udocker - *be anywhere*

Part 1 - Introduction

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

Mario David david@lip.pt, Jorge Gomes jorge@lip.pt

Containers in Scientific Computing I

Running applications across infrastructures may require considerable effort

- **Computers**:
  - Several computing systems
  - Laptops, Desktops, Farms, Cloud, HPC
- **OSes**:
  - Several operating systems
  - Linux flavors, Distribution versions

Containers in Scientific Computing II

- **Environments**:
  - Specific computing environments
  - Compilers, Libraries, Customizations
- **Applications**:
  - Multiple applications often combined
  - Portability, Maintainability, Reproducibility

Why using containers for applications I

Encapsulation:

- Applications, dependencies, configurations everything packed together.
- Portability across heterogeneous Linux systems.
- Makes easier the distribution and sharing of ready to use software.

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.

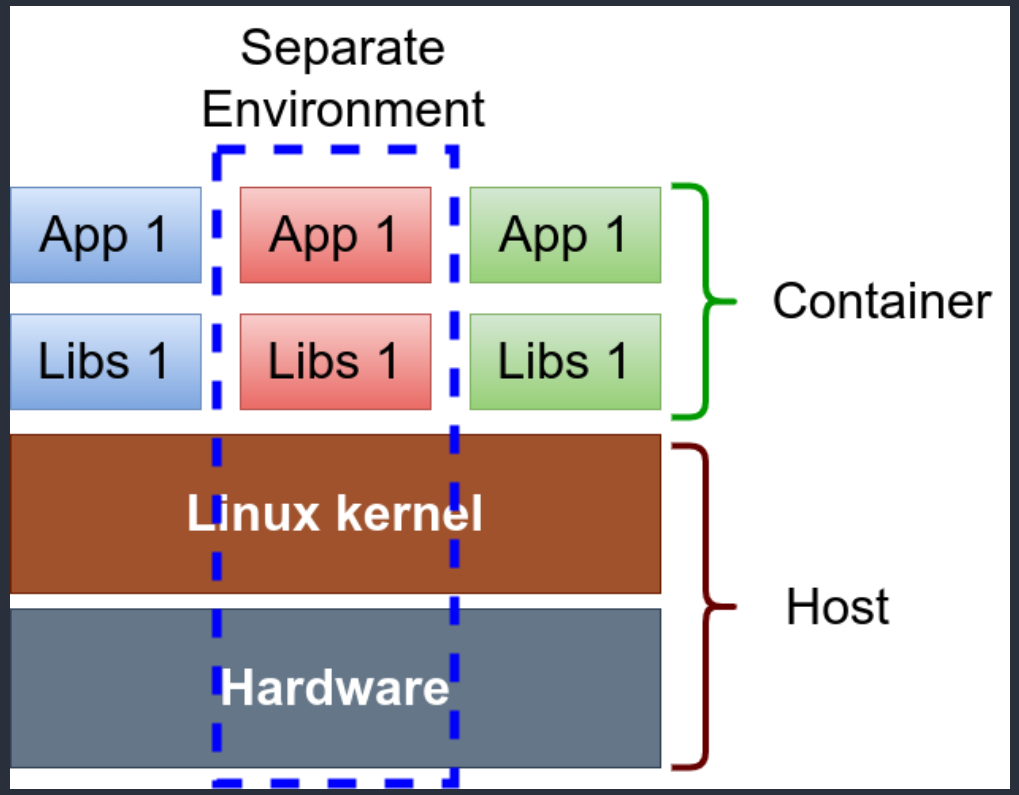
Why using containers for applications II

Reproducibility:

- The whole application and run-time environment is in the container.
- Can be easily stored for later replay, reuse and preservation.

Maintainability:

- Easier application maintenance, distribution and deployment.



udocker - beginnings

Need a consistent portable way of running applications.

udocker began to be developed in 2015 Indigo-DataCloud project.

Focused on running scientific applications in Linux clusters.

Containers for batch processing - I

- Challenges of batch systems?
  - Integrate it with the batch system (how to start/stop etc) ?
  - Respect batch system policies (such as quotas/limits) ?
  - Respect batch system actions (job delete/kill) ?
  - Collect accounting ?

Containers for batch processing - II

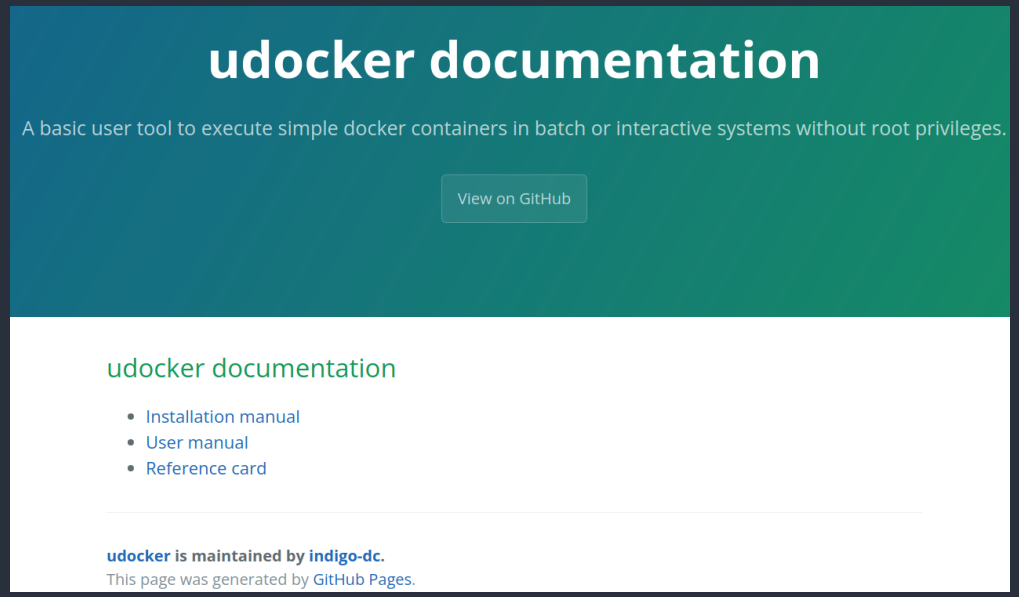
- Can we execute in a more basic way?
  - Can we download container images?
  - Can we run without a layered filesystem?
  - Can we run them as normal user?
  - Can we still enforce container metadata?

udocker: Introduction - I

- Run applications encapsulated in docker containers:
  - without using docker
  - without using (root) privileges
  - without system administrators intervention
  - without additional system software
  - does not require Linux namespaces
- Run:
  - as a normal user
  - with the normal process controls and accounting
  - in interactive or batch systems

udocker: Introduction - II

- udocker is open source.
- Developed under the Indigo-Datacloud, DEEP Hybrid-Datacloud, EOSC-Synergy and BigHPC projects.
- Github repository: https://github.com/indigo-dc/udocker.
- Documentation: https://indigo-dc.github.io/udocker/.



udocker advantages: deployment I

- udocker can be deployed and used by the end-user:
  - Does not require privileges.
  - Does not require system administrator intervention.
  - All operations performed in user space.

udocker advantages: deployment II

- udocker does not require compilation:
  - Uses Python plus some binaries.
  - Has a minimal dependencies.
  - Required executables are provided statically compiled.
- udocker deployment:
  - Just copy and untar into the user home directory.
  - Ideal to execute containers across different sites.

udocker advantages: execution I

- udocker integrates several execution engines:
  - Allows execution with several approaches/engines.
  - Allows execution with and without Linux namespaces.
- udocker can be submitted with the batch job:
  - Just fetch or ship the udocker tarball with the job.

udocker advantages: execution II

- udocker user interface:
  - Commands and logic similar to docker.
- udocker empowers users to use containers:
  - Ideal for heterogeneous computing environments.

udocker: CLI

Run time to execute docker containers:

clone	export	help	images	import
inspect	install	load	login	logout
mkrepo	name	protect	ps	pull
rm	rmi	rmname	search	setup
showconf	unprotect	verify	version	create
run	save			

udocker: How does it work...

Programing languages and OS

- Implemented
  - python, C, C++, go
- Can run:
  - CentOS 6, CentOS 7, RHEL8 (compatible distros)
  - Ubuntu >= 16.04
  - Any distro that supports python 2.6, 2.7 and >= 3.6

Features

- Components:
  - Command line interface docker like
  - Pull of containers from Docker Hub
  - Local repository of images and containers
  - Execution of containers with modular engines

udocker in 4 steps - I

1 - Installation:

- Get the udocker tarball and untar.
- No need to compile software.

2 - Get container images:

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

udocker in 4 steps - II

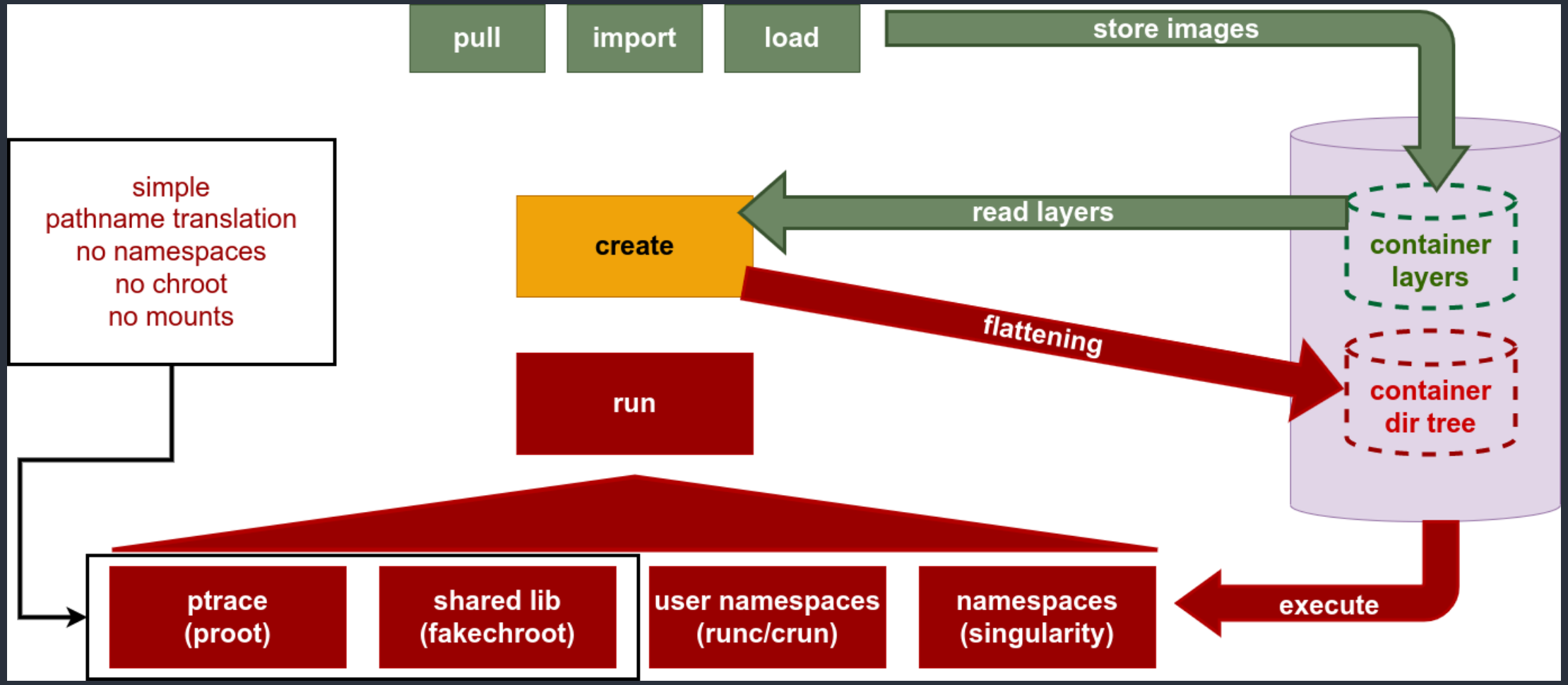
3 - Create from images:

- Create the container directory tree from the image.

4 - Execute containers:

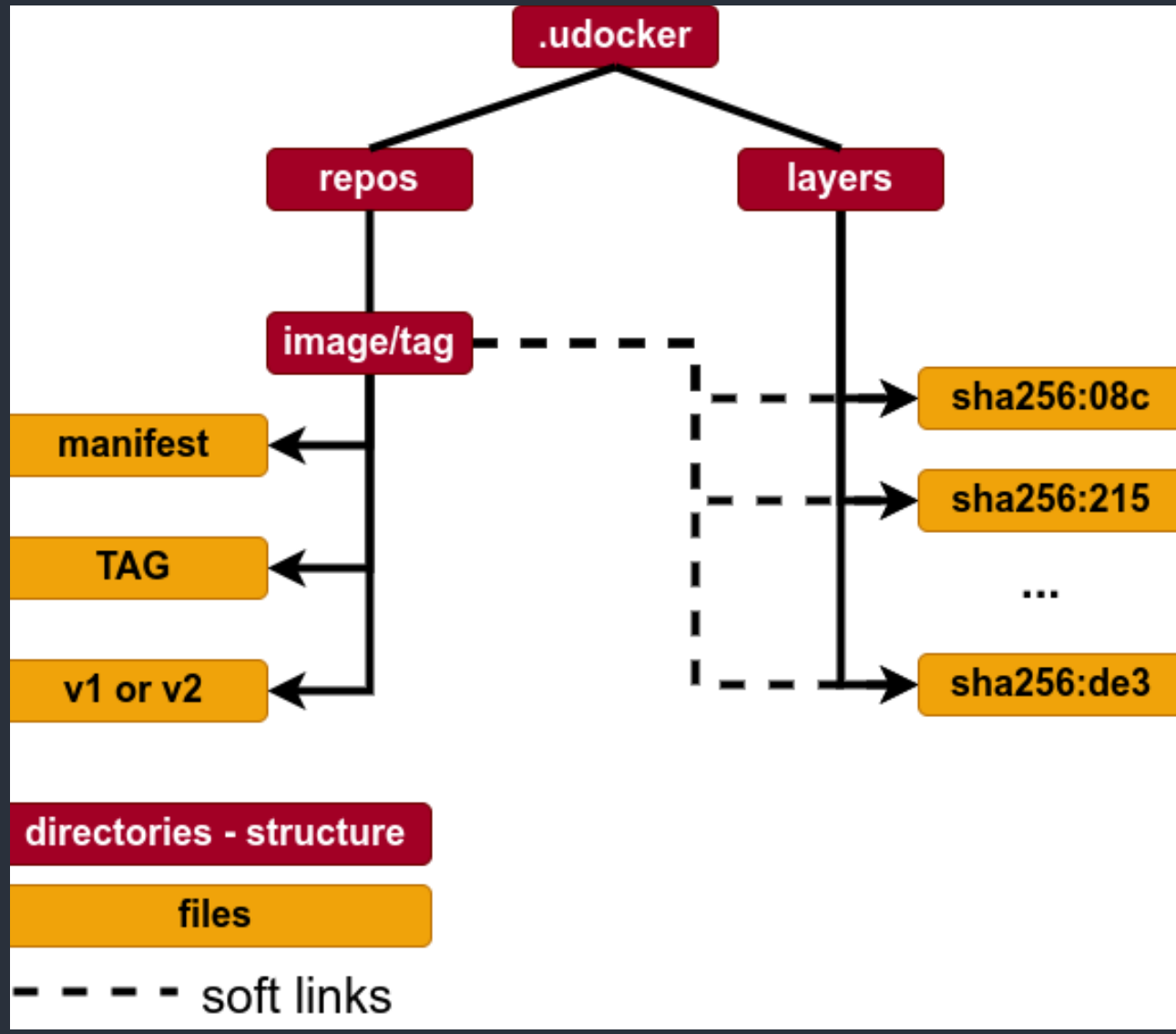
- Run using several execution methods.

udocker is an integration tool



udocker: pull - Images

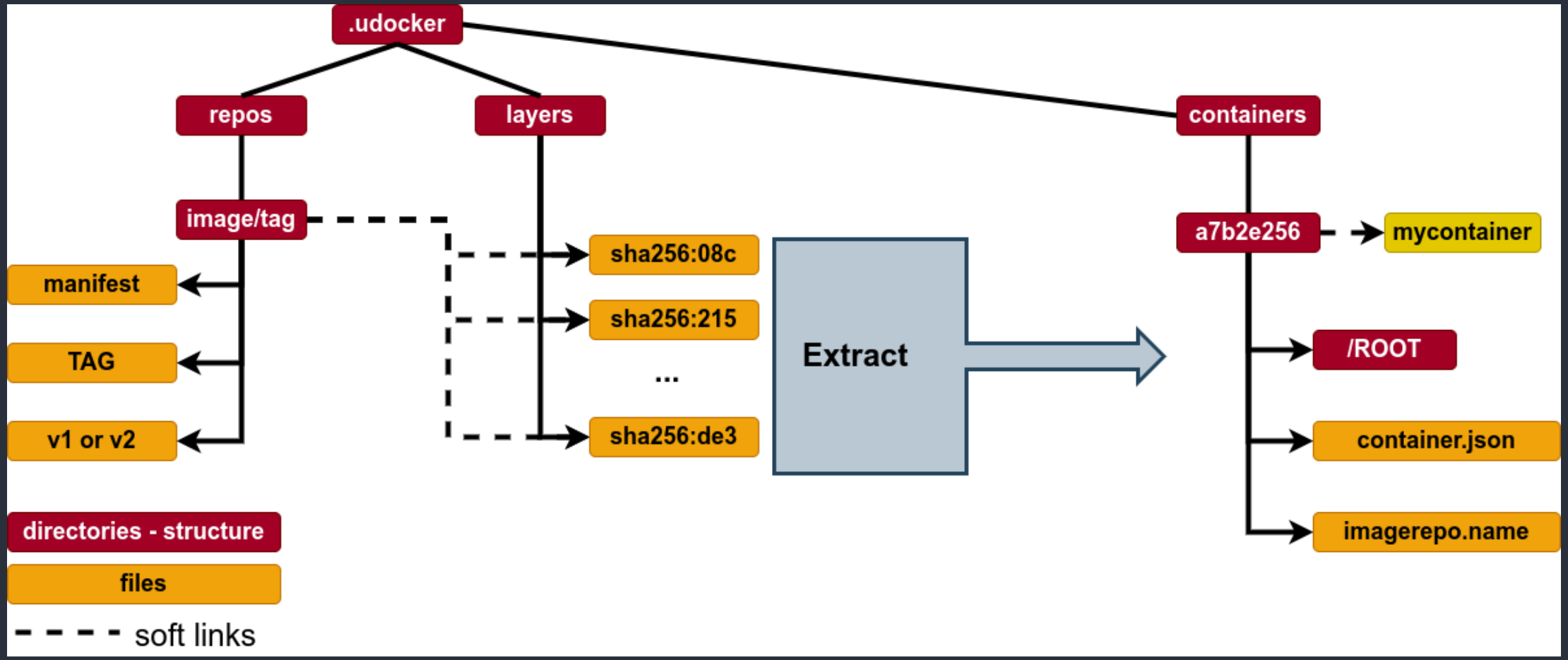
- Layers and metadata are pulled with DockerHub REST API.
- Image metadata is interpreted to identify the layers.
- Layers are stored in the use home directory under `${UDOCKER_DIR}/.udocker/layers`. so that can be share by multiple images.



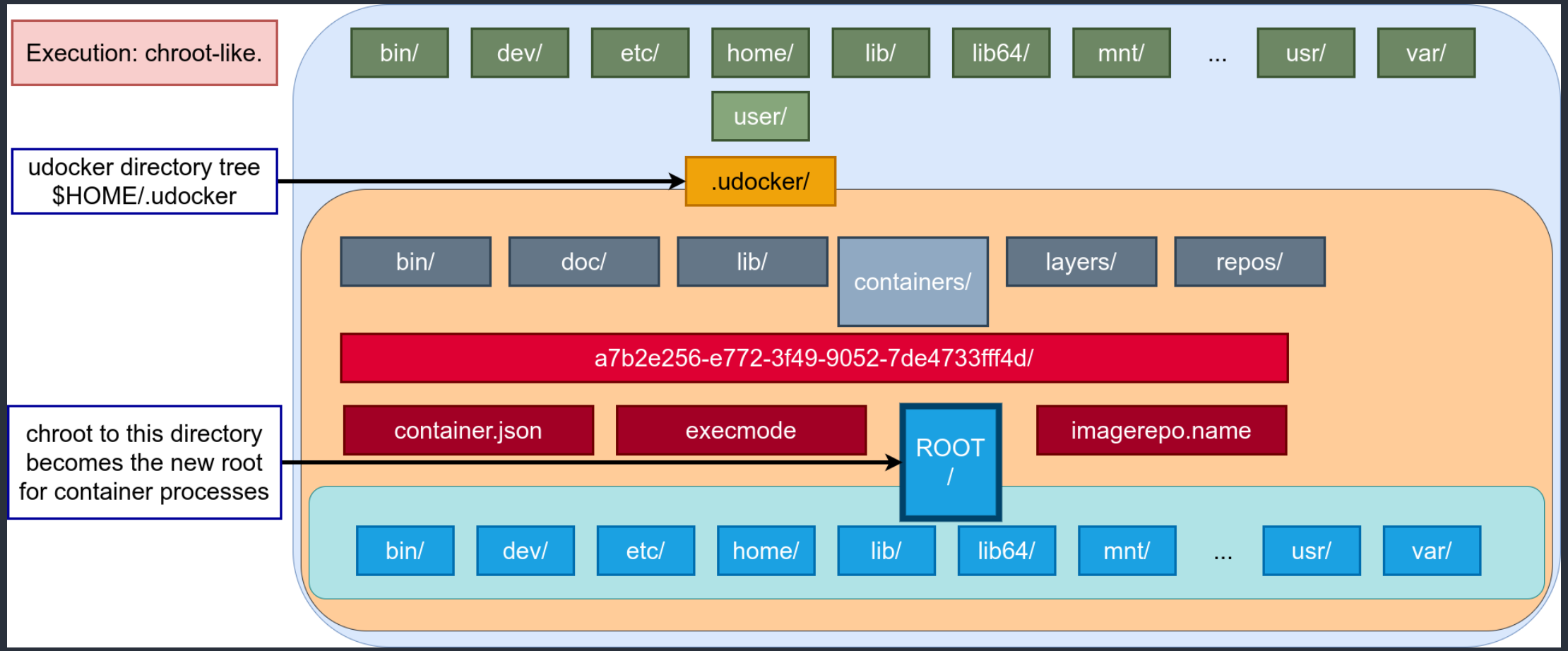
udocker: Create containers - I

- Are produced from the layers by flattening them.
- Each layer is extracted on top of the previous.
- Whiteouts are respected, protections are changed.
- The obtained directory trees are stored under `${UDOCKER_DIR}/.udocker/containers` in the user home directory.

udocker: Create containers - II



udocker: Run container



udocker: Execution engines I

udocker supports several techniques to achieve the equivalent to a chroot without using privileges, they are selected per container id via execution modes

udocker: Execution engines II

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

Selection in terms of performance

Mode	Base	Description
P1	PRoot	Multithreaded applications can suffer degradation
P2	PRoot	Same limitations as P1 apply. All system calls are traced causing higher overheads than P1
R1	runC	Same performance as namespace based applications
R2	runC	Only for software installation and similar. Same performance as P1
R3	runC	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 I

Mode	Base	Description
P1	PRoot	PTRACE + SECCOMP requires kernel >= 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	User namespace limitations apply
R2	runC	User namespace limitations apply. Same limitations as P1 also apply, this is a nested mode P1 over R
R3	runC	User namespace limitations apply. Same limitations as P2 also apply, this is a nested mode P2 over R

Selection in terms of interoperability II

Mode	Base	Description
F1	Fakechroot	May load host libraries. Requires shared library compiled against same libc as in container
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

udocker: Running applications ...

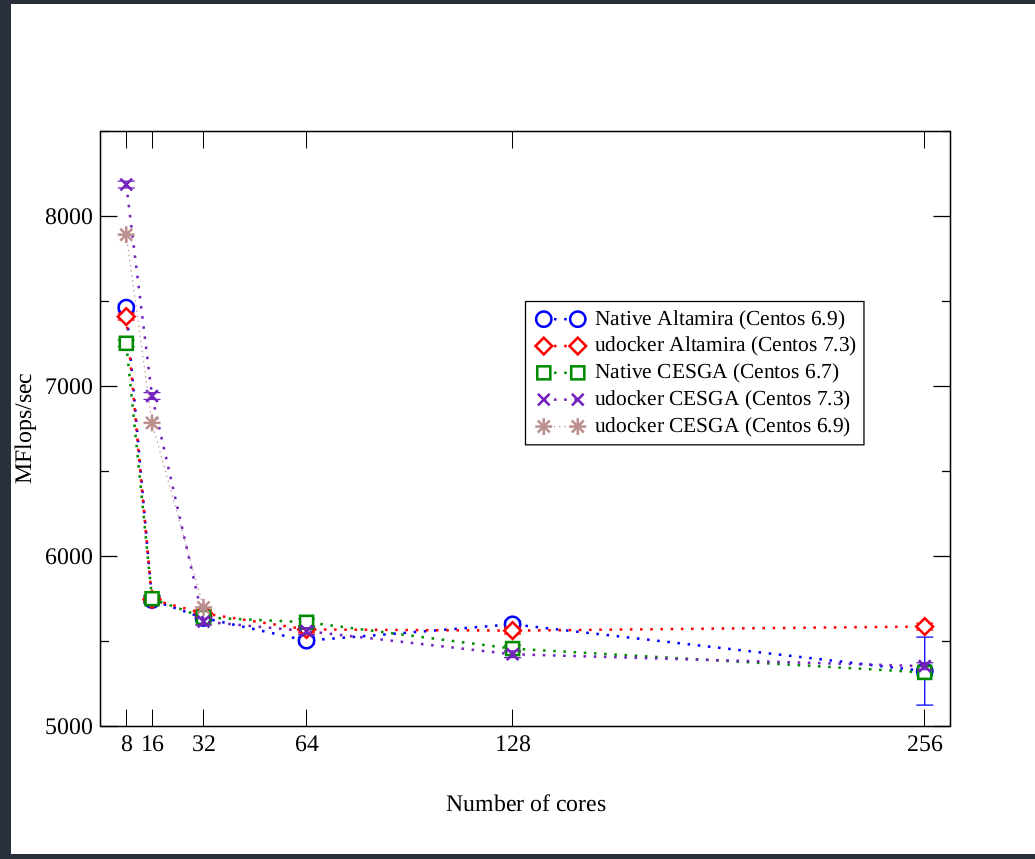
udocker & Lattice QCD

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.

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

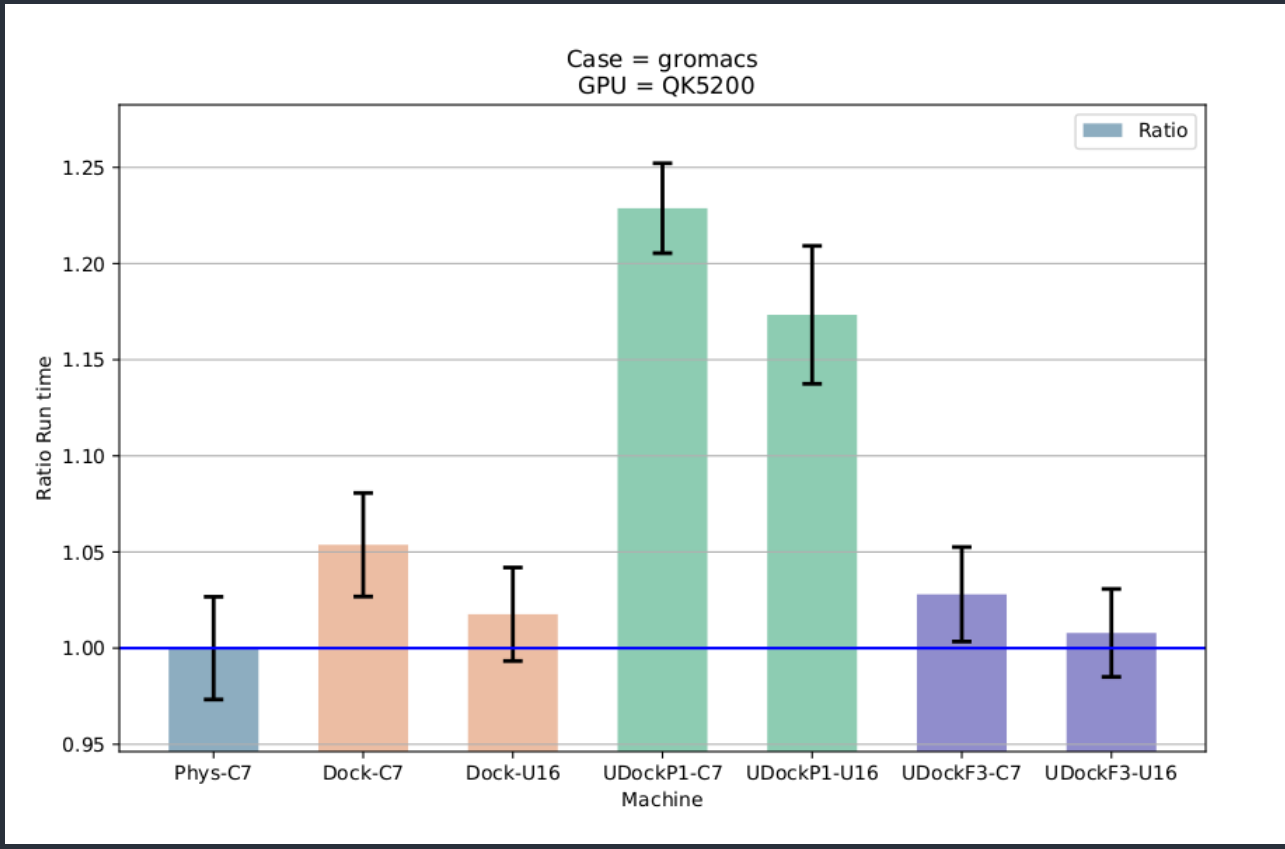


udocker & udocker & Molecular dynamics

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

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

Using CUDA and OpenMP



udocker & Phenomenology

MasterCode connects several complex codes. Hard to deploy. Scanning through large parameter spaces. High Throughput Computing.

C++, Fortran, many authors, legacy code. Performance Degradation (*udocker in P1 mode*)

Environment	Compiling	Running
HOST	0%	0%
DOCKER	10%	1.0%
udocker	7%	1.3%
VirtualBox	15%	1.6%
KVM	5%	2.6%

udocker: Next

udocker: What’s next

- Increase automation for MPI/infiniband applications:
  - OpenMPI and MPICH.
- Better translation of “volume” directories.
- Command line interface enhancements.
- Improve selection of binaries and libraries to be installed; dependent on host OS and architecture.
- Improve root emulation.

Thank you!

Questions ?

udocker@lip.pt

FCT Fundação para a Ciência e a Tecnologia
Lisb@20^20
COMPETE 2020
PORTUGAL 2020
UNIÃO EUROPEIA Fundo Europeu de Desenvolvimento Regional

Backup slides

Other container technologies

- Singularity (LBL) - udocker currently supports it as execution mode
- Charliecloud (LANL) - devels contacted Jorge: can udocker have a mode for it? "Merge" the udocker, CLI functionality with underlying Charliecloud engine?
- Shifter (NERSC) - at the moment no plans on any type of usage/integration in udocker.
- Podman (RedHat)