

CONTRIBUTING.md fix typos 16 months ago

Readme

Apache-2.0 license

971 stars

34 watching

106 forks

## Releases 19

udocker 1.3.4 Latest

on Aug 26

+ 18 releases



# Install from a release

```
$ curl -L \
https://github.com/indigo-dc/udocker/releases/download/v1.3.1/udocker-1.3.1.tar.gz \
> udocker-1.3.1.tar.gz
```

## untar the Python code. It is extracted to a directory called udocker

```
$ tar zxvf udocker-1.3.1.tar.gz
```

## optionally add the just created udocker directory to the PATH

```
$ export PATH=`pwd`/udocker:$PATH
```

## install the binaries required to execute containers under \$HOME/.udocker

```
$ udocker install
```

# udocker commands

```
$ udocker help
```

```
$ udocker pull --help
```

<b>search</b>	<b>pull</b>	<b>create</b>	<b>run</b>	<b>images</b>
<b>rm</b>	<b>rmi</b>	<b>rename</b>	<b>rmname</b>	<b>clone</b>
<b>import</b>	<b>export</b>	<b>load</b>	<b>save</b>	<b>inspect</b>
<b>verify</b>	<b>mkrepo</b>	<b>protect</b>	<b>unprotect</b>	<b>setup</b>
<b>login</b>	<b>logout</b>	<b>help</b>		

- udocker is mainly a run-time to execute containers
- Provides a subset of docker commands
- Actual container creation is better performed using docker itself

# Pull and run

```
$ udocker pull quay.io:centos/centos:7
```

```
Info: downloading layer sha256:2d473b07cdd5f0912cd6f1a703352c82b512407db6b05b43f2553732b55df3bc
```

```
Info: downloading layer sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
```

```
Info: downloading layer sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
```

```
$ udocker create --name=C7 quay.io:centos/centos:7
```

```
7464ceb6b-e9c6-3eb0-9646-6040092b4367
```

```
$ udocker -q run C7 /bin/cat /etc/redhat-release
```

```
CentOS Linux release 7.9.2009 (Core)
```

```
$ udocker -q run --user=$USER --bindhome --hostauth C7 /bin/bash
```

```
464ceb6b$
```

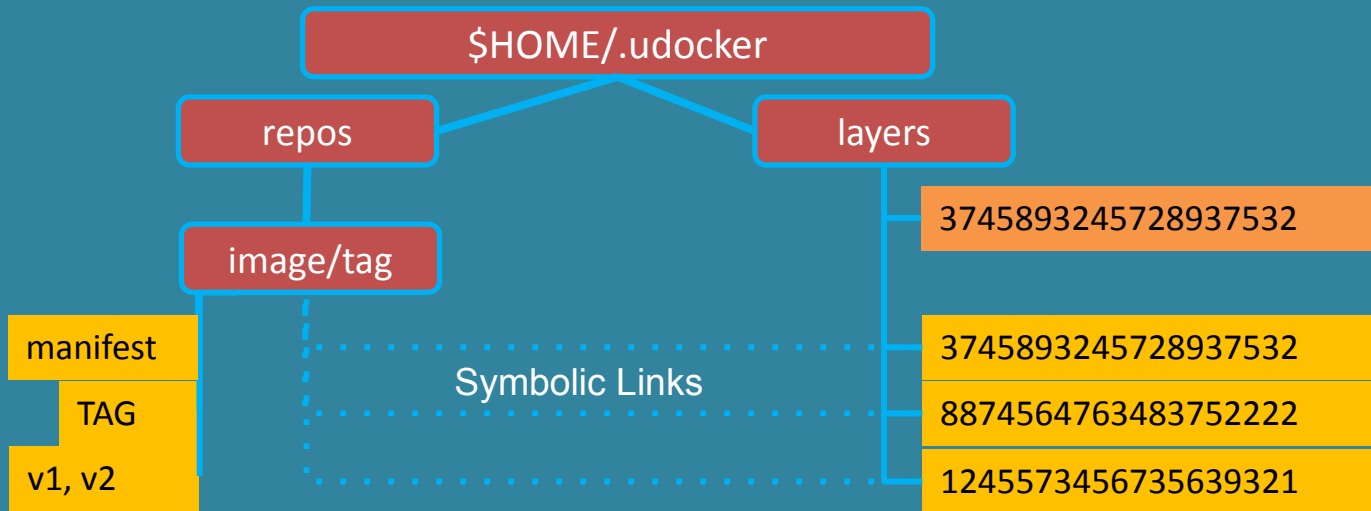
# For the impatient

```
$ udocker -q run --user=$USER --bindhome --hostauth \  
    quay.io/centos/centos:7 /bin/bash
```

```
77af3c48$ pwd  
/home/jorge
```

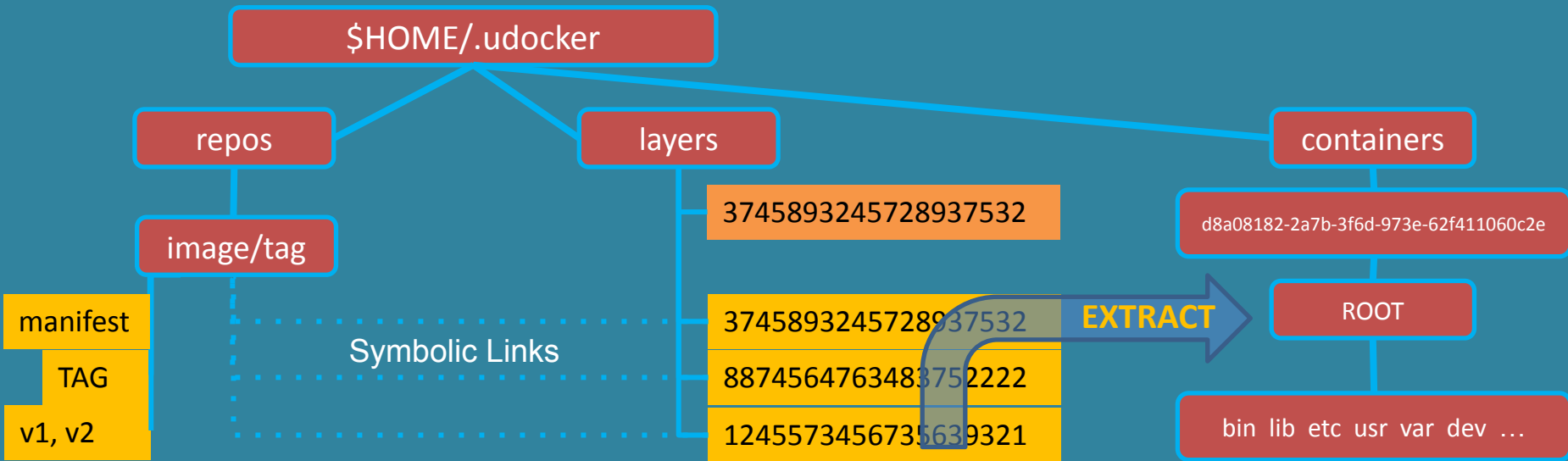
# udocker pull

- Images
  - Layers and metadata are pulled using the DockerHub REST API
  - Image metadata is parsed by udocker to identify the image layers
  - Layers are stored in the use home directory under \$HOME/.udocker/layers
  - Layers can be shared by multiple images



# udocker create

- Containers
  - Are produced from the layers by flattening them sequentially
  - Each layer is extracted on top of the previous
  - The OnionFS whiteouts are respected, and file protections are changed as needed
  - The obtained directory trees are stored under \$HOME/.udocker/containers

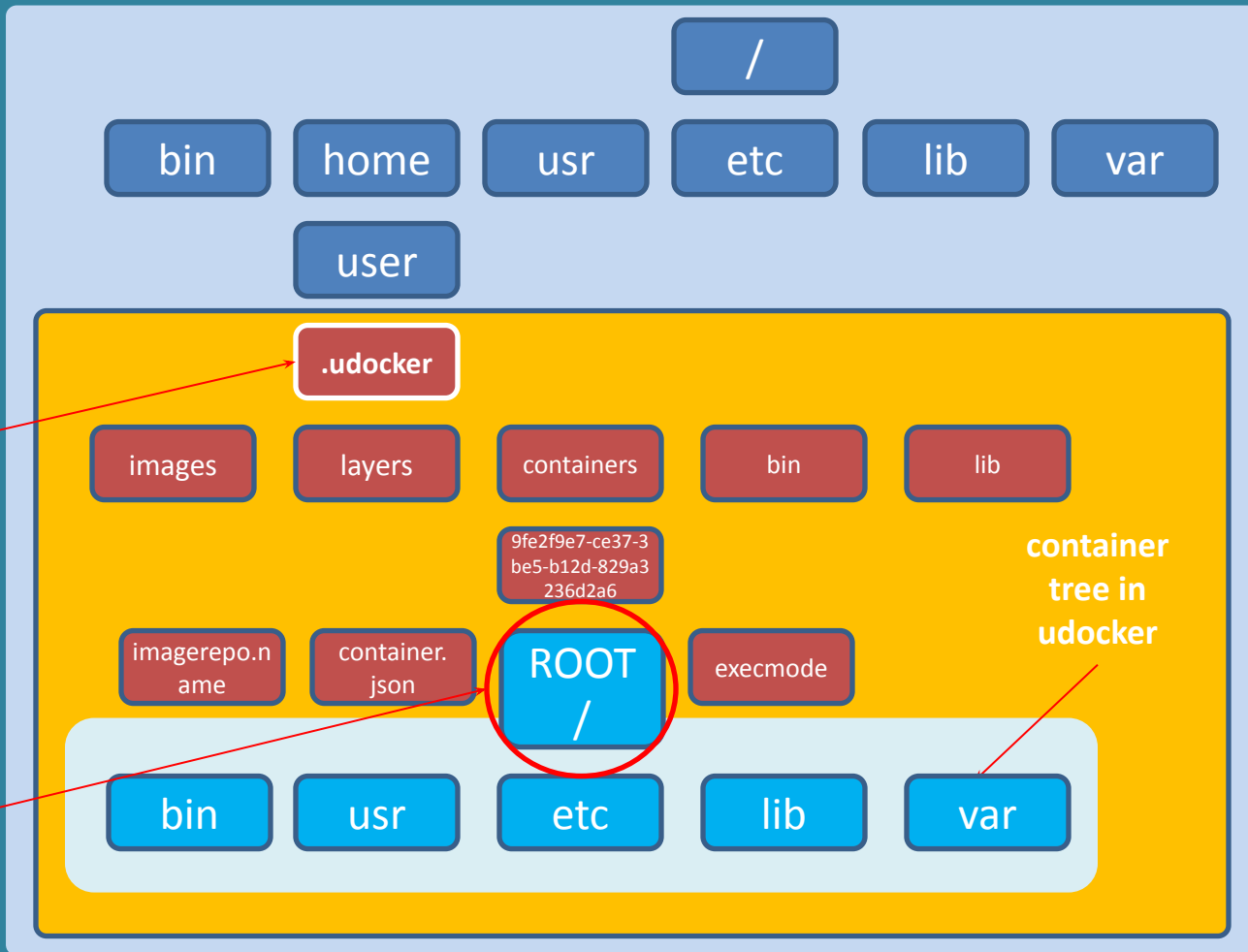


# udocker run

- Execution
- chroot-like

udocker  
directory tree  
\$HOME/.udocker

“chroot” to this directory  
and it becomes the new root  
for the container processes



# Execution engines

Mode	Base	Description
<b>P1</b>	PRoot	PTRACE accelerated (with SECCOMP filtering) <input type="checkbox"/> <b>DEFAULT</b>
<b>P2</b>	PRoot	PTRACE non-accelerated (without SECCOMP filtering)
<b>R1</b>	runC / Crun	rootless unprivileged using user namespaces
<b>R2</b>	runC / Crun	rootless unprivileged using user namespaces + P1
<b>R3</b>	runC / Crun	rootless unprivileged using user namespaces + P2
<b>F1</b>	Fakechroot	with loader as argument and LD_LIBRARY_PATH
<b>F2</b>	Fakechroot	with modified loader, loader as argument and LD_LIBRARY_PATH
<b>F3</b>	Fakechroot	modified loader and ELF headers of binaries + libs changed
<b>F4</b>	Fakechroot	modified loader and ELF headers dynamically changed
<b>S1</b>	Singularity	where locally installed using chroot or user namespaces

```
$ udocker setup --execmode=F3 ub18
```



# Thank you !

Questions ?

`udocker@lip.pt`

`https://github.com/indigo-dc/udocker`

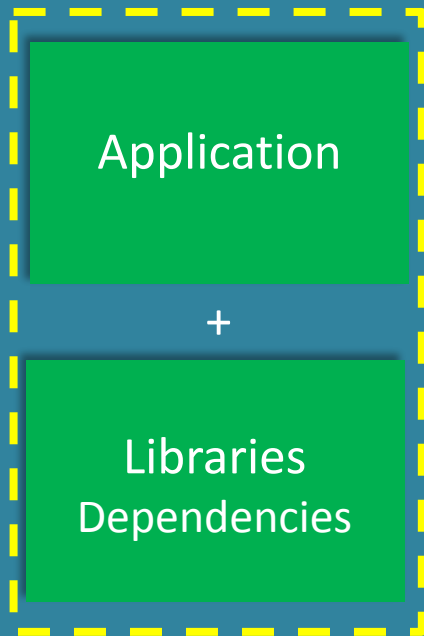


# Advantages of using containers for applications

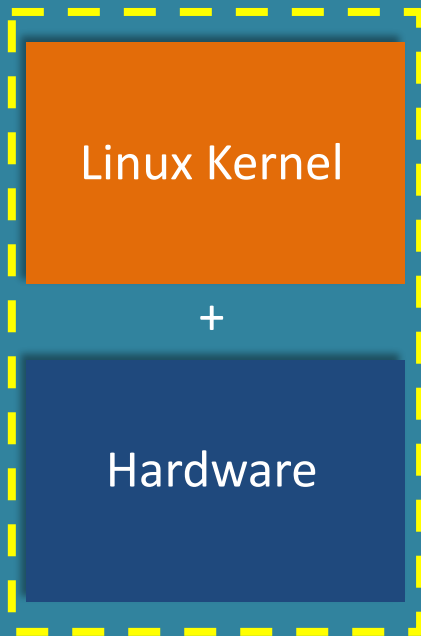
- Encapsulation
  - Applications, dependencies, configurations everything packed together
  - Portability across Linux systems
  - Makes easier the distribution and sharing of ready to use software
- Reproducibility
  - The whole application and run-time environment is in the container
  - Can be easily stored for later replay, reuse and preservation
- Efficiency
  - One single kernel shared by many applications
  - Performance and resource consumption similar to host execution
  - Take advantage of newer more optimized libraries and compilers
- Maintainability
  - Easier application maintenance, distribution and deployment

# Containers

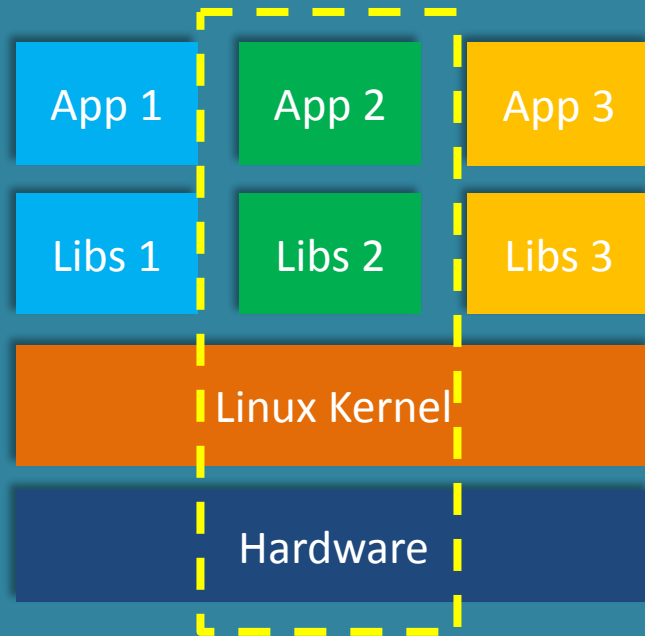
Container



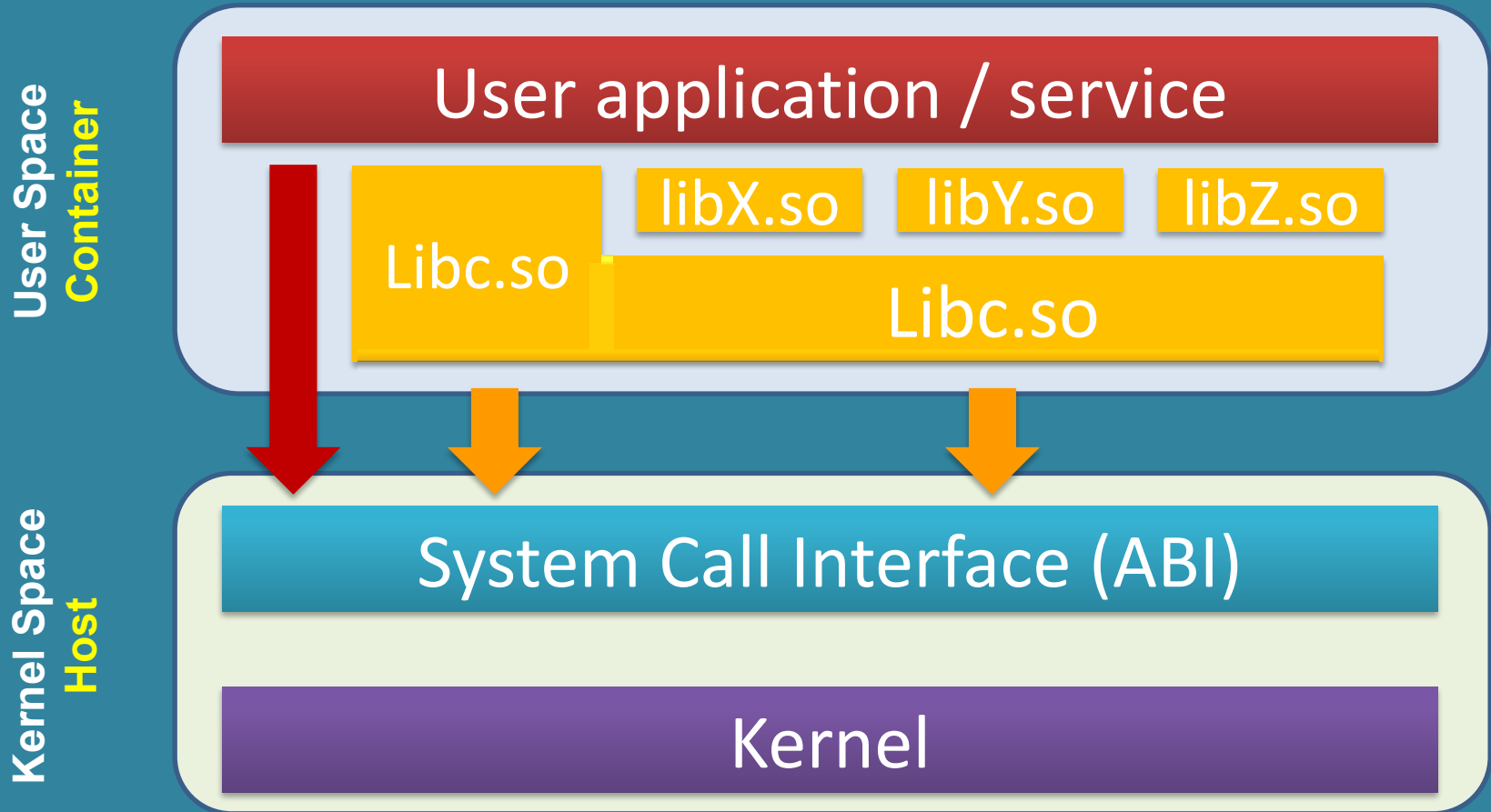
Host



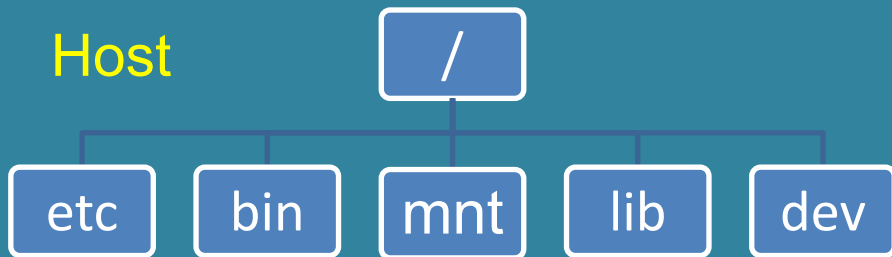
Separate Environment



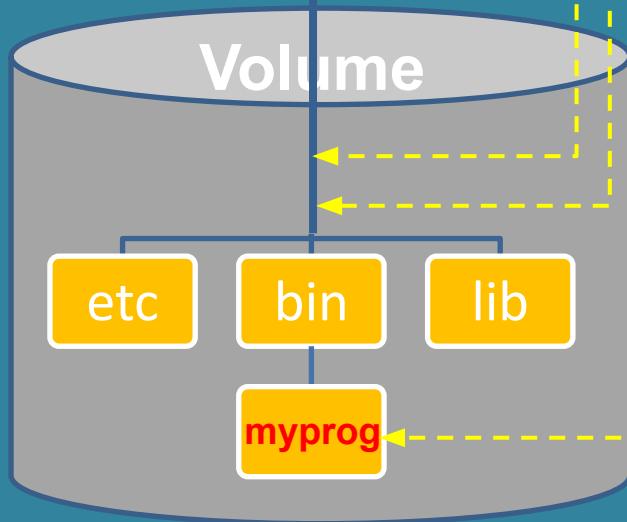
# Linux system call interface



chroot Host



Container



Process

```
mount( "VOL" , "/mnt"  
... )  
chdir( "/" )  
pivot_root( ".", "." )  
chroot( "." )  
execl( "/bin/myprog", ... )
```

- Using **mount** usually requires privileges (CAP\_SYS\_MOUNT)
  - Can use FUSE e.g. libguestfs
- Using **chroot** and **pivot\_root** usually requires privileges (CAP\_SYS\_CHROOT)
  - Can use user namespace



UDOCKER

Concept

# Motivations

## Run applications encapsulated in docker containers:

- without using docker
- without using privileges
- without system administrators intervention
- without compilation or additional system software

## and run containers:

- as a normal end-user without requiring privileges
- in Linux systems regardless of OS functionalities
- respecting normal process controls and accounting
- in Linux interactive or batch systems

Empowers end-users to run applications in containers



# udocker in 4 steps

## Installation:

- Just get the udocker python code
- No need to install or compile additional software
- No need of system administrator intervention

## Get container images:

- **Pull** containers from docker compatible repositories
- **Load** and save docker and OCI formats
- **Import** and export tarballs

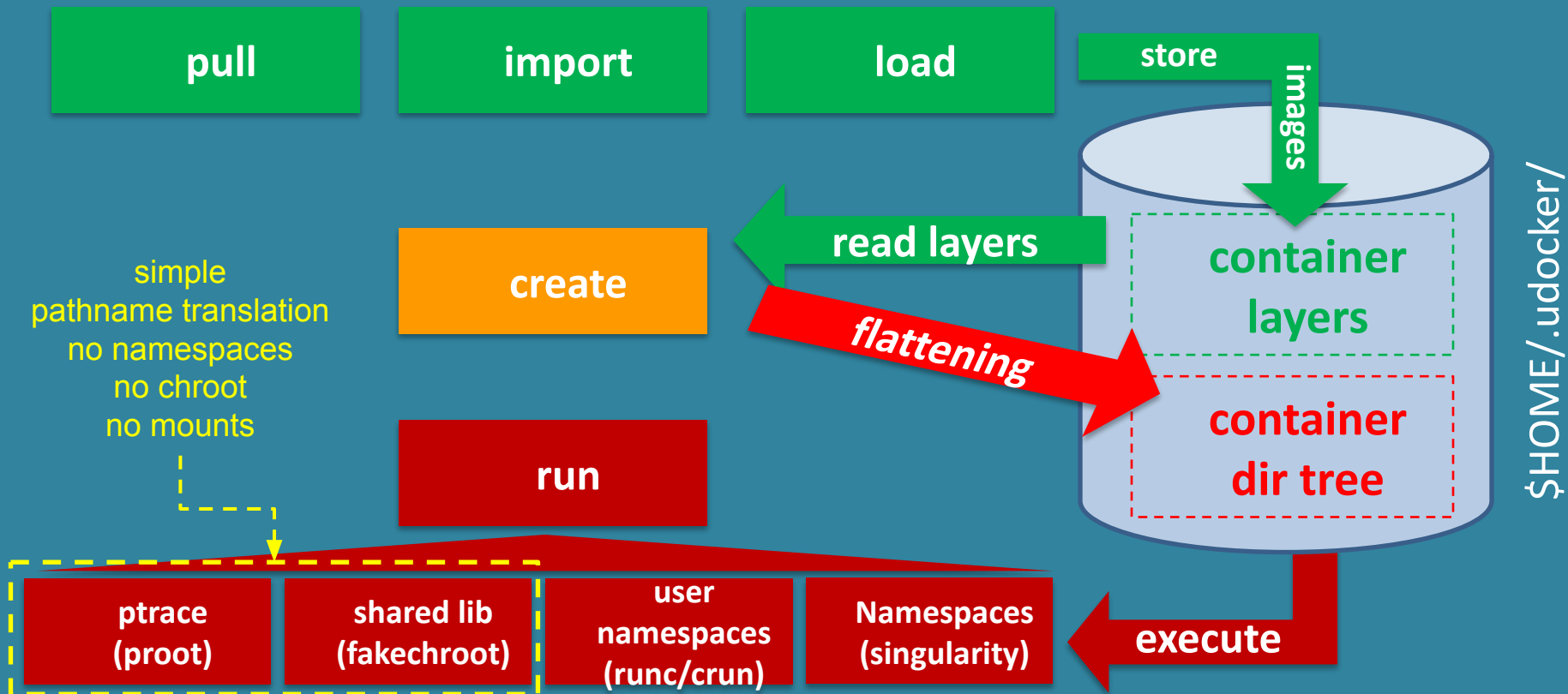
## Extract from images:

- **Create** the container directory tree from the image

## Execute containers:

- **Run** using several execution methods

# udocker is an integration tool



# Implementation

Python code plus several tools and libraries

- Python for portability and easier execution across systems
- External binary tools and libraries to enable execution

- Python code:
  - Command line interface similar to docker
  - Handling of containers (pull, load, import etc)
  - Local repository of images and containers
  - Interface to the execution engines (tools and libraries)
- External tools & libraries modified for udocker:
  - Execution of containers using several engines



# UDOCKER

## Installation

## Documentation

The full documentation is available at:

- [udocker documentation](#)
  - [Installation manual](#)
  - [User manual](#)
  - [Reference card](#)

Many recent  
improvements in the  
documentation

## 2. Installation

### 2.1. Install from a released version

Download a release tarball from <https://github.com/indigo-dc/udocker/releases>:

```
wget https://github.com/indigo-dc/udocker/releases/download/
tar zxvf udocker-1.3.0.tar.gz
export PATH=`pwd`/udocker:$PATH
```

Alternatively use `curl` instead of `wget` as follows:

```
curl -L https://github.com/indigo-dc/udocker/releases/downl
> udocker-1.3.0.tar.gz
tar zxvf udocker-1.3.0.tar.gz
export PATH=`pwd`/udocker:$PATH
```

udocker executes containers using external tools and libraries that are enhanced and packaged for use with udocker. For more information see [section 6 External tools and libraries](#). Therefore to complete the installation invoke `udocker install` to download and install the required tools and libraries.

# <https://github.com/indigo-dc/udocker>

## Latest source for Python 3 and Python 2:

- <https://github.com/indigo-dc/udocker/tree/master>
- <https://github.com/indigo-dc/udocker/tree/devel3>

## Latest old source for Python 2 only:

- <https://github.com/indigo-dc/udocker/tree/devel>

## Releases

- **1.3.0** ☐ latest for Python 3 also works on Python 2
- **1.1.7** ☐ old code for Python 2 only

# udocker since v1.3.0

v1.3.0 and above support both Python 3 and Python 2

- No longer a release prototype it has become stable
- Future new features will only be available in the Python 3

- Differences

- All new features ported or back-ported between the original Python 2 and the new Python 3, they are now equivalent
- Differences mostly at internal level where it was redesigned
- No longer a single large Python script
- Now modular to ease maintenance and new developments
- Command line interface retains the same syntax
- If there are things that don't work please add an issue

# Install from a given release

v1.1.8

<https://github.com/indigo-dc/udocker/releases>

## udocker 1.3.0



mariojmdavid released this 11 days ago

udocker v1.3.0 see the changelog and the documentation for further information.

- Changelog: <https://github.com/indigo-dc/udocker/blob/devel3/CHANGELOG.md>
- Documentation: <https://indigo-dc.github.io/udocker/>
- udocker release for Python 2.6, 2.7 and  $\geq 3.6$

Follow this steps to install and run udocker:

```
wget https://github.com/indigo-dc/udocker/releases/download/v1.3.0/udocker-1.3.0.tar.gz
tar zxvf udocker-1.3.0.tar.gz
export PATH=`pwd`/udocker:$PATH
```



# Install from a release

```
$ curl -L \  
https://github.com/indigo-dc/udocker/releases/download/v1.3.0/udocker-1.3.0.tar.gz \  
> udocker-1.3.0.tar.gz
```

**## untar the Python code. It is extracted to a directory called udocker**

```
$ tar zxvf udocker-1.3.0.tar.gz
```

**## optionally add the just created udocker directory to the PATH**

```
$ export PATH=`pwd`/udocker:$PATH
```

**## install the binaries required to execute containers under \$HOME/.udocker**

```
$ udocker install
```

```
$ udocker version
```

# Install from the source

```
$ git clone https://github.com/indigo-dc/udocker.git
```

```
$ cd udocker/udocker
```

```
## create a logical link
```

```
$ ln -s maincmd.py udocker
```

```
## optionally add the just created udocker directory to the PATH
```

```
$ export PATH=`pwd`: $PATH
```

```
## install the binaries required to execute containers under $HOME/.udocker
```

```
$ udocker install
```

```
$ udocker version
```

# Install from PyPI

**## Create Python 3 virtual env**

**\$ python3 -m venv udockervenv**

**## activate the virtual env**

**\$ source udockervenv/bin/activate**

**\$ ## install udocker from PyPI**

**\$ pip install udocker**

**## install the binaries required to execute containers**

**\$ udocker install**

**\$ udocker version**

# Install without outbound connectivity

```
$ wget \
  https://github.com/indigo-dc/udocker/releases/download/v1.3.0/udocker-1.3.0.tar.gz

## Get the additional tools (executables, libraries, etc)

$ wget \
  https://github.com/jorge-lip/udocker-builds/raw/master/tarballs/udocker-englib-1.2.8.tar.gz

## TRANSFER BOTH TARBALLS TO THE REMOTE SYSTEM and once transferred do:

$ tar zxvf udocker-1.3.0.tar.gz
$ export PATH=`pwd`/udocker:$PATH

## then install the binaries FROM THE TARBALL

$ export UDOCKER_TARBALL="udocker-englib-1.2.8.tar.gz"
$ udocker install
```



# UDOCKER

## Basic Usage

# udocker commands

```
$ udocker help
```

```
$ udocker pull --help
```

<b>search</b>	<b>pull</b>	<b>create</b>	<b>run</b>	<b>images</b>
<b>rm</b>	<b>rmi</b>	<b>rename</b>	<b>rmname</b>	<b>clone</b>
<b>import</b>	<b>export</b>	<b>load</b>	<b>save</b>	<b>inspect</b>
<b>verify</b>	<b>mkrepo</b>	<b>protect</b>	<b>unprotect</b>	<b>setup</b>
<b>login</b>	<b>logout</b>	<b>help</b>		

- udocker is mainly a run-time to execute containers
- Provides a subset of docker commands
- Actual container creation is better performed using docker itself

# Pull images from dockerhub

```
$ udocker pull ubuntu:18.04
```

```
Info: downloading layer sha256:726b8a513d66e3585eb57389171d97fcd348e4914a415891e1da135b85ffa6c3
```

```
Info: downloading layer sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
```

```
$ udocker pull quay.io:centos/centos:7
```

```
Info: downloading layer sha256:2d473b07cdd5f0912cd6f1a703352c82b512407db6b05b43f2553732b55df3bc
```

```
Info: downloading layer sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
```

```
Info: downloading layer sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4
```

- By default udocker pulls images from dockerhub

# Create runnable container from an image

container-name



```
$ udocker create --name=ub18 ubuntu:18.04
```

```
4c821126-aa28-3731-8d44-eae2f33c6477
```



container-id

- Create will extract the content of an image into the user home directory
- By default created containers are stored under \$HOME/.udocker/containers
- Created containers can be referenced either by id or name



# List created containers

```
$ udocker ps
```

CONTAINER ID	P M NAMES	IMAGE
1a0915b2-a8e1-395a-98da-f8dd61530f41	. W ['UB18P2']	ubuntu:18.10
6432f728-8577-3512-a109-0e953f05cd54	. W ['f34']	fedora:34
4c821126-aa28-3731-8d44-eae2f33c6477	. W ['ub18']	ubuntu:18.04
C40ce9b6-5902-3454-a9f0-21534b2c2a9c	. W ['UB18CC']	ubuntu:18.04
B61a6092-aff3-3579-a12c-1e68f5bfa953	. W ['C7C']	centos:centos7

- List created containers stored under `$HOME/.udocker/containers`
- Includes: id, protection, mode, names and related image
- Although the command is named **ps** there are no associated processes

# Execute a container

```
$ udocker run ub18
```

```
or
```

```
$ udocker run 4c821126-aa28-3731-8d44-eae2f33c6477
```

```
*****  
*                                                                 *  
*           STARTING 4c821126-aa28-3731-8d44-eae2f33c6477      *  
*                                                                 *  
*****
```

```
executing: bash  
root@host:~#
```

If the container has a default cmd to run  
it will be run otherwise starts a shell

- List created containers stored under \$HOME/.udocker/containers
- Includes: id, protection, mode, names and related image
- Although the command is named **ps** there are no associated processes

# Execute a container

\$ **udocker run** **ub18**

**ubuntu**

```
*****  
*                                                                 *  
*           STARTING 4c821126-aa28-3731-8d44-eae2f33c6477       *  
*                                                                 *  
*****
```

executing: bash

root@host:~#

root@host:/# **cat /etc/lsb-release**

**DISTRIB\_ID=Ubuntu**

**DISTRIB\_RELEASE=18.04**

**DISTRIB\_CODENAME=bionic**

**DISTRIB\_DESCRIPTION="Ubuntu 18.04.5 LTS"**

root@host:/# **id**

**uid=0(root) gid=0(root) groups=0(root),1000(G1000)**

**root emulation**

# Run as yourself

```
$ udocker run --user=jorge -v /home/jorge \  
-e HOME=/jorge/home --workdir=/home/jorge ub18
```

```
*****  
*                                                                 *  
*           STARTING 4c821126-aa28-3731-8d44-eae2f33c6477       *  
*                                                                 *  
*****  
executing: bash  
jorge@host:~$ id  
uid=1000(jorge) gid=1000(G1000) groups=1000(G1000)  
jorge@host:~$ pwd  
/home/jorge
```

- --user identifies a username
- -v binds a directory to be visible inside of the running container
- -e allows setting environment variables

# Run as yourself

```
$ udocker run --user=$USER --bindhome --hostauth ub18
```

```
*****  
*                                                                 *  
*          STARTING 4c821126-aa28-3731-8d44-eae2f33c6477          *  
*                                                                 *  
*****
```

```
executing: bash
```

```
jorge@host:~$ id
```

```
uid=1000(jorge) gid=1000(G1000) groups=1000(G1000)
```

```
jorge@host:~$ pwd
```

```
/home/jorge
```

- --user identifies a username
- --bindhome binds the user home directory to be visible in the container
- --hostauth uses the host passwd and group in the container

# Less verbosity

```
$ udocker -q run ub18 /bin/cat /etc/lsb-release
```

```
DISTRIB_ID=Ubuntu
```

```
DISTRIB_RELEASE=18.04
```

```
DISTRIB_CODENAME=bionic
```

```
DISTRIB_DESCRIPTION="Ubuntu 18.04.5 LTS"
```

```
$ alias u="udocker -q run --user=$USER --bindhome --hostauth ub18"
```

```
$ u /bin/ls
```

Will list the content of the user home directory in the host

- -q quiet mode

# Overriding the entrypoint

```
$ udocker -q run --user=$USER --bindhome --hostauth ub18 /bin/bash -c "id; pwd"
```

```
$ udocker -q run --user=$USER --bindhome --hostauth --entrypoint="/bin/bash" ub18 -c "id; pwd"
```

```
$ udocker -q run --user=$USER --bindhome --hostauth --entrypoint="" /bin/bash ub18 -c "id; pwd"
```

```
uid=1000(jorge) gid=1000(jorge) groups=1000(jorge)  
/home/jorge
```

- `--entrypoint` will override the entrypoint defined in the container metadata
- `--entrypoint` is valid from v1.3.0 and is the preferred method
- A container defined entrypoint can be ignored with `--entrypoint=""`

# Run commands inline

```
$ udocker -q run --user=$USER --bindhome --hostauth --entrypoint="" ub18 /bin/bash <<EOD
```

```
id
```

```
pwd
```

```
EOD
```

```
uid=1000(jorge) gid=1000(jorge) groups=1000(jorge)
```

```
/home/jorge
```

- Useful to execute commands inside a container in scripts



# Duplicate a container

```
$ udocker clone --name=new18 ub18
```

9fe2f9e7-ce37-3be5-b12d-829a3236d2a6

new cloned container-id

new cloned name

```
$ udocker run new18
```

or

```
$ udocker run 9fe2f9e7-ce37-3be5-b12d-829a3236d2a6
```

- The command clone duplicates an existing container

# Import and export tarballs with images

**export to tarball**      **input container**

\$ **udocker export -o ub18.tar ub18** ←

\$ **udocker import ub18.tar myub18:latest** ← **import ub18.tar tarball and create image with this name and tag**

\$ **udocker export -o - ub18 | docker import - myub18:latest**

\$ **docker export bd221eb5e452 | **udocker import** - anotherub18:latest**

- Export produces a tar file from a created container with the container filesystem
- Export does not include the container metadata
- Import takes a tar file and loads its content as an image
- Import and export are interoperable with docker

# Saving space

```
$ udocker import --tocontainer --name=xx ub18.tar
```

- `--tocontainer` allows importing a tar file directly to a container (NOT TO IMAGE !!)
- No image is created
- The newly created container can be named with `--name`
- This will save both space and the intermediate step of creating an image

```
$ udocker import --mv ub18.tar myub18:latest
```

- Import a tar file normally to a new image
- To save space the tar file is moved to the udocker repository instead of copied
- The original tar file disappears

# Import and export including udocker specific metadata

```
$ udocker export --clone -o ub18.tar ub18
```

```
$ udocker import --clone --name=xx ub18.tar
```

- Export a container to a tar file including the udocker specific metadata
- Both container data and metadata are included
- Import a tar file produced by *export --clone* as a new CONTAINER
- The newly created container can be named with *--name*
- NOT interoperable with docker as includes udocker specific information

# Transfer containers across machines

export the tarball to stdout

```
$ udocker export --clone ub18 | ssh user@host \  
  "udocker import --clone --name=xx -"
```

read the tarball from stdin

- Export a container and import it in another remote host or account using SSH
- Allows export and import to be accomplished in one go
- Not interoperable with docker
- Notice the “” around the remote command when using SSH
- Containers with execution modes such as F2, F3 and F4 modes may not work, breaks if the pathname to the user home (container dir) is not the same

# Save and load images

```
$ docker save -o image.tar centos:centos7
```

or

```
$ udocker save -o image.tar centos:centos7
```

```
$ udocker load -i image.tar
```

- Load an image saved by docker or udocker
- A saved image will contain multiple layers and metadata (different for export !!)
- A saved image cannot be imported only loaded
- Can be used to load docker images instead of pulling them from dockerhub
- udocker also provides a compatible save functionality

# Remove containers and images

`$ udocker rm -f ub18` ← delete container by alias or container-id

`$ udocker rm -f 4c821126-aa28-3731-8d44-eae2f33c6477`

← -f force is optional

`$ udocker rmi -f ubuntu:18.04` ← delete image

- Deleting images does not affect the created containers



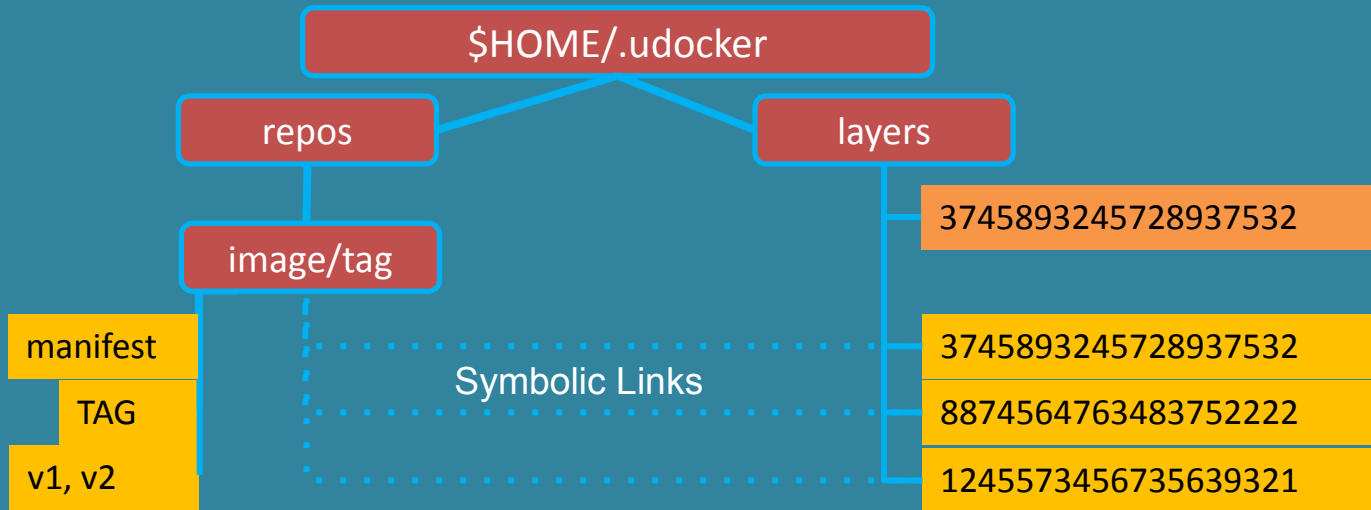
# UDOCKER

## Advanced Topics



# udocker pull

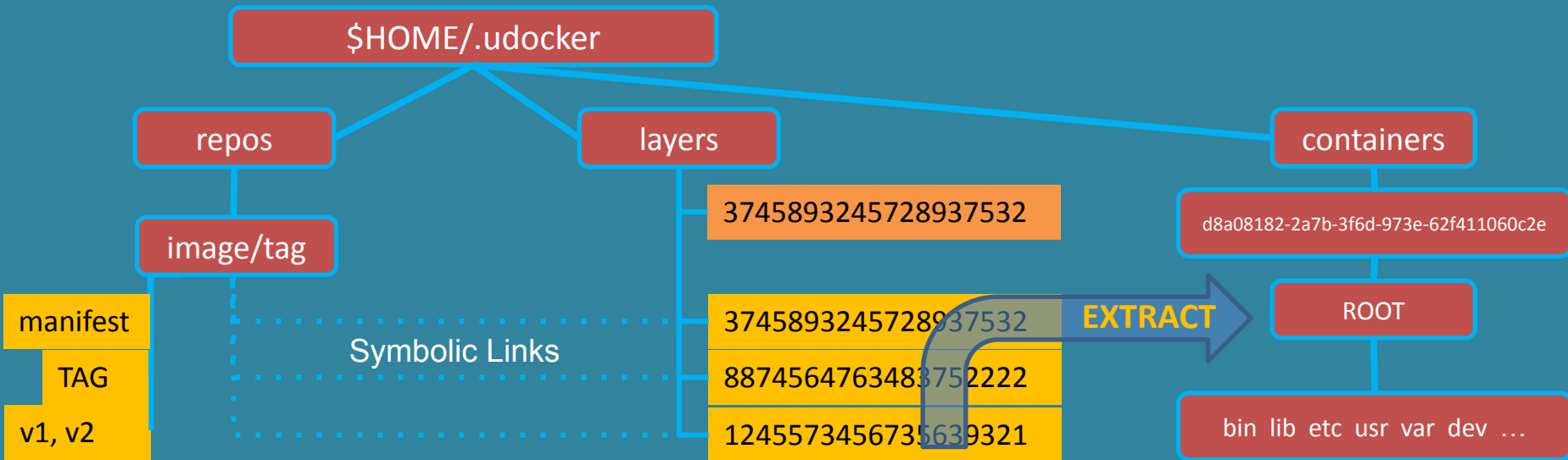
- Images
  - Layers and metadata are pulled using the DockerHub REST API
  - Image metadata is parsed by udocker to identify the image layers
  - Layers are stored in the use home directory under \$HOME/.udocker/layers
  - Layers can be shared by multiple images



# udocker create

- Containers

- Are produced from the layers by flattening them sequentially
- Each layer is extracted on top of the previous
- The OnionFS whiteouts are respected, and file protections are changed as needed
- The obtained directory trees are stored under \$HOME/.udocker/containers

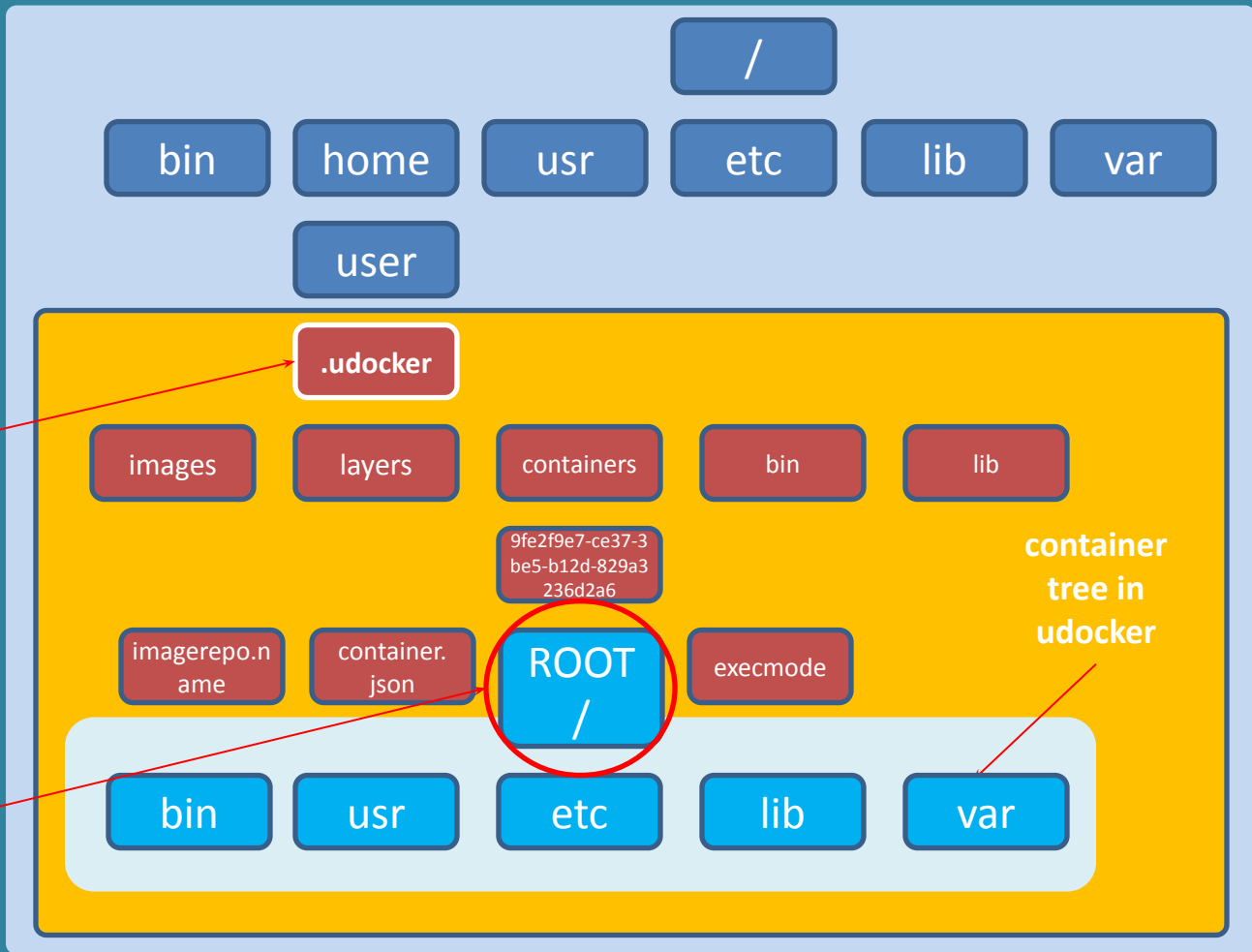


# udocker run

- Execution
- chroot-like

udocker  
directory tree  
\$HOME/.udocker

“chroot” to this directory  
and it becomes the new root  
for the container processes



# Where is my container

container alias or container-id



```
$ udocker inspect -p ub18
```

```
/home/jorge/.udocker/containers/f80f88de-3227-3cba-8551-cd62ddb14174/ROOT
```

```
$ ls $(udocker inspect -p ub18)
```

```
bin dev home lib64 mnt proc run srv tmp var boot etc lib media opt root sbin sys usr
```

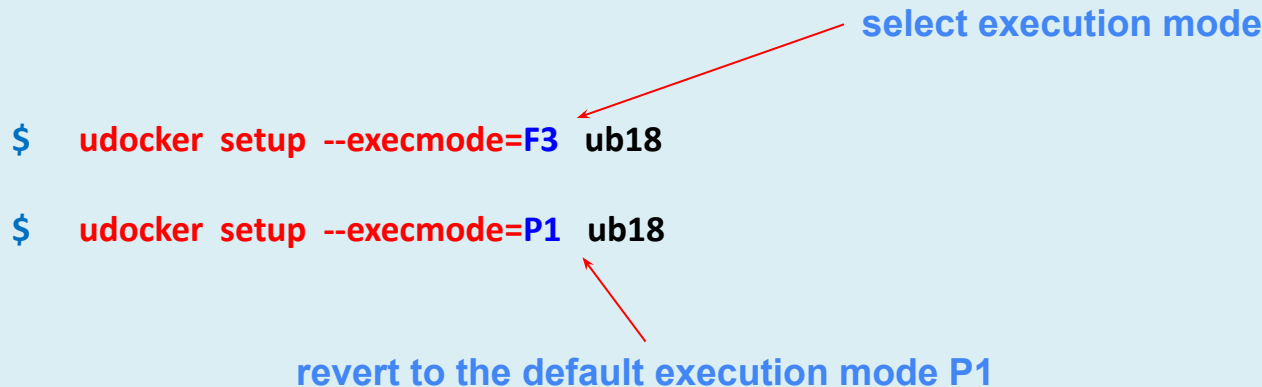
- Inspect -p prints the pathname to the container ROOT directory
- Knowing the pathname is useful to copy/manage/edit the container files directly

# Execution engines

- udocker integrates several execution methods:
  - Supports several engines to execute containers
  - They are selected per container via execution modes

Mode	Base	Description
<b>P1</b>	PRoot	PTRACE accelerated (with SECCOMP filtering) <input type="checkbox"/> <b>DEFAULT</b>
<b>P2</b>	PRoot	PTRACE non-accelerated (without SECCOMP filtering)
<b>R1</b>	runC / Crun	rootless unprivileged using user namespaces
<b>R2</b>	runC / Crun	rootless unprivileged using user namespaces + P1
<b>R3</b>	runC / Crun	rootless unprivileged using user namespaces + P2
<b>F1</b>	Fakechroot	with loader as argument and LD_LIBRARY_PATH
<b>F2</b>	Fakechroot	with modified loader, loader as argument and LD_LIBRARY_PATH
<b>F3</b>	Fakechroot	modified loader and ELF headers of binaries + libs changed
<b>F4</b>	Fakechroot	modified loader and ELF headers dynamically changed
<b>S1</b>	Singularity	where locally installed using chroot or user namespaces

# Change the execution engine

  
\$ **udocker setup --execmode=F3** ub18  
\$ **udocker setup --execmode=P1** ub18  
revert to the default execution mode P1

- Execution modes are selected using setup
- Each mode has two characters
  - the first is a letter that identifies the engine
  - the second is a number that identifies different options within an engine
- The default execution mode is P1

# List containers and execution engines

```
$ udocker ps -m
```

CONTAINER ID	P	M	MOD	NAMES	IMAGE
1a0915b2-a8e1-395a-98da-f8dd61530f41	.	W	F2	['UB18P2']	ubuntu:18.10
6432f728-8577-3512-a109-0e953f05cd54	.	W	P1	['f34']	fedora:34
4c821126-aa28-3731-8d44-eae2f33c6477	.	W	R1	['ub18']	ubuntu:18.04
c40ce9b6-5902-3454-a9f0-21534b2c2a9c	.	W	P2	['UB18CC']	ubuntu:18.04
b32db50b-ecfd-3801-bd40-a7ba64b6823b	.	W	P1	['BUSY']	busybox:latest
b61a6092-aff3-3579-a12c-1e68f5bfa953	.	W	F4	['C7C']	centos:centos7

- `udocker -m` will list all containers and their execution modes

# Engine Pn : PRoot

- PRoot uses PTRACE to intercept system calls
- Pathnames are translated before being passed to the call
  - To expand container pathnames into host pathnames
- After the call returned pathnames are translated back
  - To shrink host pathnames to container pathnames
- P1 mode uses PTRACE + SECCOMP filtering, to minimize the interception to the set of calls that manipulate pathnames
  - We developed code to make it work on recent kernels
  - P1 is the udocker default mode
- P2 uses PTRACE without SECCOMP □ traces all system calls □ slower
- The impact of tracing depends on the system call frequency



# Engine Fn : Fakechroot

- Uses LD\_PRELOAD to intercept shared library calls
- Pathnames are translated before being passed to the call
  - To expand container pathnames into host pathnames
- After the call returned pathnames are translated back
  - To shrink host pathnames to container pathnames
- Generally higher performance than Pn and even than namespace based engines
- Uses heavily modified Fakechroot for both glibc and musl libc
- Requires changes to files in the container
- Can run inside namespaces to provide nested containers
- There are 4x Fn modes (F1, F2, F3 and F4)
- Does not support root emulation

-

# Engine Fn : all modes

- F1
  - Forces 1<sup>st</sup> exec() argument to be the container ld.so
  - Populates the LD\_LIBRARY\_PATH with container libs
- F2
  - Same as F1
  - Loading from default host paths /lib, /lib64 etc is disabled
  - Loading from ld.so.cache is disabled
- F3
  - Modifies the ELF headers of executables and libraries
    - to use the ld.so of the container
    - to direct the loading of shared libraries to the container
- F4
  - Same as F3 but changes ELF headers dynamically

# Containers using kernel features

- **chroot, pivot\_root**: make a given directory root of the file system
- **Kernel namespaces**: isolate system resources from process
  - **Mount**: isolate mount points (cannot see host or other containers mounts)
  - **UTS**: virtualize hostname and domain
  - **IPC**: inter process communications isolation (semaphores, shm, msg)
  - **PID**: isolate and remap process identifiers (cannot see other processes)
  - **Network**: isolate network resources (interfaces, tables, firewall etc)
  - **cgroup**: isolate cgroup directories
  - **Time**: virtualize boot and monotonic clocks
  - **User**: isolate and remap user/group identifiers (user can be a limited root)
- **cgroups**: process grouping and resource consumption limits
- **seccomp**: system call filtering
- **POSIX capabilities**: split and drop root privileges
- **AppArmor and SELinux**: kernel access control

# Linux user namespace

Available on fairly “recent” kernels/distributions

- Allows an unprivileged user to have a different UID/GID
  - Enables an unprivileged user to become UID/GID 0 root
  - Enables executing `pivot_root`, `chroot` and other calls
- May require some setup of `subuid` and `subgid` files
  - Network namespace becomes useless as only has a loopback device
  - root has limitations
    - Cannot create devices (`mknod`)
    - Cannot load kernel modules
    - Mount is restricted to some file system types
    - Issues on changing and handling user ids group ids
    - Accessing files in the host (`mount bind`) can become problematic
  - Not enabled in some distributions (RedHat/CentOS)

# Engine Rn : runc & crun

- runc & crun are tools to spawn containers according to the Open Containers Initiative (OCI) specification
  - Support unprivileged namespaces using the user namespace
  - User namespace has several limitations but allows execution without privileges by normal users
  - Limited support for mapping of devices
- We added mapping of Docker metadata to OCI
- udocker can produce an OCI spec and run containers using runc or crun transparently

# NVIDIA GPUs

```
$ udocker setup --nvidia ub18
```

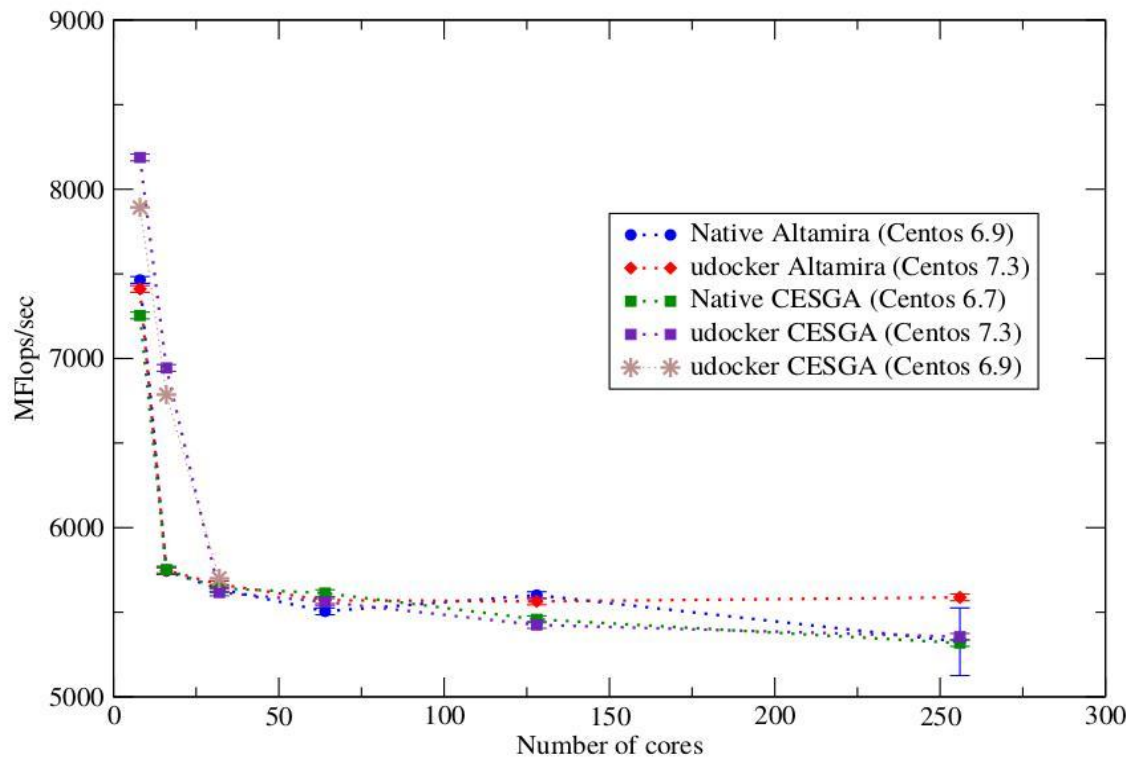
- Enables use of NVIDIA GPUs
- Similar to nvidia-docker
- Copies the nvidia libraries from the host to the container



**UDOCKER**

Benchmarks

# Lattice QCD



by Isabel Campos (IFCA/CSIC)

OpenQCD is a very advanced code to run lattice simulations

Scaling performance as a function of the cores for the computation of application of the Dirac operator to a spinor field.

Using OpenMPI

udocker in P1 mode



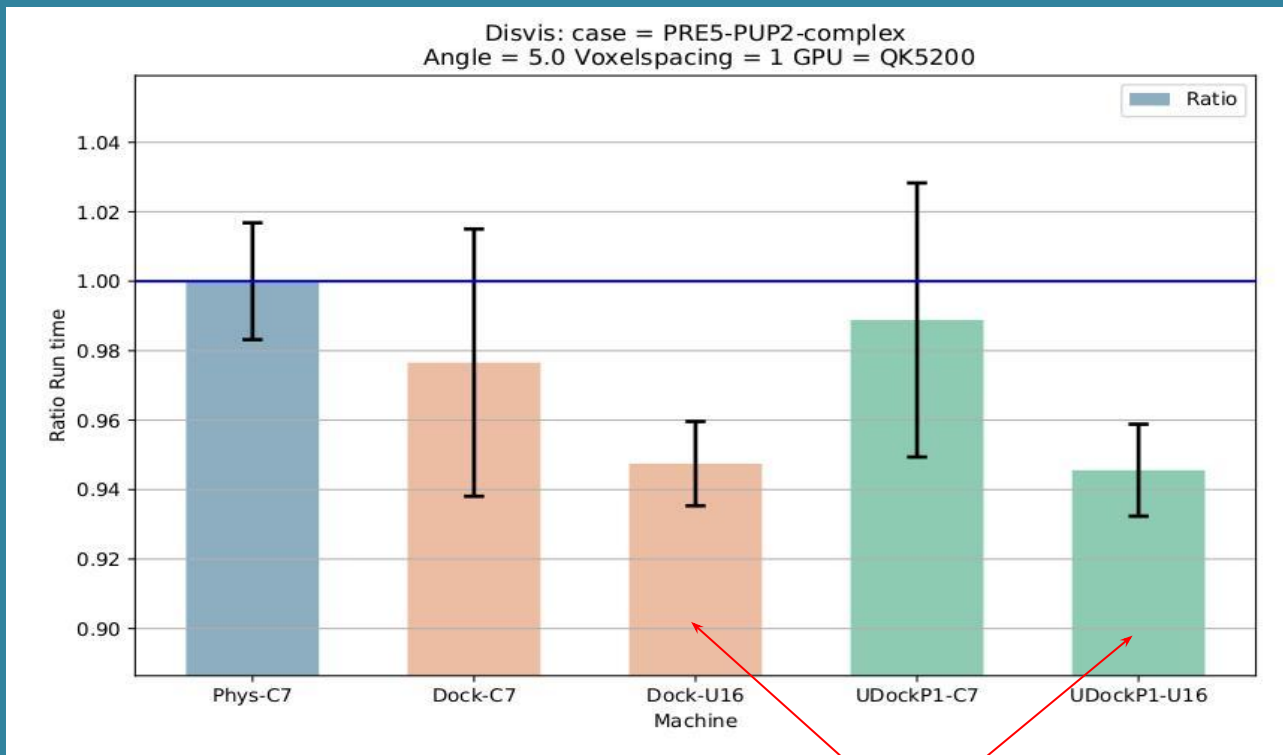
# Lattice QCD with OpenMPI

```
$ mpiexec -np 256 udocker run \  
    -e LD_LIBRARY_PATH=/usr/lib \  
    --hostenv \  
    --hostauth \  
    --user=$USER \  
    -v /tmp \  
    --workdir=/opt/projects/openQCD-1.6/main \  
    openqcd \  
    /opt/projects/openQCD-1.6/main/ym1 -i ym1.in -noloc
```

*by Isabel Campos (IFCA/CSIC)*

- mpiexec starts udocker which in turn runs the executable within the container
- LD\_LIBRARY\_PATH is redefine to point to the container /usr/lib
- The environment from the host has the OPENMPI variables that are essential
- Must run as the same user as in the host hence --hostauth and --user
- The workdir is within the container

# Biomolecular Complexes



**Better performance with Ubuntu 16 container**

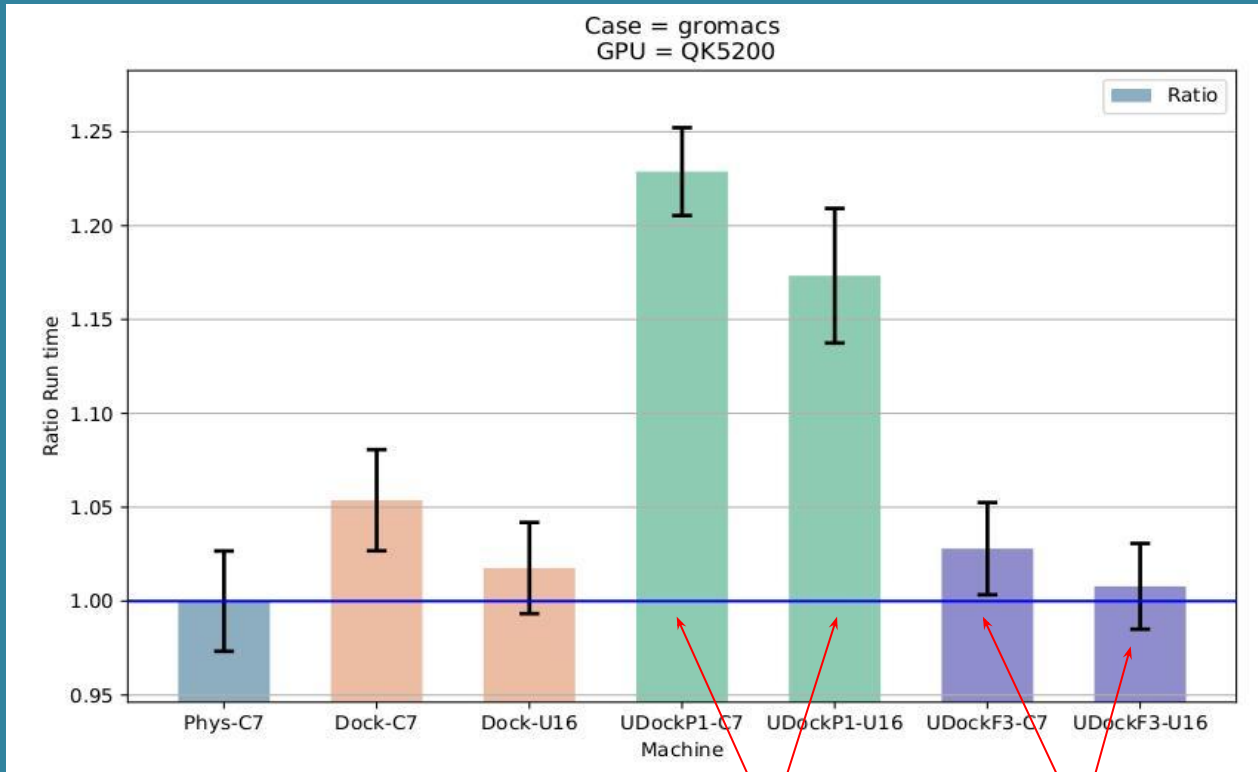
DisVis is being used in production with udocker

Performance with docker and udocker are the same and very similar to the host.

Using OpenCL and NVIDIA GPGPUs

udocker in P1 mode

# Molecular dynamics



**PTRACE**

**SHARED LIB CALL**

Gromacs is widely used both in biochemical and non-biochemical systems.

udocker P mode have lower performance  
udocker F mode same as Docker.

Using OpenCL and OpenMP

udocker in P1 mode  
udocker in F3 mode

# TensorFlow

## Container:

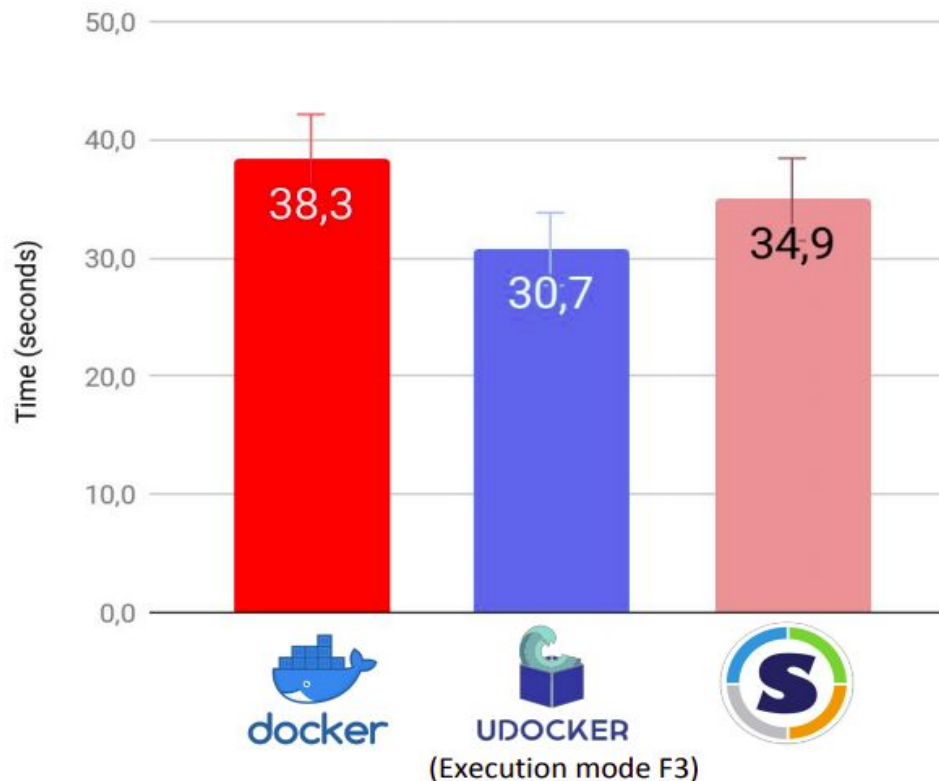
- Latest GPU version of Tensorflow (from Docker Hub).
- Train a model to recognize handwritten digits (the MNIST data set).

<https://github.com/tensorflow/models.git>



UP  
V

## EXECUTION TIME



# Thank you !

Questions ?

`udocker@lip.pt`

`https://github.com/indigo-dc/udocker`

UP  
V

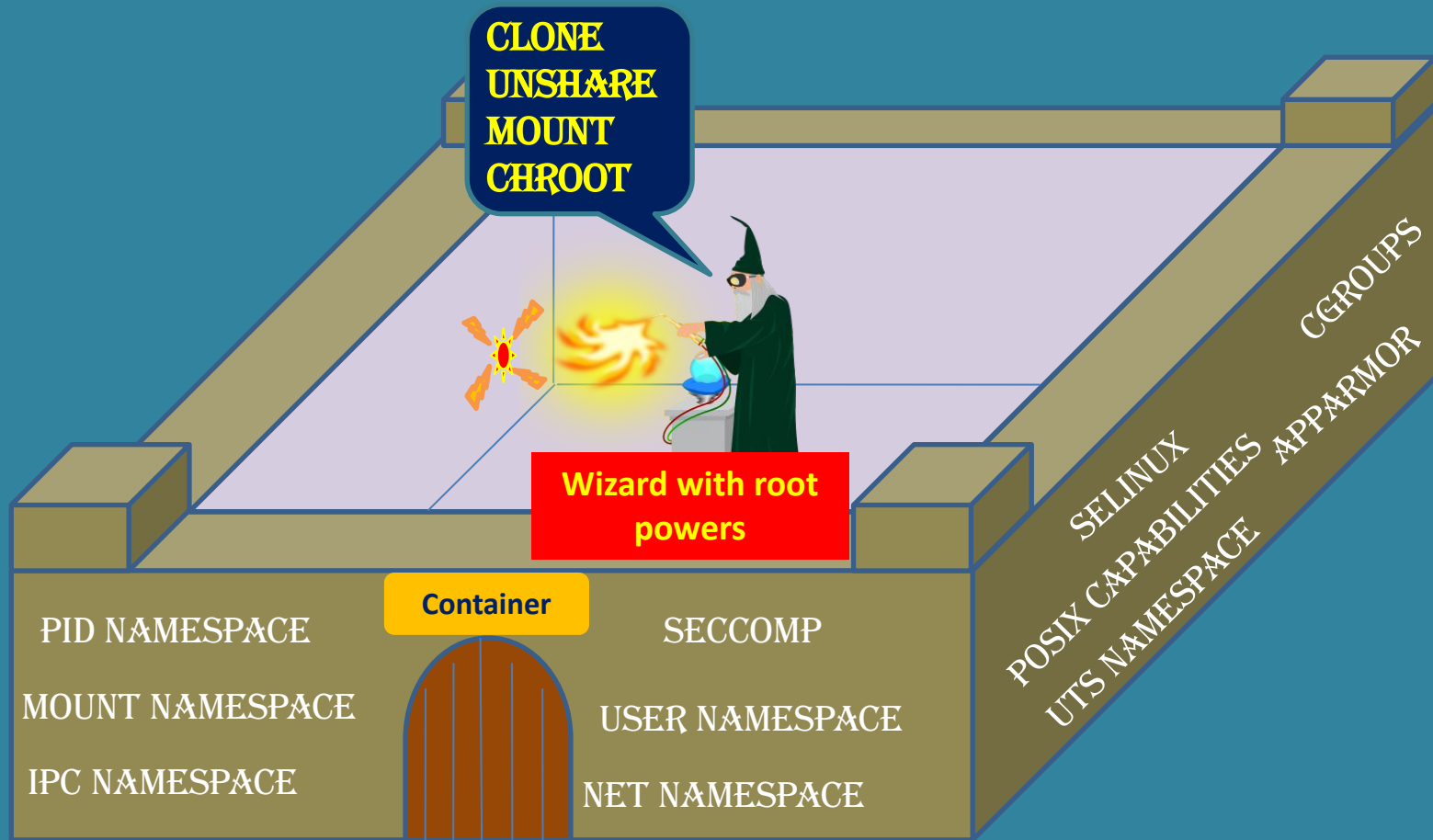
# Selection in terms of performance

Mode	Base	Description
P1	PRoot	Some multithreaded applications can suffer degradation
P2	PRoot	Same limitations as P1 apply All system calls are traced causing higher overheads than P1
R1	runC / Crun	Same performance as namespace based applications
R2	runC / Crun	Only for software installation and similar. Same performance as P1
R3	runC / Crun	Only for software installation and similar. Same performance as P2
F1	Fakechroot	All Fn modes have similar performance during execution Frequently the Fn modes are the fastest.
F2	Fakechroot	Same as F1
F3	Fakechroot	Same as F1. Setup can be very slow
F4	Fakechroot	Same as F1. Setup can be very slow
S1	Singularity	Similar to Rn

# Selection in terms of interoperability

Mode	Base	Description
P1	PRoot	PTRACE + SECCOMP requires kernel $\geq 3.5$ Can fall back to P2 if SECCOMP is unavailable
P2	PRoot	Runs across a wide range of kernels even old ones Can run with kernels and libraries that would fail with kernel too old
R1	runC / Crun	User namespace limitations apply
R2	runC / Crun	User namespace limitations apply Same limitations as P1 also apply, this is a nested mode P1 over R
R3	runC / Crun	User namespace limitations apply Same limitations as P2 also apply, this is a nested mode P2 over R
F1	Fakechroot	Requires shared library compiled against same libc as in the container. May load host libraries.
F2	Fakechroot	Same as F1
F3	Fakechroot	Requires shared library compiled against same libc as in container Binary executables and libraries get tied to the user HOME pathname
F4	Fakechroot	Same as F3. Executables and libraries can be compiled or added dynamically
S1	Singularity	Must be available on the system might use user namespaces or chroot

# Security when using kernel features





# Search repositories

```
$ udocker search centos
```

NAME	OFFICIAL DESCRIPTION	STARS
centos	[OK] The official build of CentOS.	6611
pivotaldata/centos-gpdb-dev	---- CentOS image for GPDB development. Tag names often have GCC because we	13
pivotaldata/centos-mingw	---- Using the mingw toolchain to cross-compile to Windows from CentOS	3
jdeathe/centos-ssh	---- OpenSSH / Supervisor / EPEL/IUS/SCL Repos - CentOS.	118
pivotaldata/centos	---- Base centos, freshened up a little with a Dockerfile action	5
ansible/centos7-ansible	---- Ansible on Centos7	134
consol/centos-xfce-vnc	---- Centos container with "headless" VNC session, Xfce4 UI and preinstalle	129
pivotaldata/centos-gcc-toolchain	---- CentOS with a toolchain, but unaffiliated with GPDB or any other parti	3
smartentry/centos	---- centos with smartentry	0
kinogmt/centos-ssh	---- CentOS with SSH	29
centos/systemd	---- systemd enabled base container.	99
imagine10255/centos6-lnmp-php56	---- centos6-lnmp-php56	58
blacklabelops/centos	---- CentOS Base Image! Built and Updates Daily!	1
drecom/centos-ruby	---- centos ruby	6
darksheer/centos	---- Base Centos Image -- Updated hourly	3

- Searches for repositories in dockerhub

# List tags in a repository

```
$ udocker search --list-tags centos
```

5.11	7.5.1804	centos6.10	centos7.6.1810
5	7.6.1810	centos6.6	centos7.7.1908
6.10	7.7.1908	centos6.7	centos7.8.2003
6.6	7.8.2003	centos6.8	centos7.9.2009
6.7	7.9.2009	centos6.9	centos7
6.8	7	centos6	centos8.1.1911
6.9	8.1.1911	centos7.0.1406	centos8.2.2004
6	8.2.2004	centos7.1.1503	centos8.3.2011
7.0.1406	8.3.2011	centos7.2.1511	centos8
7.1.1503	8	centos7.3.1611	latest
7.2.1511	centos5.11	centos7.4.1708	
7.3.1611	centos5	centos7.5.1804	
7.4.1708			

- This is specific to dockerhub and may not work with other repositories

# Pull from other registries

```
$ udocker search quay.io/centos
```

```
$ udocker pull quay.io/centos/centos:latest
```

Info: downloading layer sha256:7a0437f04f83f084b7ed68ad9c4a4947e12fc4e1b006b38129bac89114ec3621

Info: downloading layer sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4

Info: downloading layer sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4

- The first element in the image name is the registry name in this case quay.io
- The second element is the repository also called library
- The third element is the image name and tag separated by a semicolon