The HPC@PoliTO User Guide

June 2025, v0.4 (DRAFT)

Contents

Basic Rules

1. **Never execute simulations or intensive tasks directly in your shell**: this would disrupt operations on the login server, making the HPC infrastructure unavailable for the whole community.

2. To execute their jobs, users should submit batch requests to the SLURM scheduler. For intensive post-processing, code compilation or large data transfers, users should instead request interactive sessions to the SLURM scheduler.

3. The HPC@PoliTO infrastructures should only be used for academic or teaching purposes.

4. Users are responsible for all activities originating from their account. They should therefore be aware about their account security, adopt safe practices and avoid accounts sharing.

5. Users agree to acknowledge the HPC@PoliTO initiative in any publications and talks that use data originating from our facilities. We suggest the following citation:

   "Computational resources provided by HPC@PoliTO (www.hpc.polito.it)"

6. After the expiration of an HPC account, data in the home directory are stored for a limited amount of time (t.b.d., hypothetically 6 months). Users are responsible for collecting and downloading their data before their account expires or asking HPC@PoliTO for their recovery as soon as the account expires (provided it will not be extended).

Chapter 1

The HPC@PoliTO Infrastructure

1.1 Introduction

The HPC@PoliTO infrastructure is currently made of two separate clusters. Hactar is our legacy cluster, it was first deployed in 2015 and currently only used for light workloads by students teams and for teaching purposes. Legion is our flagship cluster, it was first deployed in 2019 and underwent a major upgrade at the end of 2024. It is used by many research groups at Politecnico di Torino (PoliTO), whose financial contributions are crucial for the overall evolution of the cluster.

1.2 The Legion cluster

Legion was originally deployed in 2019 and underwent a first early expansion in 2020. Between the end of 2024 and the first half of 2025, the cluster was the subject of a major upgrade that roughly doubled its compute node count. To manage different technologies between old and new hardware, two major partitions have been defined: Isola 1 and Isola 2.
Although Isola 1 and Isola 2 are logically separated and jobs should not be run across the two (to avoid bottlenecks due to different generations hardware working together), overall Legion is considered a single cluster: sharing login, management and storage infrastructures. When submitting jobs users should specify which partition they want to use.
Legion is a shared resource batch system whose job scheduling is based on SLURM [?]. Further information about job submission and scheduling policies can be found in Chapter 5.

Node type	CPU	Cores	RAM	GPU	Disk
CPU node	2 x Intel Xeon 6130	32	384 GB	N/A	1 TB HDD
GPU node	2 x Intel Xeon 6130	32	384 GB	4 x V100 (32 GB)	1 TB HDD

Table 1.1: Legion Isola 1 - Compute nodes

1.2.1 Legion Isola 1

Legion Isola 1 collects early-deployed compute nodes (from 2019-2020) which are then further partitioned accordingly to different access policies. Isola 1, as represented in Figure 1.1 contains 58 compute nodes, 6 of which are configured with NVIDIA V100 GPUs. The compute nodes within Isola 1 have the characteristics reported in Table 1.1.

The nodes within Isola 1 are then logically divided into partitions with different access policies; in Table 1.2 each partition is reported together with its usage policies.

Open access partitions are available to each HPC user and are defined among all available compute nodes. Users scheduling jobs on these partitions are given the same priority and are subject to stricter job walltimes.

Restricted partitions are defined among a subset of the available compute nodes, as represented in Figure 1.2. Users may have access to restricted partitions if their research group (or the group they are collaborating with) provided financial support to buy compute nodes. Jobs scheduled on restricted partitions are given the highest possible priority in order to start as soon as the requested resources are available.

Contrary from past implementations, Legion partitions do not implement job suspension or preemption: high priority jobs would eventually need to wait for jobs running on open partitions to finish before they can start.

1.2.2 Legion Isola 2

Legion Isola 2 was deployed between late 2024 and early 2025 following a dramatic expansion of computing resources thanks to the contribution of research groups at PoliTO and the availability of NextGenEU (PNRR) grants. Isola 2, as represented in Figure 1.3 contains 73 compute nodes, 26 of which are configured with NVIDIA A40 GPUs and another 1 configured with NVIDIA A100 GPUs. The compute nodes within Isola 2 have the characteristics reported in Table 1.3.

The nodes within Isola 2 are then logically divided into partitions with different access policies; in Table 1.4 each partition is reported together with its usage policies.

Open access partitions are available to each HPC user and are defined among all available compute nodes. Users scheduling jobs on these partitions are given the same priority and are subject to stricter job walltimes.

Restricted partitions are defined among a subset of the available compute nodes,

Partition	Nodes	Access	Priority	Walltime
cpu_skylake	52 (CPU)	open	standard	120 hrs
cpu_skylake_ext	10 (CPU)	open	standard	240 hrs
gpu_v100	6 (GPU)	open	standard	120 hrs
gpu_v100_ext	1 (GPU)	open	standard	240 hrs
disma_skylake	16 (CPU)	restricted	highest	N/A
denerg_skylake	6 (CPU)	restricted	highest	N/A
bluen_skylake	6 (CPU)	restricted	highest	N/A
e3_skylake	3 (CPU)	restricted	highest	N/A
simdome_skylake	2 (CPU)	restricted	highest	N/A
small_skylake	2 (CPU)	restricted	highest	N/A
cwc_skylake	2 (CPU)	restricted	highest	N/A
andriulli_skylake	2 (CPU)	restricted	highest	N/A
nemo_skylake	2 (CPU)	restricted	highest	N/A
pt-erc_skylake	1 (CPU)	restricted	highest	N/A
polive_skylake	1 (CPU)	restricted	highest	N/A
waterview_v100	1 (GPU)	restricted	highest	N/A
simdome_v100	1 (GPU)	restricted	highest	N/A
disma_v100	1 (GPU)	restricted	highest	N/A
smartdata_v100	1 (GPU)	restricted	highest	N/A
energycenter_v100	1 (GPU)	restricted	highest	N/A

Table 1.2: Legion Isola 1 - Partitions

Node type	CPU	Cores	RAM	GPU	Disk
CPU node	2 x Intel Xeon 6442Y	48	512 GB	N/A	1 TB NVMe
GPU node A40	2 x Intel Xeon 6442Y	48	512 GB	4 x A40 (48 GB)	1 TB NVMe
GPU node A100	2 x Intel Xeon 6442Y	48	1024 GB	4 x A100 (80 GB)	1 TB NVMe

Table 1.3: Legion Isola 2 - Compute nodes

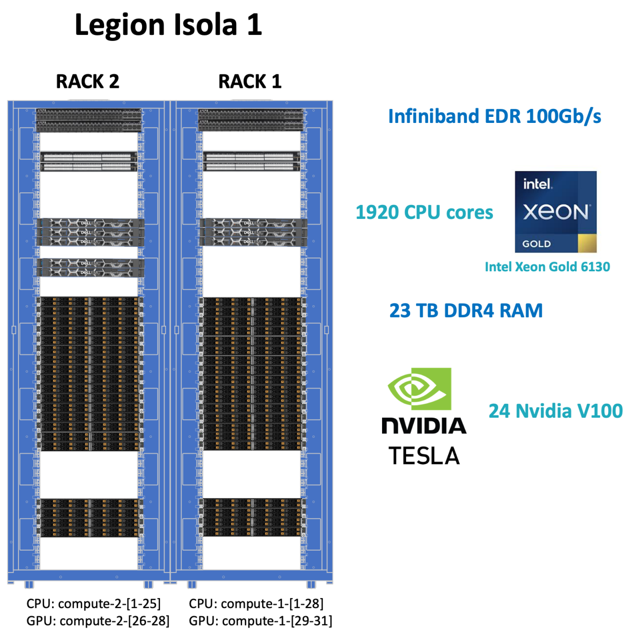

Figure 1.1: Legion Isola 1

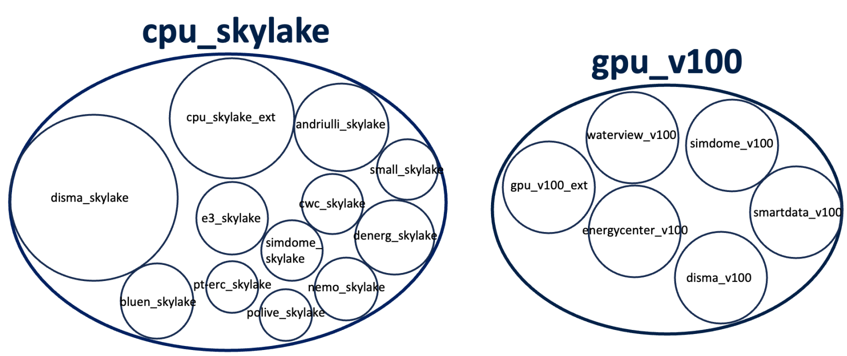

Figure 1.2: Partitions Legion Isola 1

Partition	Nodes	Access	Priority	Walltime
cpu_sapphire	45 (CPU)	open	standard	24 hrs
cpu_sapphire_ext	2 (CPU)	open	standard	120 hrs
gpu_a40	27 (GPU)	open	standard	24 hrs
gpu_a40_ext	4 (GPU)	open	standard	120 hrs
gpu_a100	1 (GPU)	open	standard	24 hrs
lining_cpu	23 (CPU)	restricted	highest	N/A
nemo_cpu	3 (CPU)	restricted	highest	N/A
melodizer_cpu	2 (CPU)	restricted	highest	N/A
mics_cpu	2 (CPU)	restricted	highest	N/A
serics_cpu	2 (CPU)	restricted	highest	N/A
teperc_cpu	2 (CPU)	restricted	highest	N/A
ddambrosio_cpu	2 (CPU)	restricted	highest	N/A
crest_cpu	2 (CPU)	restricted	highest	N/A
nodes_cpu	1 (CPU)	restricted	highest	N/A
comp_cpu	1 (CPU)	restricted	highest	N/A
hester_cpu	1 (CPU)	restricted	highest	N/A
invernizzi_cpu	1 (CPU)	restricted	highest	N/A
rossi_cpu	1 (CPU)	restricted	highest	N/A
fair_gpu	6 (GPU)	restricted	highest	N/A
restart_gpu	6 (GPU)	restricted	highest	N/A
smartdata_gpu	6 (GPU)	restricted	highest	N/A
serics_gpu	3 (GPU)	restricted	highest	N/A
musychen_gpu	2 (GPU)	restricted	highest	N/A
cgvg_gpu	1 (GPU)	restricted	highest	N/A

Table 1.4: Legion Isola 2 - Partitions

as represented in Figure 1.4. Users may have access to restricted partitions if their research group (or the group they are collaborating with) provided financial support to buy compute nodes. Jobs scheduled on restricted partitions are given the highest possible priority in order to start as soon as the requested resources are available.
Legion partitions do not implement job suspension or preemption: high priority jobs would eventually need to wait for jobs running on open partitions to finish before they can start.

1.2.3 Cluster networking

Legion deals on three separate networks to operate:

- Management network (1 GbE): to remotely manage individual servers;
- Service network (10 GbE): used by cluster services and for user homes;

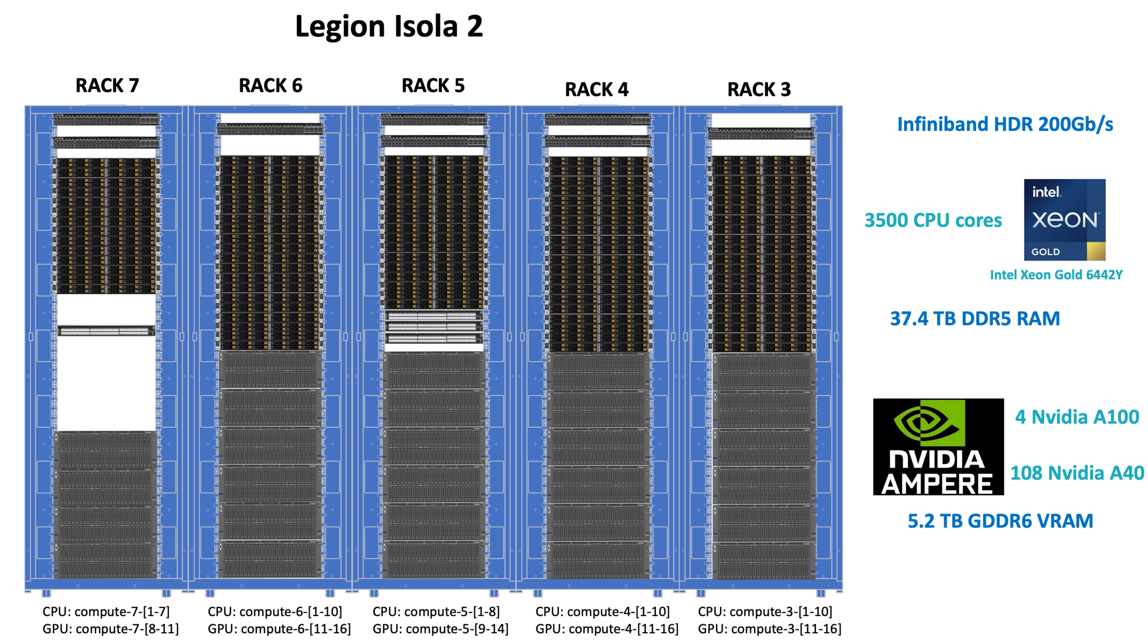

Figure 1.3: Legion Isola 2

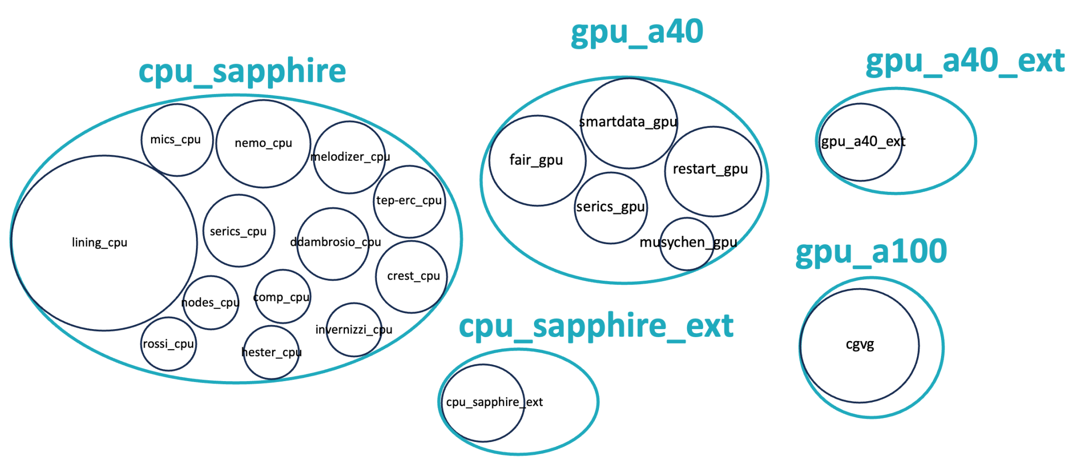

Figure 1.4: Partitions Legion Isola 2

- Infiniband network: used for multi-node jobs and high-performance IO.

Management and Service are implemented as standard TCP/IP ethernet networks, with 1 Gb/s and 10 Gb/s bandwidth, respectively. The network topology is linear with all switches connected at the same hierarchical level.

The architecture of the Infiniband network reflects instead the partitioning of Legion into Isola 1 and Isola 2 as two parallely connected fat-tree structures. Isola 1 implements a 100 Gb/s Infiniband EDR fat-tree network, while Isola 2 a 200 Gb/s Infiniband HDR fat-tree network. The two root switches are then interconnected with an aggregate of Infiniband EDR links resulting in a 600 Gb/s bandwidth. Although multi-node MPI jobs should not be run between Isola 1 and Isola 2 nodes, a high-bandwidth interconnection between the two guarantees an effective sharing of the storage infrastructure.

These different networks operate in the backend of the cluster and their usage is transparent for users when everything works fine.

1.3 Storage systems

The High Performance Computing infrastructure is supported by two complementary storage systems: a high-capacity Network Attached Storage (NAS) to store user homes and data and a lower-capacity BeeGFS parallel filesystem that provides a scratch space for IO intensive workloads that require parallel high-speed operations.

1.3.1 Home storage

The Home storage system is implemented with a high-capacity Network Area Storage (NAS) that exports user home folders to login and compute nodes using the NFS protocol over the Service network (10 GbE).

This system provides roughly 1.2 PB of available space to be shared among all users. User quota for the Home storage is defaulted to 1.5 TB, with possible temporary extensions to be granted upon motivated request. User homes are subject to automatic removal after 6 months from the account expiration.

1.3.2 BeeGFS scratch

The BeeGFS scratch is implemented with a state-of-the art multi-node storage system supported by a fault-tolerant backend network of full-NVMe LUNs.

This system provides roughly 800 TB of available space to be shared among all users. User quota for the BeeGFS scratch is defaulted to 5 TB and subject to automatic cleaning of all files older than 30 days. Temporary extensions of the quota may be granted upon motivated request.

Chapter 2

First login to the clusters

2.1 Connecting to the Legion login node via *ssh*

To connect to the cluster, use the `ssh` protocol (Secure Shell). Open a new shell window in your terminal. On Linux and macOS devices, you can use the native terminal. For Windows users, we recommend using a PowerShell terminal, though it is also possible to open an `ssh` connection directly from the Command Prompt. The shell window can be opened in any directory.

```
ssh username@hpc-legionlogin.polito.it
```

If this is the first time you are connecting to the cluster, you will be asked to authenticate the machine, confirm by typing `yes`)

As illustrated in Figure3.1, if this is your first time logging into Legion you will be asked to prompt the One-Time Password (OTP) that you have been provided when the account was activated. When entering the password no characters or placeholders will appear on the screen; once you have typed your password, press `enter`. The system will ask you immediately to change your password: follow the prompts you are given.

Upon successfull authentication, a new bash shell will be opened, the cluster's welcome page will appear and you will be in your home directory on the Home storage: `/home/username`.

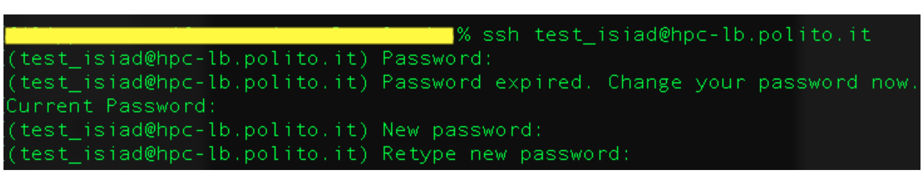

Figure 2.1: First time access

2.2 Basic commands

2.2.1 `man` and `--help` command

To see the full documentation about any command in the terminal, use the command:

```
man <command>
```

To have a quick idea of what the command does and the option of the command, use:

```
<command> --help
```

The former command provides the full documentation, the latter provides a summary of the syntax and of the more commonly used option (usually sufficient for most users).

2.2.2 `pwd`: your location

To see your current location in the file system, use the command:

```
pwd
```

This command prints the absolute path of the current directory.

2.2.3 `ls`: See the element inside a directory

To see the elements inside a directory use the command

```
ls [options] <path>
```

If you provide a directory name or a path, the command will show the contents of that specific directory. Use the `-a` option to include hidden files (name starts with.), or `-l` to view detailed information about the contents (e.g. dimension, owner, last modified, permissions...). Options can be combined (e.g., `-la`) to display both sets of information. the syntax is: `-la`).

2.2.4 `cd`: navigate inside the file system

To navigate among folders in the filesystem use the command `cd` (Change Directory). Provide the directory name or its path (paths always start with `/`).

```
cd directory_name
```

To go in the directory before (on the path), use

```
cd ..
```

2.2.5 Creating and removing directories

To create a new directory use the command:

```
mkdir directory_name
```

If you do not specify a location, the directory will be created at your current position.
To remove a directory, with all the files stored inside, use:

```
rm -r directory_name
```

2.2.6 `Vim`: file editor

With Vim, you can modify existing files or create new ones:

```
vi file_name
```

If the file_name does not exists the system will automatically create a file with that name in your location (you can open a file in a location different specifying the absolute path of the file). The main commands to use Vim are:

- `i`: insert text (use `esc` to quit inserting mode)
- `:wq`: save and exit
- `:q!`: exit without exiting

2.3 Copying files to and from the cluster

If you need to copy files or directories between your local host and the cluster (or vice versa), you can use the `scp` command (Secure Copy Protocol). Open a new shell window on your local host and use the following commands: `scp` (Secure Copy Protocol). From your host to the cluster

```
scp -r path_local_host \\
        username@{login_node}:/home/username/path_server_side
```

From the cluster to your host (notice that the following command must be run on your host)

```
scp -r username@{login_node}:/home/username/path_server_side
       path_local_host
```

If you have not set up the passwordless login, you will be prompted your password, otherwise, the copying process starts. The option `-r` is needed when copying directories, as it enables recursive copying of all files and subdirectories.

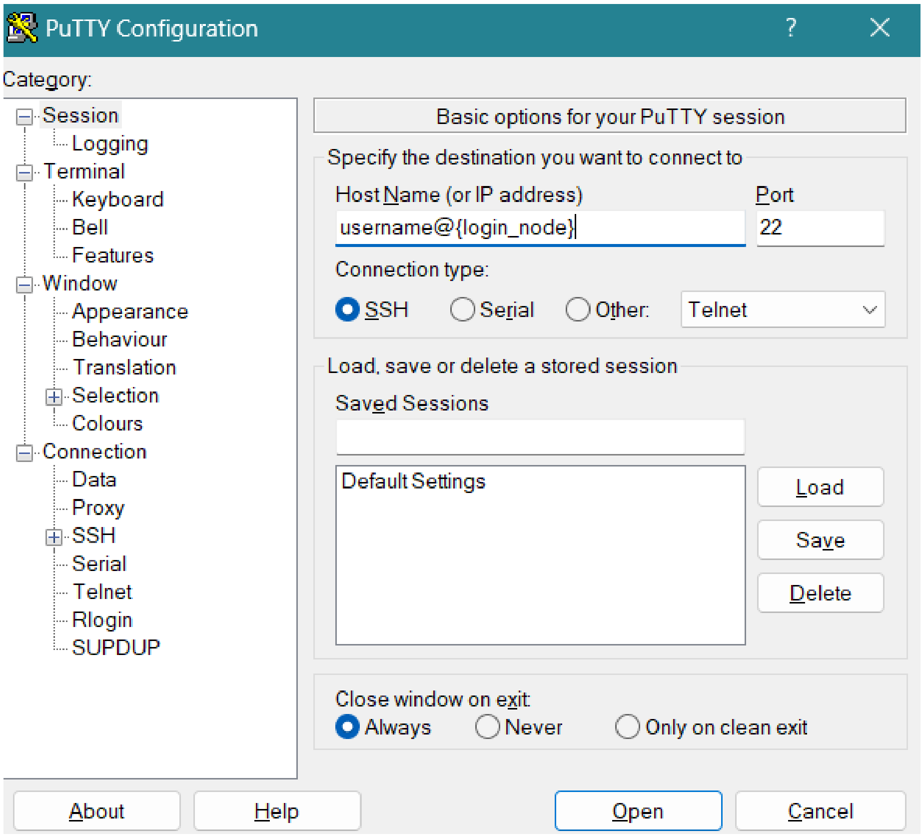

Figure 2.2: PuTTY setup

2.4 Using PuTTY (only on Windows machine)

From a Windows device, instead of using the terminal, it is possible to use PuTTY, a third-part application, to log in to the cluster. PuTTY emulates a Linux terminal on your host device and is available for download at `https://putty.org/`. To configure PuTTY, enter in the host name section your username and the login node to which you want to connect. We suggest to save the session using the specific area on the PuTTY main page. Once you have completed the procedure, open the connection, you will be prompted for the password.

Chapter 3

Connection to the clusters via passwordless-ssh

3.1 Generate key pair from a terminal

After receiving the HPC@PoliTO credentials, to enhance their account security, users should proceed to create their own OpenSSH private/pubic key pair (RSA). More information on RSA cryptography can be found at this link: `https://en.wikipedia.org/wiki/RSA_cryptosystem`.

To generate the private/public key pair you should first open a new shell window in their terminal (Linux and Mac users should use the native terminal while Windows users should use Windows PowerShell). When in a clean terminal window (independently on the actual working directory), you should prompt the following command:

```
ssh-keygen -t rsa
```

The system will propose the path where to save the key (if you change the path, you will need to specify where to find the key each time it is used). The system will then ask for a passphrase to protect the private key. This is a password to protect the private key in the host. If you set a passphrase, every time the key will be used, you will be asked for the passphrase. If you decide not to set a passphrase, leave the field blank.

3.1.1 Copying the Public Key

After copying the public key from your local host to the cluster, the system will be able, comparing the private and public keys, to recognize you as an authorized user of the system.

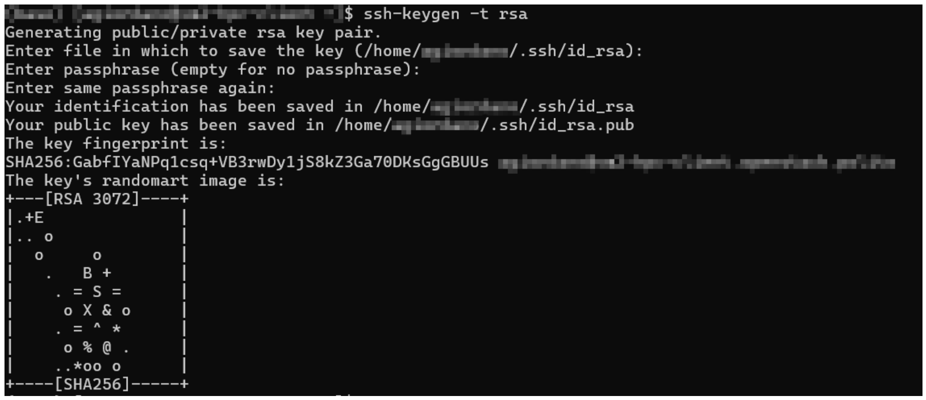

Figure 3.1: Output of the command `ssh-keygen`

From Linux/Unix/OS X systems

On Unix based systems, there is a built-in command to copy the public key to a given server; this command will take care of all the settings required on the server side:

```
ssh-copy-id [-i <FilePath>/KeyName.pub] username@{login_node}
```

Option `-i` is needed in case you have saved the public key in a non-default path.

From Windows systems

If you use the Windows PowerShell terminal, you need to manually copy the public key (`id_rsa.pub`) into the `.ssh` directory on the server. First, create the `.ssh` directory in your home on the cluster:

```
mkdir .ssh
```

Then return to the host shell and use the command `scp` to copy the public key to the cluster:

```
scp <$FilePath>\id_rsa.pub \
    username@{login_node}:/home/username/.ssh/authorized_keys
```

3.1.2 Login

To check whether the procedure was successful, verify the presence, on the server, of a file named `authorized_keys` within the `.ssh` directory. This file is where the public key is stored; it cannot have another name, since the system will not consider it.

If the file is present, you should be able to log in to the cluster without the need for a password. In case your private key is stored in a non-default directory, you should pass the correct path to the ssh command as:

```
ssh [-i <FilePath>/KeyName] username@{login_node}
```

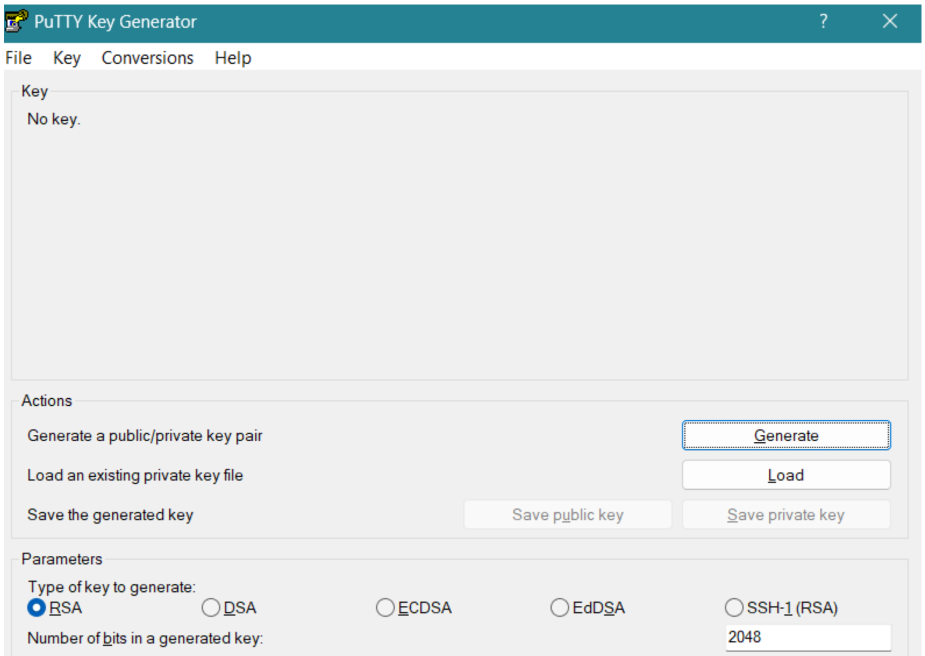

Figure 3.2: PuTTYgen homepage

3.2 Using PuTTY

From a Windows system, it is also possible to use PuTTY software to log in to the cluster.

3.2.1 Generating a Key with PuTTYgen

To generate a key, you need to install PuTTYgen (a separate application from PuTTY, available at http://www.putty.org, under "Download PuTTY", other binary files). Open PuTTYgen and select the type of key you want to generate (RSA), click on the `generate` button, then proceed by moving the mouse on the blank area that appears on the window to generate the key pair.

Once the key pair has been generated, save the private key on your device (the name and the path where you saved the key are irrelevant) and the public key on the cluster.

3.2.2 Copying the Public Key

To provide the public key on the cluster, you can either:

- Use the `scp` procedure described above (using a shell terminal).

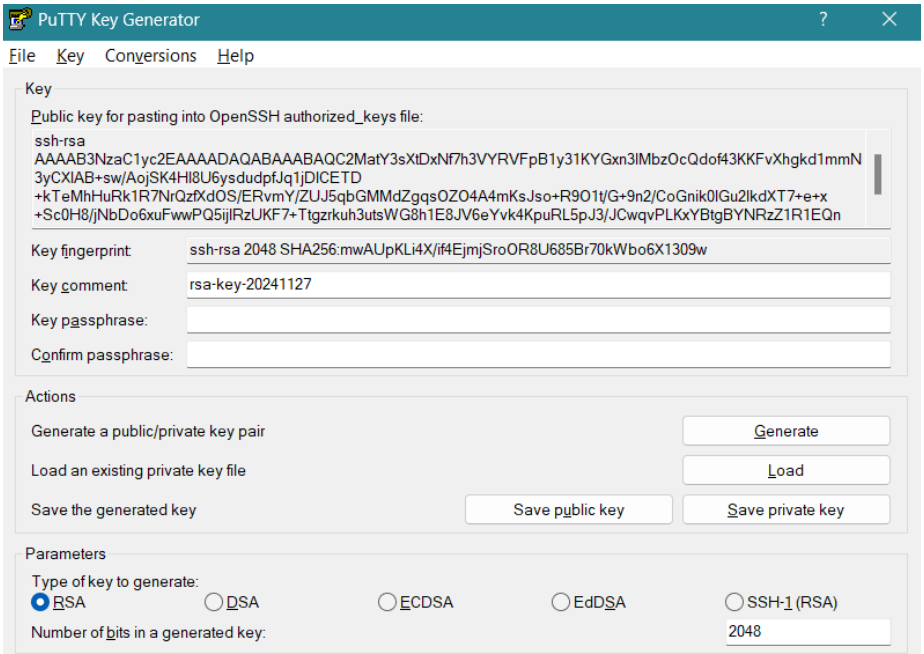

Figure 3.3: PuTTY gen has created the key pair

- Copy the public key and paste (if you are using a PuTTY terminal, right click) it into the `authorized_keys` file in the `.ssh` directory on the server.

3.2.3 Logging in with PuTTY

To log in to the cluster, follow this procedure:

1. Navigate to the section `Connection` → `Auth` → `Credentials`.
2. Add the private key that you created earlier (`key_name.ppk`).
3. Save the session and open it. PuTTY will notify you that it is authenticating using your key.

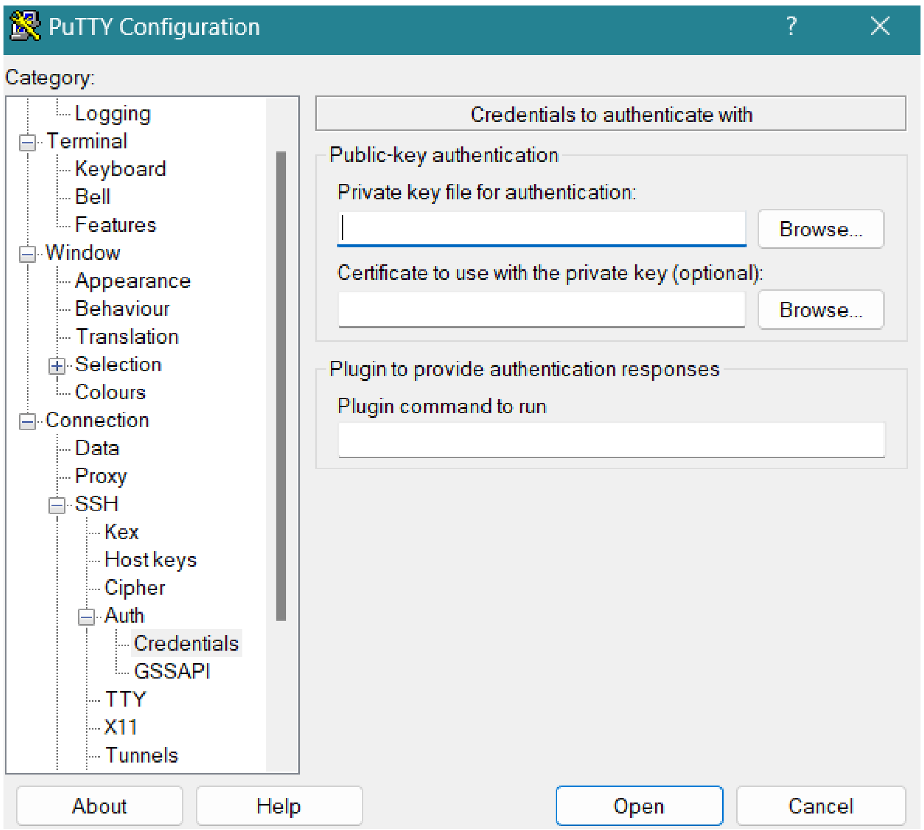

Figure 3.4: Set the private key on PuTTY

Chapter 4

HPC@PoliTO file systems

4.1 Home directories

HPC@PoliTO users have a home directory (`/home/username`) that is directly accessed upon login. Each user has a default quota of `1.5 TB` in their home directory, to store relevant data for their projects. Default quotas may be temporarily extended upon motivated request by contacting the HPC@PoliTO staff via email at hpc@polito.it).

As the Home storage system is implemented with a high-capacity NAS (see Section 1.3), IO performances are limited. For this reason simulations should not be run on data stored in the home directory. The home directory should only be used as a "warm" repository for data needed and produced within active projects.

Data stored in home directories will be automatically deleted after 6 months from the expiration of the relative account.

4.2 BeeGFS scratch: high performance storage

In addition to their home directories, each user has access to BeeGFS, a high-performance parallel file system. Users can find their BeeGFS space mounted at `beegfs-scratch/username`. Each user has a default quota of `5 TB`, subject to an automatic cleaning of all files older than 30 days. Temporary extensions may be granted upon motivated request by contacting the HPC@PoliTO staff via email at hpc@polito.it).

The BeeGFS scratch space is implemented with a state-of-the-art multi-node full-NVMe storage system, designed to achieve very IO high performances (see Section 1.3). Users should therefore use their BeeGFS scratch space to store "hot" data needed by current simulations, as well as their immediate outputs (waiting for post-processing).

Data should be stored in the BeeGFS scratch only for the strictly necessary time to run simulations and post-process their results. As there are no data protections implemented whatsoever, data consistency is never guaranteed on the BeeGFS scratch space.

4.3 File transfer between different file systems

Files can easily be copied (using the basic command `cp`) from your home directory (`/home/username`) to your BeeGFS scratch space (`beegfs-scratch/username`). To simplify this procedure, two environment variables have been created for all users:

- **$HOME**: the home directory;
- **$SCRATCH**: the BeeGFS file system;

Since different file systems are available to users as different directories, it is possible to move data between them by simply using standard Linux commands (e.g. `cp`) or `mv`. For example, to transfer `fileA` from the home directory to the BeeGFS scratch space:

```
cp $HOME/path-to-fileA/fileA $SCRATCH/path-to-store/
```

A practical application of file transfer between filesystems, within the context of a SLURM job, can be found in Section 5.3.1.

4.4 Best practices

In order to better exploit the features of each file system, users should carefully evaluate where to store their data. While the Home storage (**$HOME**) provides a large and persistent repository for users to store their project data, its read and write performances are not well suited for running simulations on it (*i.e.* jobs should not read or write data on this storage). The Home storage is therefore ideal to store "warm" data, which are not going to be accessed directly during jobs (*e.g.* simulation results that should be stored until the end of the current project).

On the other side, the BeeGFS storage (**$SCRATCH**) does not provide a persistent repository for data (files older than 30 days are canceled), but its high performances allow jobs to read and write directly on it without significant bottlenecks. It is therefore suited to store "hot" data that need to be accessed frequently during jobs, as well as their immediate outputs (*e.g.* raw results that need to be post-processed).

Chapter 5

Jobs submission

Running jobs on HPC@PoliTO's clusters is as simple as running the following command on the shell:

```
sbatch {script name}
```

To understand the meaning of this command and how to actually tailor it to your applications, read through this chapter.

5.1 Shared resources

Due to the limited resources available on the cluster and the high number of users, specific mechanisms must be implemented to ensure fair access to computing resources to all users, as well as an efficient use of the infrastructure.

This problem is mainly addressed using a batch scheduler, which takes care of collecting job requests from all users and assigns them the necessary computational resources when these are available.

5.2 SLURM scheduler

The SLURM scheduler manages all resources in the cluster, assigning computational resources to jobs when the former are available. When a job request is submitted, it is placed in a queue where it waits until the necessary resources are available, accordingly to the assigned priority. Jobs with higher priorities will start as soon as the requested resources are available.

5.2.1 Job parameters

All parameters passed to SLURM are written in the following format:

```
#SBATCH --<parameter>
```

A list of available parameters is reported for reference in Table 5.1. Some of them are optional; others may be used alternatively (as SLURM provides many options to fine tune submission request). In this guide, we suggest our preferred configurations while users are always free to adopt their scripts to better suit their workloads. According to HPC@Polito best practices, the following parameters should always be explicitly set:

- `--ntasks`
- `--nodes`
- `--ntasks-per-node`
- `--cpus-per-task`
- `--mem`
- `--time`
- `--partition`
- `--mail-type`
- `--mail-user`
- `--gres=gpu{N-GPUs}`, if the user is submitting a request for a job that requires the use of GPUs

Table 5.1 reports all the main parameters with a brief description of what they do. For a complete reference on SLURM parameters visit official SLURM webpage at `https://SLURM.schedmd.com/sbatch.html`.

Table 5.1: SLURM Job Options and Their Descriptions

Command	**Description**
`--ntasks`	Total number of tasks for the job.
`--nodes`	Number of nodes that will be used.
`--ntasks-per-node`	number of tasks per node, if used with the `--ntasks` option, the `--ntasks` option will take precedence.
`--cpus-per-task`	number of CPU per task.
`--mem`	Specify the real memory required per node, default units are megabytes.
`--time`	Indicates the hard run time limit, which is about the time that processes needs to reach the end. This value must be less than 10 days (< 240) for tasks on `global` partition.
`--mail-user`	Email where the information on task will be sent.

Table 5.1: SLURM Job Options and Their Descriptions

Command	Description
`--mail-type`	Events which cause an email notification (e.g., `ALL`). Valid type values are `NONE`, `BEGIN`, `END`, `FAIL`, `REQUEUE`, `ALL`.
`--partition`	Indicates the partition where the job has to be scheduled (default=`global`).
`--gres-gpu:{N.of.gpu}`	Generic resource scheduling, used for specifying the required number of GPU(s) per node.
`--nodelist={nodes}`	Request a specific list of hosts. The job will contain all of these hosts and possibly additional hosts as needed to satisfy resource requirements.
`--output`	the standard output is redirected to the file name specified, by default both standard output and standard error are directed to the same file.
`--error`	instruct Slurm to connect the batch script's standard error directly to the file name specified.
`--workdir={directory}`	If present, cause the execution of task using the {directory} as working directory (the path of {directory} can be specified as full path or relative path).
`--mem-per-cpu`	Minimum memory required per allocated CPU, default units are megabytes (default=1000).
`--exclude`	Explicitly exclude certain nodes from the resources granted to the job.
`--constraint`	Nodes have features assigned to them and users can specify which of these features are required by their job using this option, for example `--constraint="gpu"`. Available features can be shown in the output section `scontrol show node`.

5.2.2 Priority calculation

The main factors that contribute to the assignment of job priorities are the following:

- **age factor**: this factor changes dynamically and measures for how long the job has been waiting in queue; the longer a job waits, the higher the age factor contribution to that its priority will be.

- **fair share factor**: considers the amount of resources assigned to a user during the last month. Higher priorities will be assigned to users that used less resources in this period of time. The fair share factor is updated each time a job ends. In SLURM, fair share usage is normalized; it is determined as the ratio between the each user usage and the total cluster usage ($U_N = \frac{U_{user}}{U_{total}}$).

- **job size**: considers the amount of (CPU) resources requested by a job; it can be tuned to favor smaller or larger jobs depending on the cluster's policies. On Legion it is set to favor larger jobs.
- **partition factor**: it is used to prioritize jobs scheduled on specific partitions (*i.e.*: jobs scheduled on restricted partitions to prioritize them over jobs scheduled on global partitions).
- **TRES factor**: track the usage of specific resources. On Legion the tracked resources are: CPU, RAM and GPU. To each resource is assigned a specific weight, proportional to its scarcity, in order to consider the higher costs of scarce resources.

5.3 Submission script

To run jobs on the cluster, users must submit a request to the SLURM scheduler. Jobs should never be run directly from the shell: it is an abuse of our policies and shows a lack of respect towards your colleagues.

A submission request to SLURM is done via a batch file, which is composed of three main parts:

- job parameters
- application setup
- simulation run

The batch file is a script file executed by the shell. The first line of the file specifies the type of shell in use; on our systems always use a `bash shell`:

```
#!/bin/bash
```

A SLURM submission file should look like the example reported in figure 5.1.

5.3.1 Use the BeeGFS filesystem

Within the SLURM submission script, users can move their data between their home directory and their BeeGFS scratch space, before and after a simulation run. A typical use case, within a job, is the following:

- Move data needed from the current job, from $HOME/path-to-working-dir to $SCRATCH/path-to-simulation-dir;
- Change working directory to $SCRATCH/path-to-simulation-dir and run your simulation on the much faster BeeGFS storage system (faster IO typically means a faster job run);

```
#!/bin/bash
#SBATCH --job-name=job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user=name.surname@polito.it
#SBATCH --partition=global
#SBATCH --time=d-hh:mm:ss
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --output=job_name_%j.log
#SBATCH --mem-per-cpu=1G

module purge
module load [all the modules needed by the simulation, separated by a comma]

mpirun -np %ntask [PROGRAM]  -i [input_file].in > [output_file].out
```

Figure 5.1: Example of a sbatch script for SLURM

- Bring results back from $SCRATCH/path-to-simulation-dir to $HOME/path-to-working-dir .

Users may adapt the following example to their needs:

```
[....]
mkdir $SCRATCH/path-to-simulation-dir

cp $HOME/path-to-working-dir/* $SCRATCH/path-to-simulation-dir

cd $SCRATCH/path-to-simulation-dir

[RUN SIMULATION]

rsync $SCRATCH/path-to-simulation-dir/* $HOME/path-to-working-dir/
```

5.3.2 Simulation parameters

After setting the job parameters and loading the required software modules (see Chapter 6), the final step is to run the actual simulation. The command used to run a simulation depends on the specific application and how it is parallelized. All commands necessary to run an executable must be included in the script file. Once the scheduler allocates the requested resources, only the commands specified within the script will be run.

For example a generic `MPI-based` parallel application could be run as:

```
mpirun -np $ntasks [EXECUTABLE] -i [input_file] > [output_file]
```

Multiple simulations can be run sequentially within a single job request, specifying multiple execution commands in the submission file. This can be useful when running a series of simulations with different parameters (or configurations). They will be run sequentially following the same order as they have been defined in the submission script, with one starting as soon as the one before finishes.

5.4 Submit a job request

After editing the submission file, a job request can be simply submitted to the SLURM scheduler by running the following command:

```
sbatch {script name}
```

where *script name* is the name of the SLURM submission script (provide its path if the script is not in your current work directory).

5.5 Check the simulation status

After a job request has been submitted to the SLURM scheduler, it is possible to monitor its progress and/or check its status in the queue. SLURM provides several commands to help you track the state of your simulations, manage your jobs, and gather information about the cluster's resources. It is also possible to setup a mail alerting within the submission script (using the `--mail-user` and `--mail-type`), to be notified when a job starts and finishes.

A list of common SLURM commands to monitor jobs can be found in Table 5.2.

Table 5.2: Common SLURM Commands for Job Monitoring

Command	**Description**
squeue	Displays the status of jobs in the queue. By default, it shows all jobs, but you can filter by user squeue -u <username>, job ID squeue -j <job_id>
scontrol show job <job_id>	Provides detailed information about a specific job
sacct	Displays accounting information for all your jobs, can be filtered to show the information about a specific job: sacct -j <job_id>.
sinfo	Provides an overview of the cluster's partitions and nodes, including their current availability and status.
scancel <job_id>	Cancels a job by its ID. Use this command if you need to stop a running or pending job.

Chapter 6

Software ecosystem

The software on the HPC@PoliTO infrastructure can be managed in three different ways:

- **Centralized**: system administrators provide centralized installations of the most frequently used software suites. Users may load the required packages via the ***Environment Modules software***. If a particular software is not present between the available modules, and if it's deemed of interest for the larger community, its installation can be requested by users.
- **Local**: users are allowed to compile and install software in their home directory, without *sudo* privileges. They should specify a location within their home directory during the compilation/installation process. In this case, HPC@PoliTO provides the module ***Spack*** which takes care of handling paths and environment variables of locally installed software: as if they were private modules.
- **Containers**: users are allowed to execute their own containers, without the need of having *sudo* privileges, using Apptainer.

6.1 Environment Modules

The Environment Modules package allows one to dynamically change environment variables during a user session. For users, it appears as if they were "installing" these packages on-demand. In reality, they are changing environment variables to point to centrally installed software packages on the clusters. Using modules is recommended if the needed package and version is available.

In Table 6.1 are reported the most common options to handle modules:

Table 6.1: Environment Modules commands

Command	Description
`module avail`	Shows the modules available on the cluster.
`module load {module name}`	Load the specified module.
`module show {module name}`	Shows information about the specified module.
`module list`	Shows modules load in this user session.
`module unload {module name}`	Unload the specified module.
`module purge`	Unload all modules in this user session.