Hands-on Liger: containers

- Pierre-Emmanuel Guérin
- Davide Rovelli
- Hugues Digonnet

supercomputing.ec-nantes.fr / @cnscfr

CENTRALE
NANTES
SUPER COMPUTING

Containers overview

What are containers?

- Containers are executable units of software in which application code is packaged, along with its libraries and dependencies
- Think of it as an isolated box where you can install everything you need for your application
- Containers are **portable**: if it runs on your computer, it runs (almost) everywhere

Confusing terminology:

- Docker
- Singularity
- Apptainer
- PodMan
- microservices
- etc.

are all technologies involving containers, often used to indicate containers themselves.

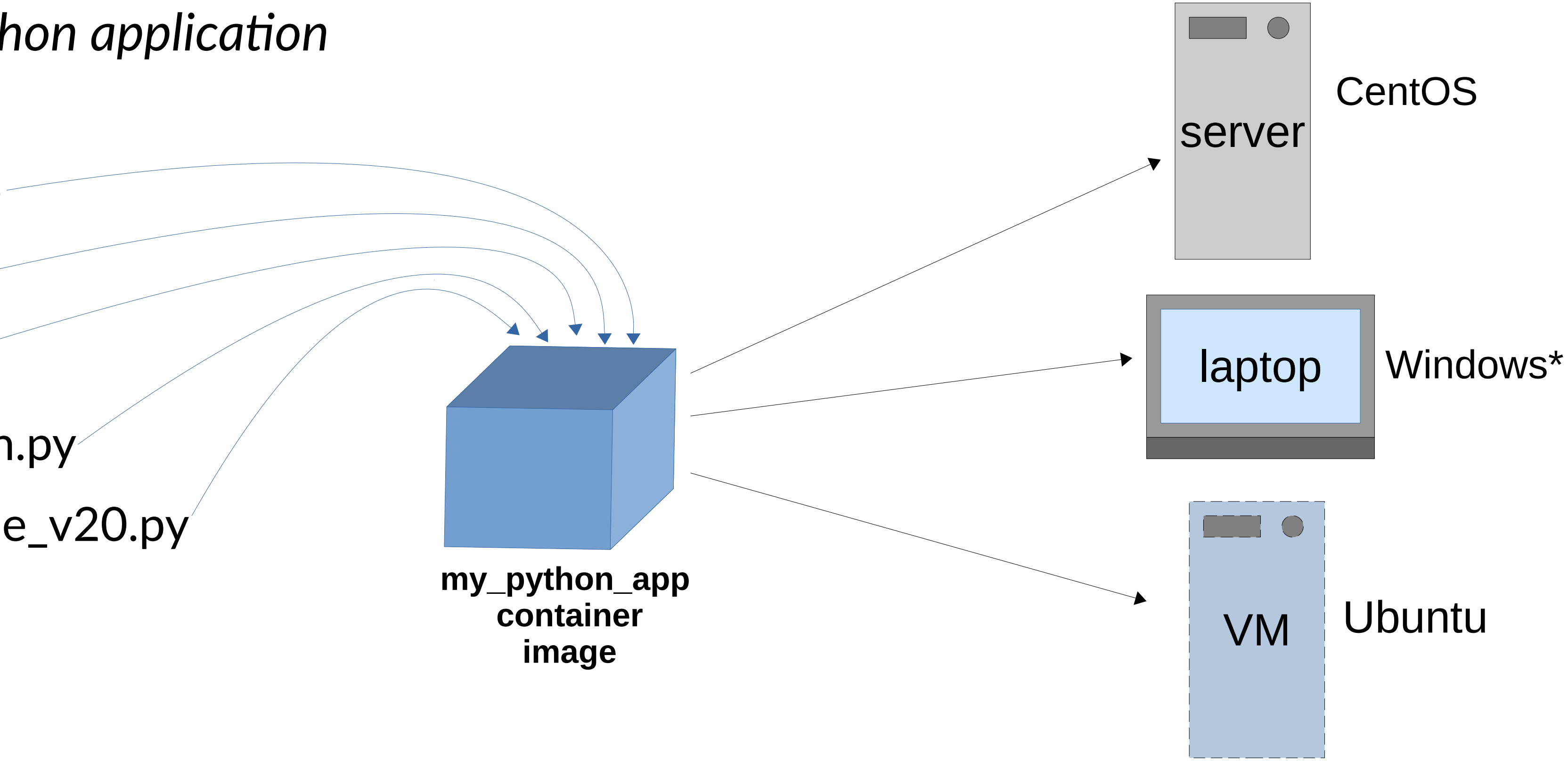
Resources:

- https://www.ibm.com/cloud/learn/containers
- https://apptainer.org/docs/user/main/introduction.html

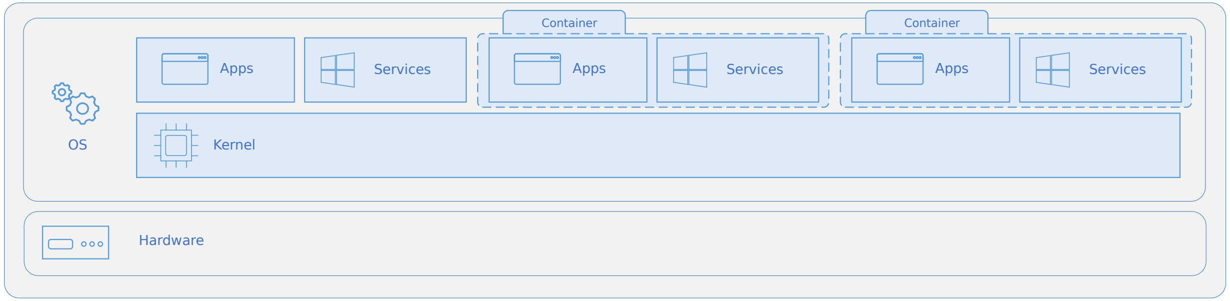
What are containers?

Example: *python application*

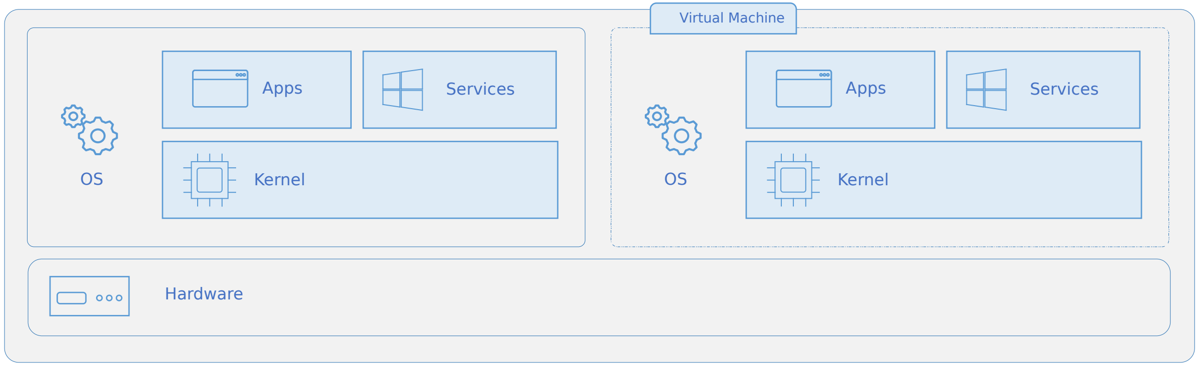
- Python3.9
- meshio
- numpy
- application.py
- my_module_v20.py



Container vs. Virtual Machine



Containers are a partial abstraction on top of the operating system kernel, managed by a container engine.



Virtual machines are a full operating system abstraction on top of the hardware, managed by an hypervisor.

Container vs. Virtual Machine

Differences

- Containers are application oriented, VMs are system oriented
- Containers are lightweight, VMs are heavy but isolate more
- Containers are easier to manage, start, stop: you can forget it is a container and think of it as a portable application

Resources:

- https://www.docker.com/blog/docker-hearts-wsl-2/
- https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm

Typical applications

- Cloud, microservices, DevOps
- Containers are used extensively for web application within orchestrators such as Kubernetes and Docker Swarm
- This configuration allows for fast scalability and high avalaibility that leverage on the reproducibility of the container
- Application development and deployment because of the compatibility and portability of containers
- Jupyter, Redis, DB synamic redundancy

Container engines

- Container lifecycle is managed by a **container engine**
  - Only dependency of containers
  - Several: Docker, Singularity(now apptainer), PodMan, Saurus, LXC, etc.
- **Docker**: most popular, most resources, mostly used
- Supercomputer do NOT support docker
  - Priviledge excalation: docker needs root/super user access.
  - Not suitable for multiuser systems as HPC clusters
- Liger and most HPC use **Singularity (now Apptainer)**
  - Compatible with Docker! Can use interchangeably

Container registries

- Containers live in registries: online repositories with thousands of images built by companies, application developers, researchers, community
- A container registry is easily accessible by the container engine.

```
docker pull ubuntu
```

```
singularity pull library://davide/myapp
```

Pull (download) from registry

```
docker push ecn-mech/solver1b
```

```
singularity push docker://ecn-mech/solver1b
```

Push custom images from registry

Containers in HPC

HPC software: traditional approach

- Admin installs HPC software and libraries on the supercomputer
- User can load and user a specific version of a software with module

```
module load python/3.8.1/gcc/4.8.5-c7
```

- Submit job via slurm, running or compiling the application with the selected modules
- No Module? Missing version? Compiling/execution error? Ask admin

HPC software with containers

- Requirement: supercomputer must have a container engine
  - Liger has Singularity
- Use available containers or
  - Pull an existing container with the software environment needed. It can be provided by admins or in an official registry
  - Copy your application inside and run
- Build your own container
  - Build a container with all the required dependencies and programs in your machine
  - Move it to the supercomputer and run it wih
- Submit job via slurm as usual
- Incompatibilities? Ask admin **or rebuild it yourself**

Performance

- Literature shows that modern containers add negligible computing overhead to applications
  - https://sc19.supercomputing.org/proceedings/tech_poster/poster_files/rpost227s2-file3.pdf
- Often built and optimised by the framework / OS / programming language developer therefore likely better than custom installation
- For the same reason: more likely to be bug-free
- Suggested standard for AI workflows

Why should I bother?

- Learning how to use containers is an overhead, why should I do it?
  - Wider software library: no module? Can use containers on public registries often provided directly by the software makers
  - Do not rely completely on admins. Installing software is hard… and reaching admins is even harder → long delay times.
    - U: Can you install this new software please?
    - A: *3 weeks and several build from source later* does it
    - U: It misses a library
    - A: *2 weeks later* reinstall with library
    - Program: *crashes* because incompatible with centos…
  - Make your app once, use it everywhere (different clusters, computers)
  - Paper? Can reproduce the results much more easily

Singularity + Docker

- Workflow:
  - Build and test images with Docker locally
  - Use Singularity on the supercomputer just for running and testing
- Advantage: more resources for Docker, well documented, more compatible registries
- Disadvantages:
  - Using 2 technologies → more to learn, impractical (?)
  - Slight differences that sometime require some readjustement
- Just one way to do it, nothing against full Singularity

Use and build containers

Important resources

- Liger docs container info (AI):

  https://ecn-collaborations.pages.in2p3.fr/liger-docs/artificial_intelligence/container_guide/

- Reference repository with useful tools:

  https://gitlab.in2p3.fr/ecn-collaborations/liger-ai-tools

- Container registry:

  https://gitlab.in2p3.fr/ecn-collaborations/liger-ai-tools/container_registry

- Singularity (Apptainer) docs:
  https://apptainer.org/docs/user/main/index.html

Use: pulling containers

–
```
module load singularity
```
Load singularity

–
```
export SINGULARITY_CACHEDIR=/scratch/$USER
```
Avoid overflowing /home quota

– Pull any container from any docker, singularity or OCI compliant registry!

```
singularity pull docker://{registry/img}
```
most used

```
singularity pull library://{registry/img}
```
private singularity registry

– Remember the tag! Tags are used to specify image versions

- Format: registry.io/image:**tag**. If tag is not specified, defaults to **latest**

Example: Recent version of GCC

Use: running containers

Still use singularity module but *make sure to clear all previous modules*

- `module purge`
- `module load singularity`

- **Exec**: run a command inside the container
-
```
singularity exec image.sif echo “hi from container”
```
- **Shell:** start a shell session inside the container

```
singularity shell image.sif
Singularity>
```

Example: Compile custom app in GCC container

Use: running containers

Useful options:

-
```
singularity exec –help
...
  -B, --bind strings              a user-bind path specification.  spec has
                                  the format src[:dest[:opts]], where src and
                                  dest are outside and inside paths.  If dest
                                  is not given, it is set equal to src.
                                  Mount options ('opts') may be specified as
                                  'ro' (read-only) or 'rw' (read/write, which
                                  is the default). Multiple bind paths can be
                                  given by a comma separated list.
  -e, --cleanenv                  clean environment before running container
  -c, --contain                   use minimal /dev and empty other
                                  directories (e.g. /tmp and $HOME) instead
                                  of sharing filesystems from your host
  -C, --containall                contain not only file systems, but also
                                  PID, IPC, and environment
...
      --nv                        enable experimental Nvidia support
...
```

Use: running containers

Focus on 2 options:

- Binding directories
  - Inside the container is a separate environment from the host – different OS, filesystem, programs
  - Therefore, directories that are on Liger are not visible by the container by default. They can be by a mechanism called binding, that is like “inserting a USB to the container”
  - Singularity binds the current folder by default. The rest needs to be bound explicitly with option -B
  - Syntax is `-B /source/folder:/container/folder`

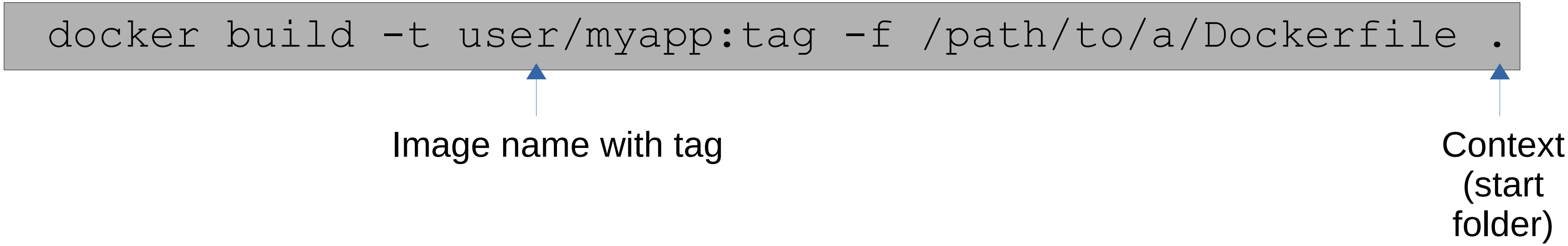
```
$ singularity exec paraview_egl-py3-5.9.0.sif ls /Myscratch
      ls: cannot access '/Myscratch': No such file or directory
$ singularity exec -B /scratch/drovelli:/Myscratch paraview_egl-py3-5.9.0.sif ls /Myscratch
      sif  singularity-cache     visu-1204115.txt
```

- Include NVIDIA libraries for GPU applications with `--nv`

Build: Dockerfiles

Build using Docker in your local machine (choice, could use Singularity directly)

- Build is specified in a **Dockerfile**: a list of instructions
- There are plenty of instructions, we will cover the basics
- https://docs.docker.com/engine/reference/builder/#usage



```
docker build -t user/myapp:tag -f /path/to/a/Dockerfile .
```

Image name with tag

Context
(start
folder)

Build: Dockerfiles instructions

- FROM image
  - A very powerful feature is building images from existing ones.
  - Allows for adding small changes to official established images (ex. Add a package)
  - Images can be local or on a registry online, docker will pull them automatically
- RUN command
  - Run shell commands inside the container
  - Install, create directories, set system options, compile etc.
- COPY
  - Copy a folder inside the container

Build: Dockerfile example

```
FROM gcc:10.3.0


# change directory
WORKDIR /workspace/
# download library
RUN wget https://gitlab.com/libeigen/eigen/-/archive/3.4.0/eigen-3.4.0.tar.gz
# extract library
RUN tar -xf eigen-3.4.0.tar.gz
# copy file inside the container
COPY matrix_init.cxx .
RUN g++ -I/workspace/eigen-3.4.0 /workspace/matrix_init.cxx -o /workspace/matrix_init
```

Build: manage images

- `docker images`
  - View all images that were built with docker
- `docker ps`
  - View all running containers
- `docker rm/rmi`
  - Remove containers/images
- `docker tag`
  - Rename existing images

Build: move images to Liger

- Via the registry
  - Docker push to the registry
  - Singularity pull on Liger
  - Liger has its own public registry that you can use (use/buy a private one if need confidentiality)
- Export the image to a compressed archive with

```
docker save ...
```

  - Move the image to Liger

Practical utils

Container registries out there

- Liger GitLab registry
- Dockerhub
- NGC (NVIDIA GPU Cloud)
- quay.io
- SyLabs Cloud Library
- CSAN: initiative to create a registry for scientific containers for french researchers
- ...many private and public ones

Containerized apps ready-to-use

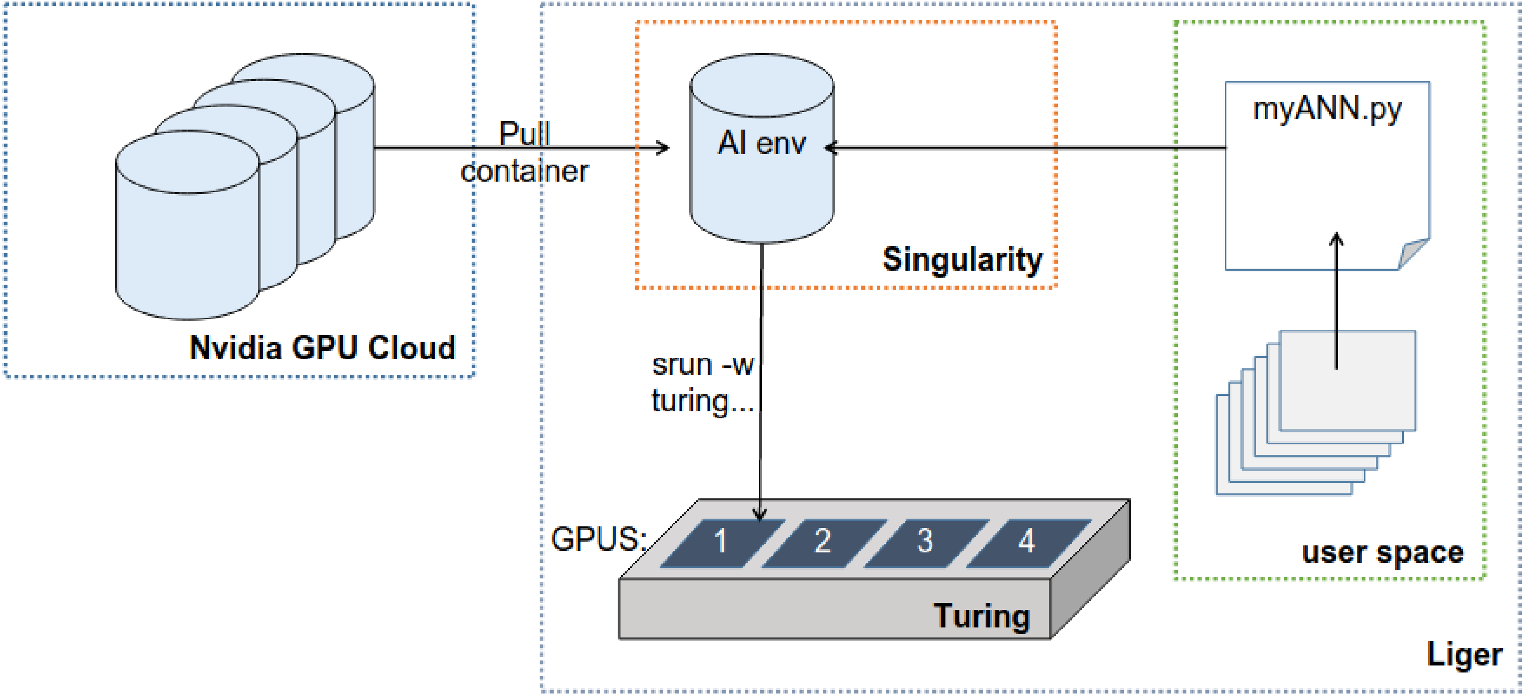
- AI: standard workflow has containers → plenty of resources
  - Tensorflow, Pytorch, major DL/ML frameworks
  - All sorts of packaged models for biology, chemistry
  - Good place to look is NGC
- HPC: not as common – some major apps starting to provide containers or recipes
  - FeniCS, code Aster etc.
- Standard software, compilers, programming languages make regular releases on official registies (often dockerhub)

AI containers in Liger

- Jupyter with Python, TensorFlow and common AI libraries
- GPU resources are configured to host **containerised** applications. The container engine on Liger is **singularity**
- Pre-build containers can be found on Liger and on the liger-ai-tools repo. Container description here
- Pre-built containers available on Liger at:

  /softs/singularity/containers/ai

Containerised applications - diagram



Conclusions

Pros and cons

To use or not to use containers?

- Performance is the same
- Several benefits but have to learn new workflow
- Depends on the type of work:
  - One time use, maybe better to stick with your current workflow
  - Developers and frequent users: might be worth to invest the time to save it in the future

Pros and cons

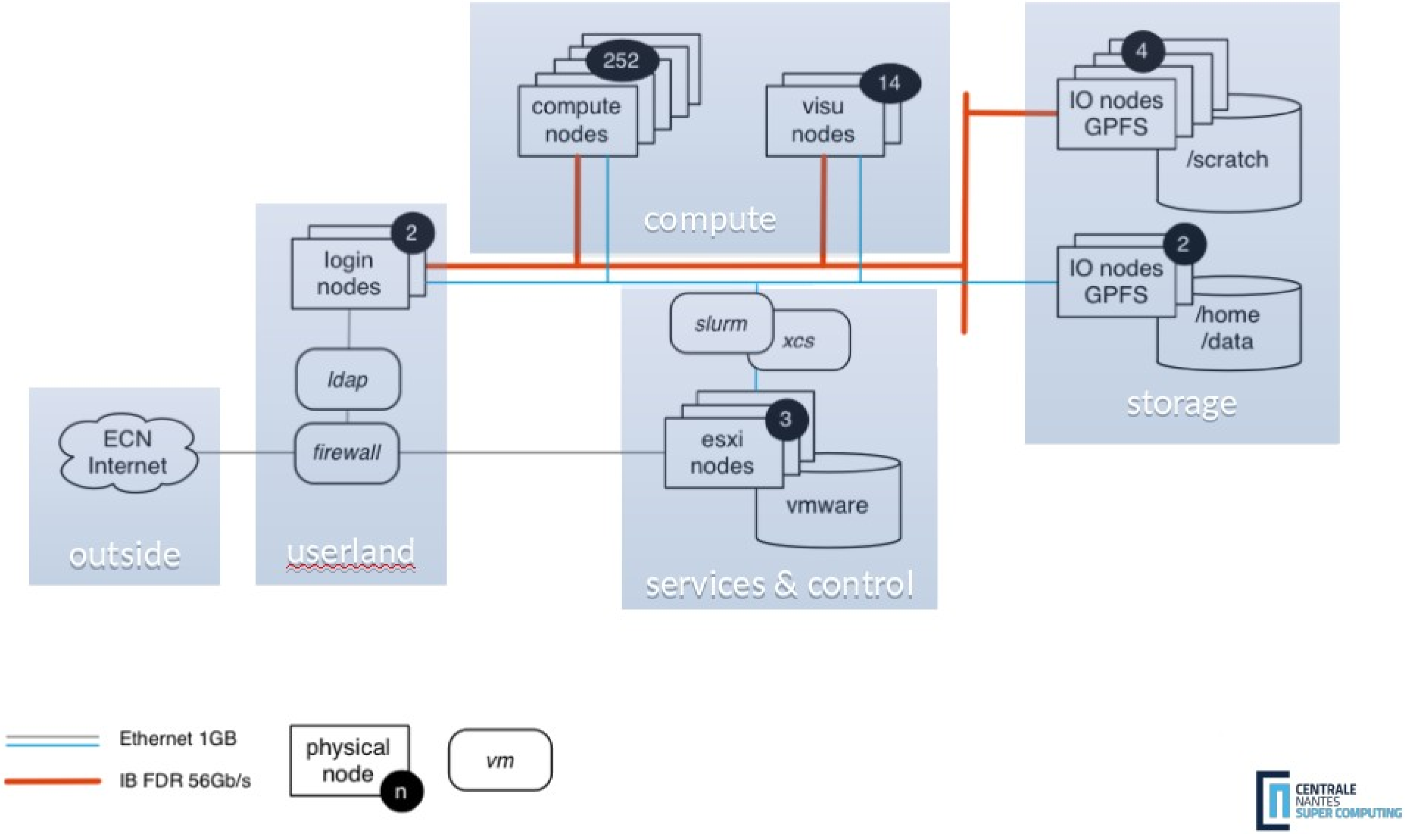
- Why can I not use CONDA instead? Python venv?
  - Conda doesn't work well on Liger :D
  - Resource consuming for HPC: every user has its own environment, no sharing
  - Only for python
- Other tools: GUIX, SPACK…
  - GUIX ensure higher reproducibility but less widespread
  - Matter of preference?

Questions?

Appendix

Liger basics

Liger: system topology



User Env : Filesystems & storage

- /scratch
  - 815 TB, 1 000 000 files quota per user
  - Your directory is $SCRATCHDIR
  - Computations and temporary files
- /home
  - 30 TB, 5GB quota soft per user
  - Your directory is $HOME
  - Sources files
- /data
  - 45 TB, quota per group={100GB and 2 million files}
  - Your project directory is $DATADIR
  - Permanent projects data and group sharing data

User space

Project space

Connect to Liger

- Client tool to connect on remote console:
  - Windows : PowerShell, putty, cygwin, mobaxterm
  - Mac/Linux : xterm, xquartz (only mac)
- Use a VPN to connect to Centrale Nantes network
- SSH secure protocol

```
$ ssh myUsername@liger.ec-nantes.fr
```

Move files to Liger

- SCP (or WinSCP for Windows): secure copy
  - Example: tranfer program to */home*

```
$ scp ./Desktop/program.c LIGER-ID@liger.ec-nantes.fr:~
```

  - WinSCP: GUI, same principle
- Download directly on Liger: git, wget etc.
  - Example: clone git repository on scratch

```
$ git clone https://repo.git $SCRATCHDIR
```

Job submission

- Compute resources are managed by a scheduler:
  - Liger uses **SLURM**
- Jobs are submitted to the scheduler
  - The scheduler choose available nodes (job running)
  - Or the computation is queued (job pending)

Job submission

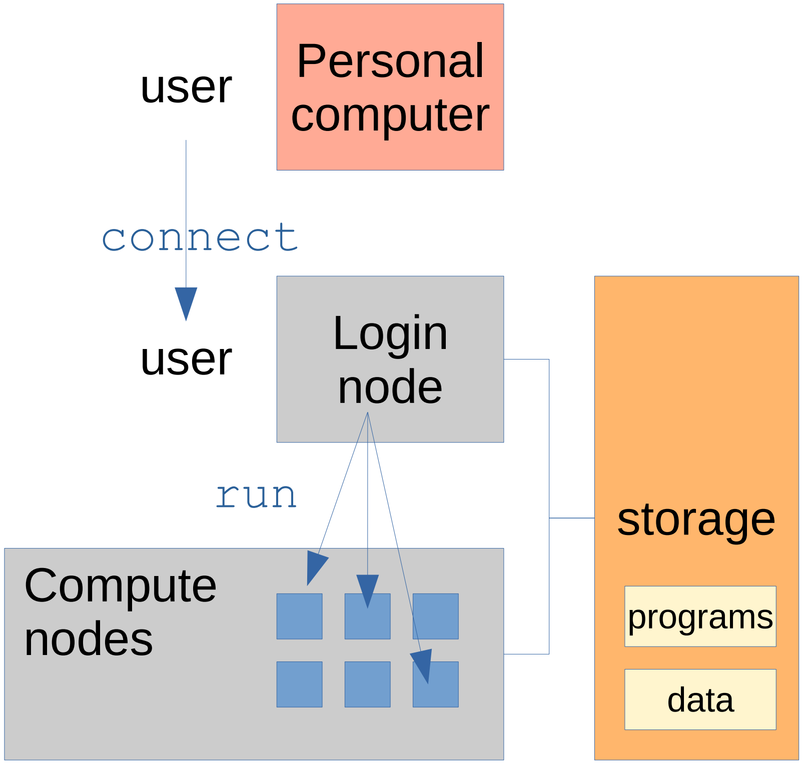
- With slurm commands you can run program on compute nodes.
  - Tell the SLURM what to run
  - SLURM will find the available resources and run the program

```
$ srun PROGRAM # run a job in the foreground
```

```
$ sbatch SCRIPT # run a job in the background
```

Liger : User environment

- You have 3 directories
- You can compile and test codes on login nodes
- You can use available softwares/libraries
- And you can submit jobs on nodes.



Load programs: *modules*

- Your environment is initally empty: no programs installed
- Modules is a tool to load or unload software packages.
  - List available software

```
$ module avail
```

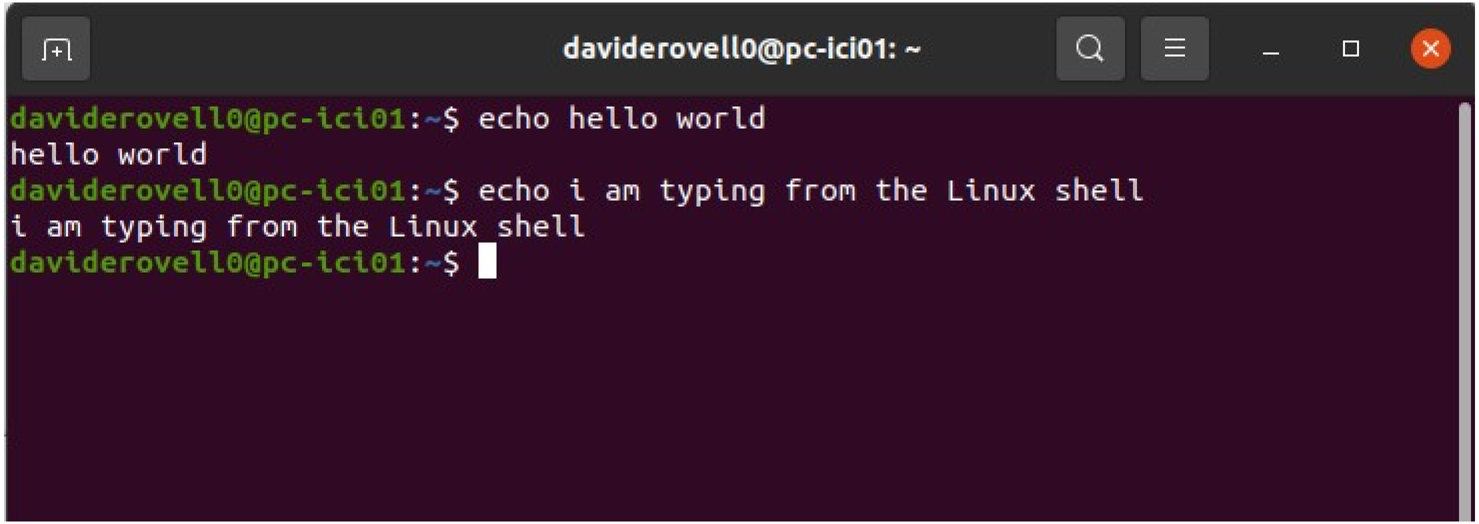
  - Load python

```
$ module load python
```

Linux shell basics

The Linux shell - terminal

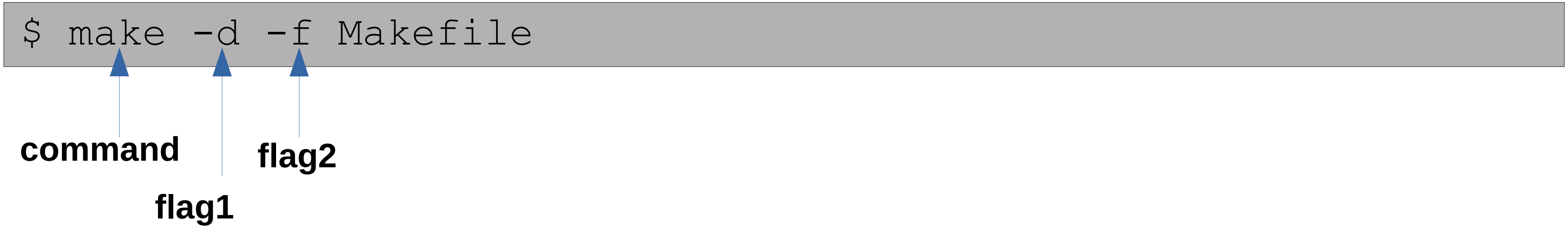
- No Graphical User Interface
- Issue commands through a CLI: command lone interface



```
daviderovell0@pc-ici01:~$ echo hello world
hello world
daviderovell0@pc-ici01:~$ echo i am typing from the Linux shell
i am typing from the Linux shell
daviderovell0@pc-ici01:~$ 
```

Issuing commands

- A command is a program that corresponds to a string of text. Use return to send a command, ctrl-C to interrupt it.
- A command can have **options**, set through **flags**.



```
$ make -d -f Makefile
```

- The “**-h**” flag shows a help guide for most commands

Navigating directories

- *pwd* – shows which directory you are in
- *ls* – list the files in the current directory
- *cd* – change to another directory

The base folder (top of the tree) is represented by “/”

The current folder is represented by “.”

The parent folder is represented by “..”

Editing files

- *cp* – copy a file to another location
- *mv* – move the file to another location (used for renameing as well)
- *rm* – remove a file, **-r** flag for recursive and folders

General rule: all commands are executed in the current folder (*pwd*), to execute a command in another folder use its path:

/absolute/path/to/file *relative/path/to/file*

File operations

- Text editors: *nano*, *vi*, *gedit (requires GUI)*
  - Relies on a lot of key combinations, can be hard at the beginning. Use an editor wherever possible
- View file content: *cat*, *less etc*

```
$ cat your_file.txt
```

Run programs

- *gcc* – C / C++ compiler
- *python3* – run a Python script
- Javac – run a Java program
- …any installed program. Install with package manager:
  - Ubuntu, Debian: *apt*
  - RHEL: *yum*

Useful resources

There's much much more!

- https://supercomputing.ec-nantes.fr/publications/tutorials
- https://projects.ncsu.edu/hpc/Documents/unixtut/
- http://swcarpentry.github.io/shell-novice/